# PKI and TCP/IP based networks

L. Dostálek, L. Dubchak

UDC 004.056.5:004.738.5
BBK 32.973.202-018.2
D 70

Recommended by the Academic Council of the West Ukrainian National University, Ministry of Education and Science of Ukraine (record No. 3 of 27.11.2024).

**Reviewers:**

**Jan FESL,** Ph.D., Czech Technical University in Prague, University of South Bohemia, Czech Republic

**Serhii LUPENKO**, Dr. of Technical Science, Professor, Opole University of Technology, Poland

**Oleksandr ROMANYUK**, Dr. of Technical Science, Professor, Vinnytsia National Technical University, Ukraine

Dostalek L., Dubchak L. PKI in TCP/IP based networks – Ternopil: WUNU, 2026. – 309 p.

The textbook examines the main applications of computational mathematics problems in engineering practice, providing examples of solutions and fragments of programs in the modern algorithmic language Python. The manual presents computational tasks most commonly encountered in the design of computer systems and information technologies. The textbook primarily targets specialists in the field of technical sciences. Therefore, significant attention was given to translating all the described calculation methods into specific practical algorithms. It may also be of interest to undergraduates and postgraduates studying computer science, cybernetic systems, and information technology.

UDC 004.056.5:004.738.5
ISBN 978-966-654-892-7

This is a monograph dealing with the application of Public Key Infrastructure (hereinafter PKI) in computer networks based on the TCP/IP protocol family. The motivation for writing this monograph was my lecture ”Administration of Computer Networks and Services” at the Czech Technical University in Prague, where I focused on the application of security mechanisms in application protocols. From practice, I know that knowledge of these mechanisms is a basic prerequisite for network and computer administrators. This knowledge distinguishes professionals from trained amateurs, who, by randomly configuring systems, very often unsuccessfully attempt to reach their goal.

To ensure that the reader does not get the impression that PKI is the only mechanism for securing networks and computers, two other mechanisms are included in the publication:

- Kerberos, based on symmetric cryptography, which is used in networks based on Microsoft technology.
- Authentication and Key Agreement (AKA), which is mainly used for securing the SIP application protocol, and is utilized by everyone who gets their hands on a mobile phone.

# Contents


**1 Information security** **1**
- 1.1 Security policy and Information security policy . . . 2
- 1.2 Vulnerability of Computer Systems . . . 3
- 1.3 Information Leakage and Interception Channels . . . 6
- 1.4 Classification of Threats to Information Security . . . 9
- 1.5 International Approaches to Information Protection Policy . . . 10

**2 Basic Cryptographic Algorithms** **13**
- 2.1 Hash . . . 13
- 2.2 HMAC . . . 16
- 2.3 Replay attack, nonce . . . 16
- 2.4 Padding . . . 16
- 2.5 XOR . . . 17
- 2.6 Symmetric ciphers . . . 18
  - 2.6.1 Block ciphers . . . 18
    - 2.6.1.1 Block cipher mode of operation . . . 19
    - 2.6.1.2 CMAC . . . 20
  - 2.6.2 Stream ciphers . . . 21
  - 2.6.3 AEAD . . . 22
- 2.7 Asymmetric ciphers . . . 23
  - 2.7.1 RSA . . . 24
  - 2.7.2 Key exchange algorithms . . . 24
  - 2.7.3 ECC . . . 25
- 2.8 Hybrid encryption . . . 25
- 2.9 Electronic signature . . . 26
  - 2.9.1 Signature based on RSA algorithm . . . 26
  - 2.9.2 Signature based on (EC)DSA algorithms . . . 27
  - 2.9.3 Why is MAC not considered an electronic signature? . . . 28
  - 2.9.4 Composite electronic signature . . . 28

**3 Some other algorithms** **30**
- 3.1 Base64 . . . 30
  - 3.1.1 Base64/MIME . . . 31
  - 3.1.2 Base64URL . . . 32
- 3.2 Shamir's secret sharing . . . 32
- 3.3 Photuris . . . 32

3.4 AKA . . . 34

**4 Certificates and certification authorities** **38**
4.1 What is the defense? . . . 39
4.1.1 Does Bob have the corresponding private key? . . . 39
4.1.2 Did Bob generated his key pair using secure device? . . . 40
4.1.3 Conclusion . . . 40
4.2 Public key certification . . . 40
4.3 Proof of ownership of the private key . . . 42
4.4 Certificate . . . 43
4.4.1 Version . . . 44
4.4.2 Serial number . . . 44
4.4.3 Signature Algorithm . . . 44
4.4.4 Validity . . . 44
4.4.5 Distinguished Name . . . 45
4.4.6 Issuer . . . 47
4.4.7 Subject . . . 48
4.4.8 Public key . . . 48
4.4.9 Certificate extensions . . . 48
4.5 Guide to some certificate extensions . . . 50
4.5.1 Subject Key Identifier and Authority Key Identifier . . . 51
4.5.2 Private key usage period . . . 52
4.5.3 Key Usage . . . 53
4.5.4 Extended Key Usage . . . 55
4.5.5 Subject Alternative Name . . . 55
4.5.6 Certification policies . . . 56
4.5.7 Constrains . . . 57
4.5.7.1 Basic Constraints . . . 57
4.5.7.2 Name Constraints . . . 57
4.5.8 CRL Distribution Points (CDP) . . . 58
4.5.9 Authority Information Access (AIA) . . . 58
4.5.10 Certificate template . . . 58
4.5.11 Qualified Certificate Statements . . . 58
4.6 Certificate chain and trusted anchors . . . 59
4.6.1 Tree of certification authorities . . . 59
4.6.2 Certificate chain . . . 60
4.6.3 Verifying the validity of the certificate . . . 61
4.6.4 Chain validation starts from a trusted anchor! . . . 61
4.6.5 We are verifying the Certificate chain . . . 62
4.7 Certificate Lifecycle . . . 63
4.7.1 Certificate renewal . . . 64
4.7.2 Was certificate revoked? . . . 66
4.8 Certificate in Windows . . . 66
4.9 Certification and registration authorities . . . 67
4.10 Certificate request . . . 68

4.10.1 Information related to the certificate request . . . 69
4.10.2 Proof of possession of the relevant private key . . . 70
4.10.3 Self-signed certificate . . . 70
4.10.4 PEM . . . 71
4.10.5 PKCS#10 . . . 72
4.10.6 CRMF . . . 73
4.10.7 CMC . . . 73
4.10.8 CMP . . . 73
4.11 Exercises . . . 73
4.11.1 Root CA . . . 74
4.11.2 Sub-CA certificate . . . 75
4.11.3 Sensor ECDH certificate . . . 76
4.11.4 Sensor ECDSA Certificate . . . 76
4.12 Certificate revocation . . . 77
4.12.1 Certificate revocation request . . . 79
4.13 CRL . . . 79
4.13.1 CRL Extension . . . 81
4.13.2 CRL entry extension . . . 81
4.14 Exercises . . . 83
4.15 OCSP . . . 84

**5 ACME** **86**
5.1 External Account Binding . . . 86
5.2 Security of ACME communication . . . 87
5.3 Creating an account on the ACME server . . . 87
5.4 Certificate issuance . . . 88
5.4.1 Order . . . 88
5.4.2 Proof of control . . . 89
5.4.3 Sending a request for a certificate . . . 90
5.4.4 Certificate revocation . . . 90

**6 CMS** **91**
6.1 Signed Data structure . . . 93
6.1.1 Signer Info structure . . . 94
6.1.2 Certificates export . . . 96
6.2 Enveloped Data . . . 96

**7 Timestamps** **98**
7.1 What is time? . . . 99
7.1.1 Length of the Day and the Second . . . 100
7.1.2 Leap Seconds and UTC . . . 101
7.1.3 Time Zones and Daylight Saving Time . . . 101
7.1.4 Computer Time . . . 101
7.1.5 Time resources . . . 102
7.1.6 Time Providers . . . 102
7.2 TSA . . . 103

7.2.1 Timestamp Protocol (TSP) . . . 105
7.2.2 Transport protocols . . . 106
7.2.3 Timestamp request . . . 106
7.2.4 Timestamp . . . 106
7.2.5 Timestamp verification . . . 108
7.2.6 Timestamp validity . . . 108
7.2.7 What is not a timestamp? . . . 109

**8 HTTP 110**
8.1 Messages and frames . . . 111
8.1.1 HTTP/1.1 . . . 111
8.1.2 HTTP/2 . . . 111
8.1.3 HTTP/3 . . . 114
8.2 Message format . . . 115
8.3 An example of communication . . . 116
8.4 Headers compression . . . 118
8.5 HTTP URI . . . 118
8.6 Intermediaries . . . 119
8.6.1 Proxy . . . 120
8.6.2 Gateway . . . 122
8.6.3 Reverse proxy . . . 123
8.6.4 Tunnel . . . 124
8.6.5 More intermediaries . . . 125
8.6.6 Intermediaries and HTTP/2 or HTTP/3 . . . 125
8.7 Methods . . . 125
8.8 Status codes . . . 125
8.9 Representation of Data and Metadata . . . 126
8.10 Message Context . . . 128
8.11 Content negotiation . . . 128
8.12 Protocol Upgrade . . . 129
8.13 HTTP authentication . . . 129
8.14 Cache . . . 130
8.15 Conditional requirements . . . 132
8.16 Cookie - HTTP state mechanism . . . 132
8.17 Third party cookies . . . 134
8.17.1 Tracking cookies . . . 134

**9 Kerberos 137**
9.1 Principle . . . 139
9.2 Pre-authentication . . . 140
9.3 Authentication by smartcard (PKINIT) . . . 141
9.4 Password-based encryption . . . 142
9.5 Proxy . . . 142
9.6 Forwarding . . . 143
9.7 Trust Between Realms . . . 145
9.8 Timers . . . 147

9.9 Ticket structure . . . 147
9.10 Examples of tickets . . . 149
9.11 SPNEGO protocol . . . 150

**10 SAML, OAuth, Open ID Connect and Identity Management 151**
10.1 SAML . . . 151
10.2 Identity Federation . . . 152
10.3 JWT . . . 153
10.4 JWS . . . 153
10.4.1 JWS Header . . . 154
10.4.2 JWS body . . . 154
10.5 OAuth 2.1 . . . 155
10.6 Open ID Connect . . . 158
10.7 Identity management . . . 160
10.7.1 RBAC . . . 160
10.7.2 Identity Management (information system) . . . 161

**11 QUIC 163**
11.1 Connection, packet, frame and stream . . . 165
11.2 QUIC packets . . . 166
11.2.1 Security of QUIC packets . . . 166
11.2.2 Long header packets . . . 167
11.2.3 Short header packets . . . 169
11.3 QUIC Frames . . . 170
11.3.1 CRYPTO and STREAM frame types . . . 170
11.3.2 PADDING and PING type frames . . . 172
11.3.3 ACK type frames . . . 172
11.4 Explicit Congestion Notification . . . 173
11.5 Address validation . . . 174
11.5.1 Address Validation Using Retry Packets . . . 174
11.5.2 Address Validation for Next Connections . . . 174

**12 TLS 175**
12.1 TLSv1.3 Architecture . . . 176
12.2 Perfect Forward Secrecy . . . 176
12.3 Handshake Protocol . . . 177
12.3.1 Key Exchange phase . . . 177
12.3.2 Server Parameters phase . . . 178
12.3.3 Authentication phase . . . 178
12.3.4 Application data . . . 179
12.3.5 Cryptographic keys . . . 179
12.3.6 Post Handshake authentication . . . 180
12.3.7 ClientHello a ServerHello messages . . . 181
12.3.8 Finished message . . . 183
12.4 AP . . . 183
12.5 RLP . . . 183

12.5.1 CCS . . . 184
12.5.2 RLP packet . . . 184
12.5.2.1 The insecure fragment in the RLP protocol . . . 184
12.5.2.2 The secured fragment in the RLP protocol . . . 185

**13 DTLS** **187**
13.1 Handshake protocol (HP) . . . 188
13.2 DTLS Record layer protocol . . . 188
13.2.1 Unsecured frames . . . 188
13.2.2 Secured frames . . . 188

**14 SCTP** **190**
14.1 SCTP association . . . 191
14.2 Packets and Chunks . . . 192
14.3 Streams . . . 193
14.4 Multi-homing . . . 194

**15 SIP** **195**
15.1 SIP messages . . . 197
15.2 SIP URI . . . 198
15.3 TEL URI . . . 198
15.4 URN . . . 199
15.5 SIP message format . . . 199
15.6 SIP registration (first part) . . . 200
15.6.1 Authentication . . . 201
15.6.2 SIP INVITE . . . 202
15.7 Underlying protocols . . . 205
15.8 SIP protocol security . . . 206
15.9 SIP and DNS . . . 206
15.9.1 NAPTR . . . 206
15.9.2 DNS ENUM . . . 207
15.9.3 SIP URI translation . . . 208
15.10 Network architecture . . . 208
15.11 SBC . . . 210
15.12 IMS . . . 211
15.13 Roaming . . . 214
15.13.1 SIP roaming . . . 215
15.13.2 IP roaming . . . 216
15.14 SIP registration (second part) . . . 216
15.15 Lawful Interception . . . 217

**16 H.248** **219**
16.1 Terminations . . . 219
16.2 Contexts . . . 220
16.3 Commands . . . 221

**17 SDP** **222**

**18 RTP/RTCP** **225**
18.1 RTCP . . . 227
18.2 SRTP . . . 228
18.3 SRTCP . . . 230

**19 Mail** **232**
19.1 Webmail . . . 234
19.2 Mail delivery . . . 234
19.3 Securing e-mails . . . 235
19.4 Internet Message Format . . . 235
19.5 Mail address . . . 237
19.6 Internationalized email adress . . . 239

**20 SMTP** **240**
20.1 Internationalized mail support . . . 242
20.2 SMTP and DNS . . . 243
20.3 Authentication . . . 243
20.4 SASL . . . 245
20.5 SMTP over TLS . . . 245

**21 POP3** **247**
21.1 POP3 Capabilities . . . 249
21.2 POP3 commands . . . 250
21.3 Authentication . . . 251
21.4 POP3 over TLS . . . 251

**22 IMAP4** **252**
22.1 Not Authenticated State . . . 253
22.2 Authenticated state . . . 253
22.3 Selected state . . . 254
22.3.1 COPY . . . 254
22.3.2 SEARCH . . . 255
22.3.3 FETCH . . . 255
22.3.4 STORE . . . 256
22.3.5 EXPUNGE . . . 257
22.3.6 CLOSE . . . 257
22.3.7 LOGOUT . . . 257

**23 MIME** **258**
23.1 MIME header fields . . . 258
23.1.1 Mime-Version . . . 259
23.1.2 Content-Transfer-Encoding . . . 259
23.1.3 Content-Type . . . 260
23.1.3.1 Multipart . . . 261

23.1.3.2 Message . . . 263
23.2 S/MIME . . . 264
23.2.1 CMS and S/MIME . . . 267
23.2.1.1 SignedData . . . 267
23.2.1.2 EnvelopedData . . . 268
23.2.2 Certificate and S/MIME . . . 268
23.3 Application/pkcs7-mime . . . 269

**24 DNS and DNSSEC** **271**
24.1 DNS name syntax . . . 271
24.2 Internationalized domain name . . . 272
24.3 Reverse domains . . . 272
24.4 Zones . . . 273
24.5 Name server . . . 274
24.6 Resolver . . . 274
24.7 Recursive queries . . . 274
24.8 Resource records . . . 275
24.9 DNS protocol . . . 277
24.10 DNSSEC . . . 279
24.10.1 Zone Validation . . . 281
24.10.2 Proof of Non-Existence . . . 281
24.10.3 DNS over TLS or HTTPS . . . 281
24.10.4 Transport protocol and ports . . . 283
24.10.5 DNSSEC deployment . . . 283
24.10.6 DNSSEC Tools . . . 283
24.10.7 File /etc/hosts . . . 284
24.10.8 DNSSEC Security Considerations and Best Practices . . . 284
24.10.9 Conclusion . . . 285

# 1 Information security

Information security involves techniques aimed at protecting information by mitigating associated risks. Depending on its context, the concept of information security is viewed from various perspectives. In its broadest sense, information security refers to safeguarding the information environment to facilitate its creation, utilization, and advancement in the best interests of individuals, organizations, and the state.

The information lifecycle encompasses the spectrum of activities related to the creation, transformation, consumption, and disposition of information. As with paper documents, disposition can signify either the disposal of information or its permanent archiving, particularly if the information is deemed cultural heritage.

The following two supporting processes are associated with the information lifecycle:

- Creation and implementation of information systems, technologies, and their support.
- Development and application of means and mechanisms for information security.

A more detailed definition of information security can be given as follows: the state of protection required by individuals, society, and the government, ensuring its existence and progressive development irrespective of internal and external information threats.

It is worth noting that fulfilling information needs to any extent leads to acquiring knowledge about the surrounding world and its processes, thereby enhancing the awareness of individuals, society, and the government. This awareness dictates the level of accuracy in perceiving reality, influencing the validity of decisions and actions taken.

Information security varies depending on the types of threats, encompassing protection against:

- Forged information that influences individuals, society, and the state.
- Improper influence on information and resources by third parties.
- Protection of the information rights and freedoms of individuals and citizens.

Within information law, information security is one of the aspects considered in information legislation. It focuses on protecting the vital interests of individuals, society, and the state, highlighting threats to these interests and the legal mechanisms for eliminating or preventing such threats.

Components of information security:

- **Confidentiality:** This is a characteristic of information, which indicates that unautho-

rized users cannot access it. Privacy, achieved through methods such as encryption, is a means of ensuring confidentiality.

- **Integrity:** This refers to the inability of unauthorized users to modify the information.
- **Availability:** This denotes the property of information that is accessible to authorized users with appropriate permissions whenever they require it.
- **Non-repudiation:** Non-repudiation means that the author of a message cannot successfully dispute its authorship. It is primarily used to validate obligations according to the law. For instance, in EU law, a qualified electronic signature serves as a means of non-repudiation, acting as an alternative to a handwritten signature.

Individuals in IT often assume that information security is solely on IT measures, such as cryptography. However, this is not always feasible. Therefore, it is important not to overlook the existence of other layers of information security:

- Physical security including physical access to IT resources.
- Personal security including screening of persons and their training. This also involves assigning specific roles (user, administrator, operator, etc.) to individuals, including establishing rules that prohibit certain role combinations (segregation of duties).
- Organizational, including management of the organization operating IT resources.

## 1.1 Security policy and Information security policy

**An Information Security Policy (ISP)** is a document containing a set of requirements, rules, restrictions, and recommendations that regulate the procedures of information activities within an organization. Its aim is to achieve and maintain the state of information security within the organization. The ISP is an integral part of the organization's general security policies and should align with its fundamental principles.

The primary reason for implementing an ISP is often the requirement of regulatory bodies that govern the operational rules of the organization. Failure to have an ISP in place can result in punitive actions against the organization, including potential cessation of its activities.

Moreover, specific requirements or recommendations for improvement may be outlined in industry-specific or general, local, or international standards. External auditors, who assess the company's operations, often highlight non-compliance with these standards as comments. The absence of an ISP can lead to a negative evaluation, impacting the company's public metrics such as rankings and reliability levels.

The purpose of the ISP should be the implementation and effective management of an information security system, with the following objectives:

- Protecting the organization's **information assets**.
- Ensuring the **stable operation** of the organization.
- Minimizing **risks** to information security.
- Fostering **positive relationships** with partners, clients, and within the organization.

The primary objective of information security is to protect information assets against both external and internal threats.

A security policy is defined as a set of documented management decisions aimed at protecting information and its associated resources.

When formulating a security policy, it is advisable to adhere to the following principles:

- Ensuring the impossibility of bypassing protective measures.
- Strengthening the weakest link in the security chain.
- Prohibiting the transition to an open state.
- Minimizing privileges.
- Implementing the separation of duties.
- Establishing multi-level protection.
- Employing a variety of protective measures.
- Ensuring simplicity and manageability of the information system.
- Providing general support for security measures.

Based on implementation methods, all measures to ensure the safety of information systems are categorized into legal, moral and ethical, organizational (administrative), physical and technical (software and hardware).

## 1.2 Vulnerability of Computer Systems

A threat is an event that can potentially violate protected information. If the source of threats is human activity, then we talk about the violator, if objective phenomena, then we talk about man-made and natural sources of threats.

The result of the analysis of the computer systems vulnerability for the selected objects should be the development of separate models of the following types:

- Threat model – a description of threats and a schematic representation of the ways of their implementation for the object of protection.
- Model of man-made and natural sources of threats – an abstract formalized or informal description of factors and sources of threats for the object of protection.
- Model of a violator – an informal description of a criminal capable of implementing a threat (attack) on the object of protection.

Threats to the information protection system can be classified according to 9 features:

1. Based on the threat type:
    - Breach of the information confidentiality.
    - Violation of the information integrity (losses from such actions can be much greater than when confidentiality is violated).

- Performance violation (partial or full) of **Automated Information Processing Systems (AIPS)**.

2. According to the principle of influence on AIPS:
    - Using system subject (user, process) access to the object (data file, communication channel, etc.).
    - Using covert channels. A covert channel is a communication path that allows two interacting processes to exchange information in a way that violates the system security policy.

The influence based on the first principle is simpler, more informative, but it is easier to protect against it. Influence based on the second principle is distinguished by the difficulty of organization, less information, and the difficulty of detection and elimination.

3. According to the nature of the impact on AIPS:
    - Active threat that leads to a change in the state of the system and can be carried out either using access (for example, to a data set) or both using access and using hidden channels.
    - Passive threat, which is carried out by the user observing any side effects (for example, from the operation of the program) and analyzing them. An example of passive influence can be eavesdropping on the communication line between two network nodes. Passive influence does not lead to a change in the state of the system. It is always associated only with a violation of the confidentiality of information in the AIPS.
4. According to the error of used protection. Such an error can be due to one of the following reasons:
    - Inadequacy of the security policy to the real AIPS.
    - Errors of administrative management, which are understood as incorrect implementation or support of the adopted security policy of the AIPS.
    - Errors in algorithms, in the connections between them, etc., which occur at the stage of designing a program or a set of programs, in connection with which they can be used in a way that is not described in the documentation.
    - Errors in the implementation of algorithms (coding errors), connections between them, etc., which arise at the stage of implementation or debugging and which can also be a source of non-documentation.
5. According to the method of impact on the object of attack (with active influence):
    - Direct impact on the attack object, such actions are usually easy to prevent using access control tools.
    - Impact on the authorization system (including escalation of privileges).
    - Indirect influence (through other users).
    - ”Impersonation” – in this case the user assumes the authority of another user,

pretending to be him.

- ”Blind user” – when one user forces another to perform the necessary actions, and the latter may not be aware of them; a virus can be used for this.

6. According to the method of influence on AIPS:
   - Interactive mode.
   - Batch mode.
7. By the object of the attack.

   The following components of AIPS may be affected:
   - AIPS in general (system penetration), for this, as a rule, the method of ”masquerade”, interception or forgery of a password, ”hacking” and access to AIPS through the network are used.
   - AIPS objects — data or programs, system devices themselves, data transmission channels.
   - AIPS subjects are user processes and sub processes, a frequent case of such influence is the introduction of a virus by an attacker into the environment of a second process and its execution on behalf of this process.
   - Data transmission channels — data packets transmitted by communication channels and the channels themselves, channel listening and graph analysis (message flow, replacement or modification of messages in communication channels and on relay nodes, changing topology and network characteristics).
8. By the used means of attack (either standard software or specially developed programs can be used).
9. According to the state of the object of attack. The attack object can be in one of three states:
   - Preservation — influence on the object, as a rule, is carried out using access.
   - Transmissions — influence involves either access to fragments of transmitted information or simply listening using hidden channels.
   - Processing — the object of the attack is the user process.

Among the most common threats is unauthorized access (UA). This is when a user gains access to an object for which he does not have permission according to the organization’s adopted security policy.

In order to reduce the risk of UA, most protection systems implement the necessary functions with the help of an appropriate set of privileges. Illegal seizure of privileges is possible in the presence of errors either in the protection system itself, or due to negligence in managing the system and privileges.

Dangerous actions that can lead to a violation of the confidentiality, integrity and availability of certain components and resources of the AIPS can be grouped as follows:

1. Natural disasters.

2. External influences (connection to the network, interactive work, actions of intruders).
3. Intentional violations.
4. Unintentional errors (inputting the wrong command, data, using faulty devices, media, and neglecting some safety rules).

Types of threats that may appear as a result of dangerous actions:

- Disclosure (leakage) of information. For this type of threat, the objects of action are equipment (theft of media, connection, unauthorized use of resources), programs (unauthorized copying, interception), data (theft, copying, interception), personnel (transfer of protection information, disclosure, negligence).
- Violation of information integrity: equipment (connection, modification, special attachments, change of modes, unauthorized use of resources), programs (introduction of ”Trojan horses” and ”bugs”), data (distortion, modification), personnel (recruitment, bribery of personnel, ”masquerade”).
- System malfunction: equipment (mode change, disabling, destruction), programs (distortion, removal, replacement), data (deletion, distortion), personnel (dismissal, physical removal).

## 1.3 Information Leakage and Interception Channels

Modern systems for processing secret and confidential information include software and hardware components that may contain information leakage channels enabling interception. These channels can facilitate lawful interception (see section 15.15 Lawful Interception), result from system vulnerabilities, or be intentionally embedded by the author for potential future misuse.

Figure 1.1 illustrates the classification of information leakage and interception channels.

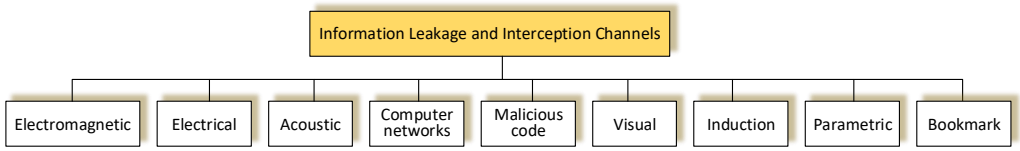



Figure 1.1 *Classification of information leakage and interception channel*

One of the primary requirements for comprehensive protection is a systematic approach. Therefore, when identifying technical channels of information leakage, it’s essential to consider the entire array of protection elements. This includes the core equipment of **Technical Means of Information Processing (TMIP)**: terminal devices, connecting lines, distribution and switching devices, power supply systems, grounding, and more.

In addition to the primary technical means directly involved in the processing and transmission of information resources, it is crucial to consider auxiliary technical means and systems (ATMS). These include telephone communication devices, security and fire alarm systems, wiring, wireless infrastructure, household appliances, and other conductive metal struc-

tures.

Based on the methods of intercepting information, as well as the distribution medium, channels of information leakage and interception can be categorized into electromagnetic, electrical, acoustic, computer network cables, visual, induction, parametric, implants, and malicious codes (refer to Figure 1.2).

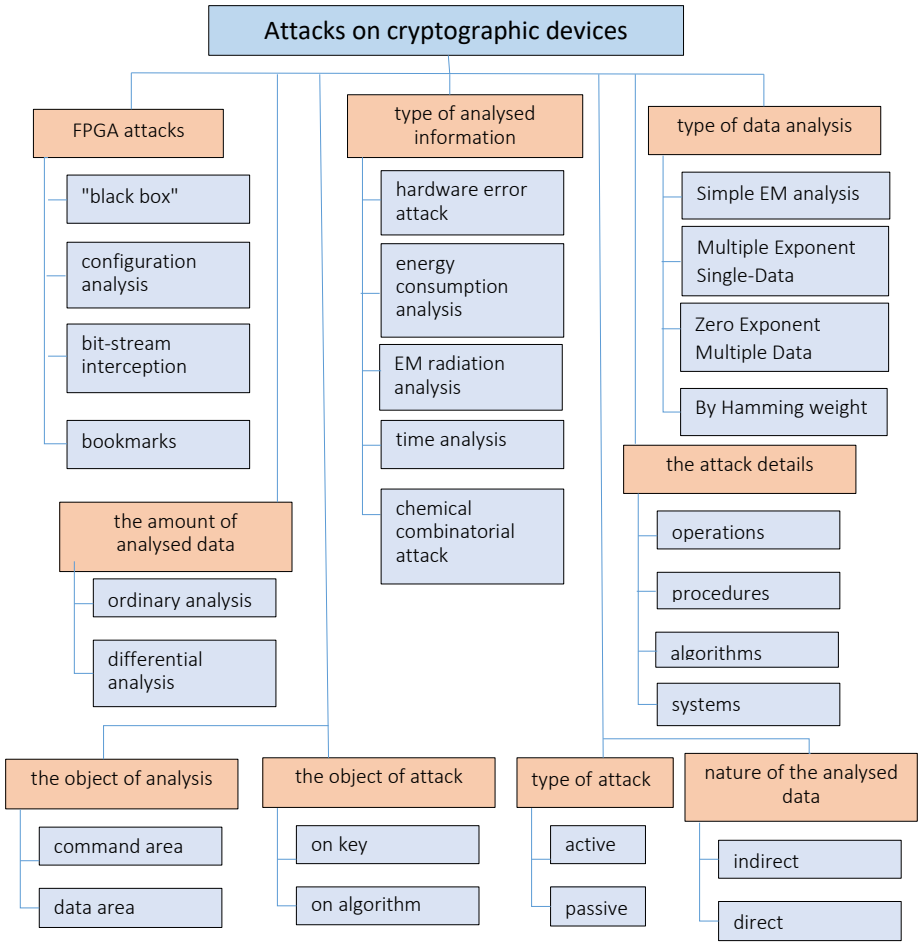



Figure 1.2 *Attacks on cryptographic devices*

**Electromagnetic leakage channels** are characterized by the following types of side radiation [1] [2]:

1. Electromagnetic (EM) radiation from TMIP elements (where the carrier of information is the electric current, with changes in current strength, voltage, frequency, or phase according to the information signal).
2. EM radiation at the operational frequencies of high-frequency generators in TMIP (resulting from external influences of the information signal on the generator elements, inducing electric signals that can benignly modulate their own high-frequency oscillations and radiate into the environment).

3. EM radiation at the self-excitation frequencies of low-frequency TMIP amplifiers (where self-excitation occurs due to random transformations of negative feedback into parasitic positive feedback, leading to the amplifier transitioning from amplification mode to self-generated signal modulation mode influenced by the information signal).

Potential causes of **electrical leakage channels** are as follows [1] [2]:

1. Electromagnetic (EM) radiation emissions from TMIP elements (occur when TMIP emit information signals, or due to galvanic connections between connecting lines of TMIP, side conductors, and ATMS lines).
2. Infiltration of information signals into the power supply network (can happen due to magnetic coupling between the output transformer of the amplifier and the power supply transformer, or due to uneven load on the rectifier device, resulting in changes in current consumption according to the information signal's pattern).
3. Infiltration of information signals into grounding links (possible with galvanic connections with various conductors beyond the control zone, including the neutral wire of the power supply network, screens, metal pipes of heating and water supply systems, metal fittings, etc.).
4. Information interception using embedded devices (these are miniature transmitters installed in TMIP, emitting radiation modulated by an information signal that is received outside the control zone).

Air, building structures, water supply, heating pipes, and other solid bodies can serve as environments for **acoustic channels** of information leakage and interception. Among acoustic channels, the following are distinguished [3] [4]:

1. Air (where the carrier of information is air, and miniature highly sensitive and narrowly directed. microphones connected to recorders or special mini-transmitters are used for interception).
2. Vibration (where the carrier of information is the vibrating structures of buildings within the control zone, and contact, electronic, and radio stethoscopes are employed to intercept information).
3. Electroacoustic (formed by converting acoustic signals into electrical ones, for example in telephones with electromechanical calls).
4. Optoelectronic (created by laser beam irradiation of thin reflective surfaces vibrating in the acoustic field, such as window glass, mirrors, paintings, etc.).
5. Parametric (formed as a result of the effect of an acoustic field on the elements of high-frequency generators and a change in the relative placement of circuit elements, conductors, chokes, etc., which alters signal parameters).

**Note:** Acoustic channels could be a source of leakage not only of speech information but also of data from mechanical locks, keys, information from printers or computer keyboards, etc.

Physical cables are allocated to a separate group, as modern information processing systems rely heavily on **computer networks**, which typically involve a sophisticated network of vari-

ous types of conductors. While the cable system itself does not contain active or non-linear elements, making it unable to emit ”side” radiation, it plays a crucial role in connecting all elements of the computer network. It serves to transmit network data while also acting as a receiver for parasitic signals and a medium for the transfer of side EM radiation [5].

There are several reasons for the emergence of leakage channels and information interception:

1. Radiation from active network equipment and computers (due to the heterogeneity of the cable system and grounding, information from the keyboard or monitor may inadvertently be emitted by the network system).
2. EM radiation of LCN traffic (various attacks are aimed at analyzing LCN traffic to determine critical areas of system load, making such information valuable to attackers).
3. Reception and re-radiation from nearby lines and devices (the cable system can function as an antenna for re-radiating signals from other devices, such as telephones, fax machines, modems, etc.).

**Visual leak channels** have been utilized since the pre-computer era of information security and remain in use today. Special technical means, such as optical and thermal radiation devices, are employed for this purpose. Documentation of observation results involves photography or filming of objects, while video bookmarks enable remote information collection.

The **induction information interception channel** does not require direct contact with communication channels, making it ideal for masking the information capture process. Modern induction transmitters can extract information from cables protected not only by insulation but also by double steel tape with tightly wrapped steel wire.

The **parametric channel** of information leakage is created by high-frequency irradiation of TMIP elements, wherein the magnetic field interacts with these elements, causing radiation of the electromagnetic field modulated by the information signal.

A **dropper** is a specialized tool to bring attack scripts or tools into an environment, deliver malicious codes, or transfer data between systems during lateral movement. In modern computer systems, both software and hardware droppers can be utilized, such as software keyboard spies or hardware micro radio transmitters. Detecting such embedded means directly impacts the security of computer information resources within the system.

## 1.4 Classification of Threats to Information Security

Information threats within an information system typically depend on its structure and configuration, the technology used for information processing, the state of the surrounding physical environment, human actions, and the nature of the information itself.

Of the many ways of classifying information threats, the most generalized (basic) is their classification according to the consequences of possible impact on information:

- Confidentiality breach threats.
- Integrity violation threats.

- Accessibility violation threats.

Threats to privacy are aimed at disclosing confidential or secret information. If these threats are realized, people without access rights can expose information.

The confidentiality breach threat occurs when there is a potential unauthorized access to certain confidential information, which is stored in a computer system or transmitted from one system to another.

Threats to the information integrity are aimed at its change or distortion, which leads to a violation of its quality or complete destruction. The integrity of information can be violated intentionally, as well as it can result from objective actions of the system environment surrounding. This threat is especially relevant for information transmission systems, computer networks and telecommunications systems. Intentional violations of the information integrity should not be confused with its authorized change, which is carried out by authorized persons with a justified purpose.

Availability threats (denial of service) are aimed at creating such situations when certain intentional actions either reduce the performance of the information system or block access to some of its resources.

For example, if one user of the system requests access to a certain service, and another one takes actions that lead to the blocking of this access, then the first user receives a denial of service. Blocking access to resources can be permanent or temporary. In addition, among the main threats to information may be the following:

- Theft, intercepting (copying of information).
- Destruction of information.
- Modification of information.
- Violation of availability (blocking) of information.
- Denying the validity of information (falsification).
- Imposition of false information.

Information remains available if it remains possible to obtain or modify it only in accordance with the established rules within a certain time.

Therefore, the threats, the implementation of which leads to the loss of information of the above-mentioned properties, are threats to confidentiality, integrity or availability of information, respectively.

## 1.5 International Approaches to Information Protection Policy

Networks and information systems have become integral to our daily lives, thanks to rapid technological advancements and increased social interconnections, including international data exchange. However, this progress has also led to a rise in cyber threats, necessitating flexible, coordinated, and innovative measures across all European Union countries and their partners. The scale, complexity, frequency, and consequences of these incidents continue to grow, posing serious threats to network and information system performance. Incidents can

disrupt economic activity, cause financial losses, erode user confidence, and inflict significant damage on the economy and society of the European Union. Consequently, cybersecurity preparedness and effectiveness are now more crucial than ever for the smooth functioning of the internal market. Furthermore, cybersecurity serves as a cornerstone for many mission critical industries seeking successful digital transformation and the full realization of the economic, social, and sustainability benefits of digitization.

Large-scale cybersecurity breaches and crises within the European Union require coordinated measures to ensure a prompt and effective response due to the high level of interdependence between different sectors. The presence of a reliable cyber network and information systems, availability, confidentiality and integrity of data are important for the security of the country and the protection of its citizens, enterprises and institutions from incidents and cyber threats. In addition, it contributes to increasing the confidence of citizens and organizations in the system's ability to provide and protect a global, open, free, stable and secure cyberspace based on the principles of human rights, fundamental freedoms, democracy and the rule of law.

Since threats to the security of network and information systems can have different origins, cybersecurity risk management strategies must be based on an approach to all potential dangers that may affect the availability, authenticity, integrity or confidentiality of stored, transmitted or processed data. Therefore, cybersecurity risk management strategies should also take into account the physical and environmental security of network and information systems, including measures to protect these systems from system failures, human errors, cyberattacks or natural disasters, in accordance with the European and international standards, such as those contained in the series ISO/IEC 27000.

In this regard, major and important subjects should, as part of their cybersecurity risk management measures, also pay attention to the security of human resources and have appropriate access control policies. These measures must be consistent with Directive (EU) 2022/2557 and the internal laws, regulations and standards of each individual state.

Each state adopts a national cybersecurity strategy, which provides strategic goals, the resources needed to achieve these goals, as well as relevant policies and regulatory measures to achieve and maintain a high level of cybersecurity. The National Cybersecurity Strategy includes:

- Goals and priorities of the cybersecurity strategy.
- Management structure to achieve goals and priorities.
- Governance framework that clarifies the roles and responsibilities of relevant stakeholders at national level, supporting cooperation and coordination at national level between competent authorities.
- Mechanism for identifying relevant assets and assessing risks.
- Identifying measures to ensure preparedness, response and recovery from incidents, including cooperation between the public and private sectors.
- List of various authorities and stakeholders involved in the implementation of the national cybersecurity strategy.

- Policy framework for improved coordination between competent authorities under Directive (EU) 2022/2557 in order to exchange information on risks, cyber threats and incidents, as well as on non-cyber risks, threats and incidents and to carry out supervisory tasks, if applicable.
- Raising the general level of awareness of citizens regarding cybersecurity.

As part of the national cybersecurity strategy, states, according to Directive (EU) 2022/2557, adopt an information protection policy, which consists of the following steps:

- Address cybersecurity in the supply chain of information products and services.
- Cybersecurity certification, encryption, and use of open source cybersecurity products.
- Vulnerability management.
- Maintenance of public availability, integrity and confidentiality of information on the global Internet.
- Promote the development and integration of relevant advanced technologies aimed at implementing state-of-the-art cybersecurity risk management measures.
- Promoting and developing cybersecurity education and training, cybersecurity skills, awareness raising and research and development initiatives, and guidance on good cyber hygiene practices and controls aimed at citizens, stakeholders and organizations.
- Supporting academic and research institutions to develop, improve, and facilitate the deployment of cybersecurity tools and secure network infrastructure.
- Appropriate procedures and information sharing tools to support the voluntary exchange of cybersecurity information between entities according to the law.
- Strengthening cyber resistance and the basic level of cyber hygiene of small and medium enterprises.
- Promote proactive cyber defense.

Such approaches make it possible to create an international information protection policy.

# 2 Basic Cryptographic Algorithms

This chapter provides an introduction to cryptographic algorithms commonly used in computer networking, explained in layman's terms. Specifically, we will examine hash functions and ciphers (encryption algorithms).

A hash is a fixed-length string that uniquely represents input data of arbitrary size. A hash function computes the hash; it is a **one-way function**, meaning the original input cannot be feasibly reconstructed from the hash. Unlike encryption functions, hash functions are not designed for reversible data transformation; they ensure data integrity rather than confidentiality.

A cipher is an algorithm that transforms plaintext into ciphertext to conceal its meaning. Encrypted data can be decrypted using the corresponding decryption function, which restores the original plaintext.

## 2.1 Hash

First, we will demonstrate the power of the hash function. Hash is a one-way function that generates a short string of constant length from an arbitrarily long text (data). The resulting string (hash) should effectively represent the original text. Hashing algorithms are not encryption algorithms (due to their one-way nature—generally lacking an inverse function), but they serve as high-quality "fingerprints" of data.

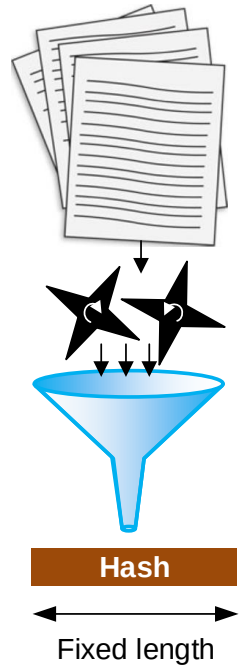



Figure 2.1 *Hash*

A one-way function is a computationally inexpensive function that transforms a message. However, it is computationally impossible to reverse the transformation and recover the original text of the message. A one-way function can be likened to breaking a glass—it is easy to break, but very difficult to put back together. High-quality one-way hash functions should yield a significantly different result when the original text is slightly changed or when additional text is inserted.

Since the hash is calculated from an arbitrarily long message and there are only a finite number of possible hash outputs, there exist an infinite number of original messages for a specific hash (pigeonhole principle). It is possible to find texts with the same hash (known as "col-

lision") for some outdated algorithms. Consequently, these algorithms should have been abandoned and replaced by others (e.g., algorithms of the SHA-2, SHA-3 class, etc.).

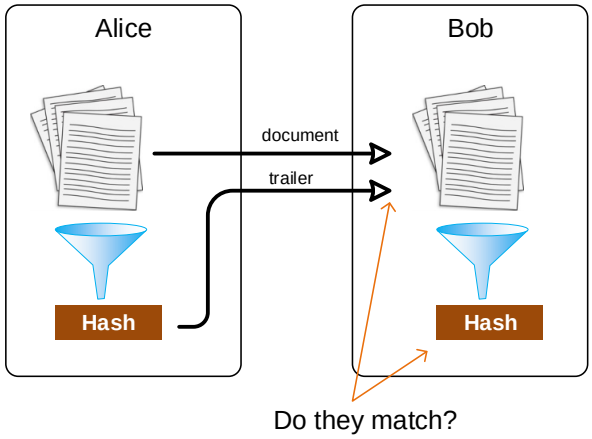



Figure 2.2 *Hash-based message integrity*

The problem arises in the case of very short messages, where the text of the original message can be easily found through brute force.

Hash functions are constructed using low-level computational operations (primarily bitwise operations and shifts) and therefore are generally computationally very fast and efficient. How can Alice use the hash in her communication with her beloved Bob? Well, she will use the hash as proof that the message was not altered during transmission from Alice to Bob, i.e., she will use the hash as proof of integrity of the message. Alice will send not only the data (document) of the message itself but will also attach a trailer containing a hash of the document (Figure 2.2).

Table 2.1 *Overview of hash algorithms [6]*

| Hash Algorithm | Output in bits | Published | Current Standard |
|---|---|---|---|
| **MD5** | 128 | 1991 | *Broken* |
| **SHA-1** | 160 | 1995 | *Broken* |
| **SHA-2** | | | |
| SHA-224 | 224 | 2001 | FIPS 180-4 |
| SHA-256 | 256 | | FIPS 180-4 |
| SHA-384 | 384 | | FIPS 180-4 |
| SHA-512 | 512 | | FIPS 180-4 |
| SHA-512/224 | 224 | | FIPS 180-4 |
| SHA-512/256 | 256 | | FIPS 180-4 |
| **SHA-3** | | | |
| SHA3-224 | 224 | 2015 | FIPS 202 |
| SHA3-256 | 256 | | FIPS 202 |
| SHA3-384 | 384 | | FIPS 202 |
| SHA3-512 | 512 | | FIPS 202 |

**Note:** In addition to the term "hash," we will also use the terms "digest" and "imprint":

- "Hash" can refer to both the algorithm and its output, while "digest" refers exclusively to the output produced by a hash function.

- ”Imprint” denotes the combination of a hash algorithm and its corresponding digest.

After receiving the message, Bob calculates the hash from the received message and compares his result with the hash from the trailer of the received message (i.e., with the hash calculated by Alice). If both hashes are the same, then the message has not been altered in transit, and Bob has verified the integrity of the message. This type of proof of data integrity preservation is used, for example, by network protocols to detect errors caused by line failures (failures of the physical layer of communication in a computer network). However, these protocols typically employ very simple one-way algorithms, so we refer to them as checksums rather than hashes.

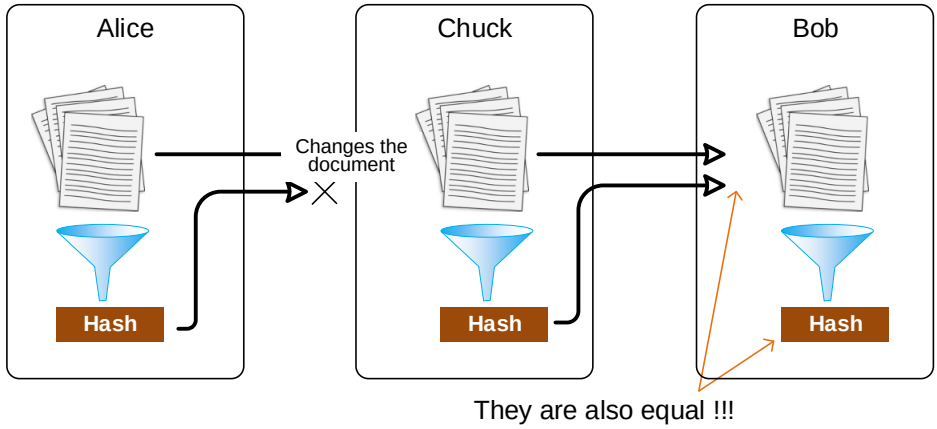



Figure 2.3 *Chuck's attack*

The hash calculation algorithm used by Alice and Bob is publicly described in the technical standard. This standard can be read by Chuck, who may modify the message from Alice to his advantage and add the hash from the modified message to the result (Figure 2.3). Alice would check hash of the message and think that the message if from Bob, while it is from Chuck instead.

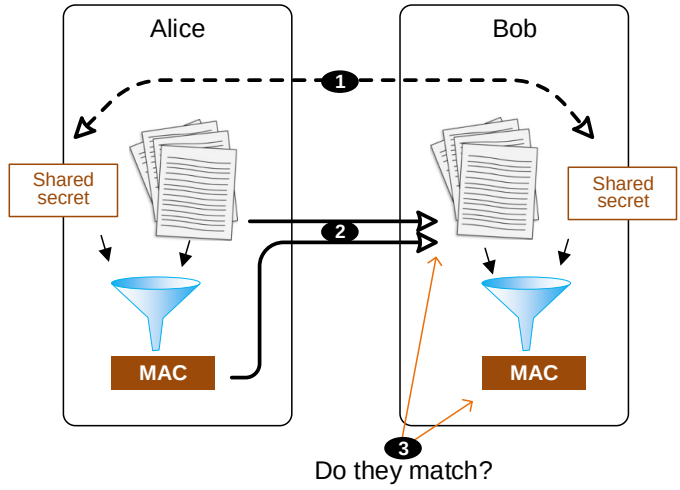



Figure 2.4 *Hash-based message integrity (HMAC)*

Generally, hashing cannot be used as a protection against an intelligent attacker. It is rather used as a protection to indicate a technical fault.

## 2.2 HMAC

Fortunately, Alice and Bob are familiar with Chuck, so during their secret meeting, they exchange a shared secret, which is shared only by Alice and Bob, thus unknown to Chuck. For instance, a text string is sufficient as a shared secret. However, the text string must not be too short, otherwise it could be determined using brute force.

From the moment Alice and Bob exchanged their shared secret, every time Alice sends a message to Bob, the hash will not only be calculated from the message but will also include the shared secret in the hash calculation, following a pre-agreed method (Figure 2.4). Since Chuck is unaware of the secret, he is unable to compute such a hash, preventing him from altering the message without Bob's knowledge.

A hash calculated not only from a message, but from a message concatenated with a shared secret, is often referred to as a hash-based integrity protection algorithm, HMAC (Hash-based Message Authentication Code). The method for calculating HMAC was specified in RFC 2104 [7], published in 1997. This standard is generally applicable to various hash calculation algorithms; thus, in practice, we employ algorithms such as HMAC-MD5, HMAC-SHA1, or preferably HMAC-SHA256, etc.

## 2.3 Replay attack, nonce

This mechanism has one major drawback: an attacker can eavesdrop and record the transmitted data sent by Alice to Bob, including the HMAC, and after a while, resend the entire transmission to Bob (i.e., repeat as the attack). This type of attack is referred to as a replay attack.

If Alice sends Bob a love letter, the attacker can at best make Bob happy. However, if Alice sends a payment order instead, the attacker could cause Bob to process the payment twice, resulting in Alice losing money (in banking, this is referred to as a *dual spend attack*).

Alice can defend against this attack by using various methods. For example, she can employ ascending numbering for her messages. If Bob receives a message with a lower-than-expected number, he can recognize it as a repeated old message. Similarly, the bank will refuse to process two orders with the same number.

Another defense is the use of a **nonce** (number only used once). A nonce is a sufficiently long random number (usually more than 16 bytes long) that Alice always adds to her message (to the transmitted data). This ensures that Alice is unlikely to send two identical messages and thus generate two identical MACs. However, Bob must check whether he has already received the message with the same nonce in the past.

## 2.4 Padding

Some algorithms expect input of a specific length. To ensure these algorithms are able to process any plaintext, it is required to lengthen the plaintext to the required length (or integer multiple of the required length). This process is called Padding.

Some padding schemes operate on bytes; some schemes operate on bits.

For example, algorithm AES expects that input is exactly 16 bytes long. If we have only 10 bytes of the plaintext, we have to add 6 bytes (16 - 10) of padding before we can use AES. Or, when our plaintext has 30 bytes, we need to add 2 bytes (32 is nearest bigger multiple of 16, 32 - 30). It is important to mention, that when the use of padding is expected, padding must be used, even when the plaintext length is multiple of the expected length. In that case the whole block of 16 bytes must be added. There always must be at least 1 byte (bit in bitwise padding schemes) of padding added.

Probably most common padding scheme is PKCS#7 padding. It works on full bytes and is quite easy to understand:

- Count number of bytes you are required to add (let us call it N).
- Append to plaintext $N$ times byte $N$.

**Example:**

Input text in hexadecimal `01 02 03 04 05 06 07 08 09 0A` has to be padded to length 16.

- Input text is 10 bytes long, the closest bigger multiple of 16 is 16, 16 – 10 = 6, so 6 bytes of padding must be added.
- That means padding will be `06 06 06 06 06 06`.
- Padded input text is `01 02 03 04 05 06 07 08 09 0A 06 06 06 06 06 06`.

## 2.5 XOR

Now, let us take a moment to revisit high school. There, we might have learned about logical operations like AND, OR, and NOT. However, they mostly omitted mentioning the logical operation XOR, which is also denoted by the operator $\oplus$. The logical operation $\oplus$ has a truth table, which is shown in Table 2.2.

Table 2.2 *Truth table of XOR ($\oplus$) operation*

| A | B | A $\oplus$ B |
|---|---|---|
| False – 0 | False – 0 | False – 0 |
| False – 0 | True – 1 | True – 1 |
| True – 1 | False – 0 | True – 1 |
| True – 1 | True – 1 | False – 0 |

Let us consider the XOR operation as a one-bit operation, for which interesting relationships can be applied:

$$(A \oplus B) \oplus B = A$$

and also

$$(A \oplus B) \oplus A = B$$

We will perform the XOR operation on bit strings A and B of the same length, bit by bit. Imagine A as a message and B as an encryption key. Then $(A \oplus B)$ will represent an encrypted message, which we decrypt by XORing it again with the encryption key B to obtain the original message A.

**Example:**

```
1 0 1 1 = A                    1 1 0 1 = A ⊕ B
0 1 1 0 = B                    0 1 1 0 = B
-----------                    -----------
1 1 0 1 = A ⊕ B                1 0 1 1 = A
```

## 2.6 Symmetric ciphers

But Alice and Bob may also desire to keep their communication secret to preserve the privacy of their relationship. Therefore, they encrypt their communication using symmetric ciphers (see Figure 2.5). To use symmetric encryption, Alice and Bob must exchange a secret encryption key beforehand, for example during their previous meeting (Figure 2.5, arrow 1). They will share this key in the same manner as they shared a shared secret, ensuring it remains inaccessible to third parties such as Chuck. In the case of symmetric ciphers, this key is referred to as the secret key. Colloquially, it is also known as a symmetric key.

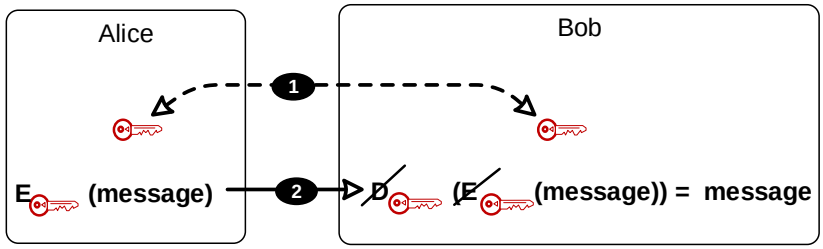



Figure 2.5 *A symmetric cipher*

Alice encrypts the message with a secret key. Bob then applies a decryption algorithm to the result, using the same secret key. Decryption reverses the encryption process, allowing Bob to retrieve the original message. In a sense, a symmetric cipher provides authentication. From Bob's perspective, only Alice could have encrypted the message, as only Alice possesses the secret key.

Symmetric ciphers are divided into block and stream ciphers.

### 2.6.1 Block ciphers

Block ciphers encrypt/decrypt data in blocks of fixed length. If the incoming data is shorter, then padding is necessary to match the block length.

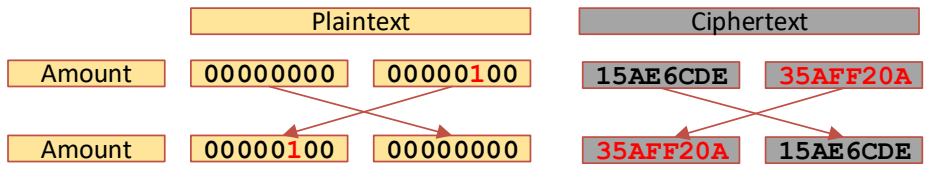



Figure 2.6 *Exchange of encrypted blocks*

#### 2.6.1.1 Block cipher mode of operation

But an encrypted message can consist of several blocks. The way in which the encryption of individual blocks is linked to each other is called a cipher mode of operation (more precisely, an encryption scheme). The simplest mode is ECB (*Electronic Codebook*) , where individual blocks are encrypted separately, independently of other blocks. In other words, ECB does not bind the encrypted blocks to each other in any way.

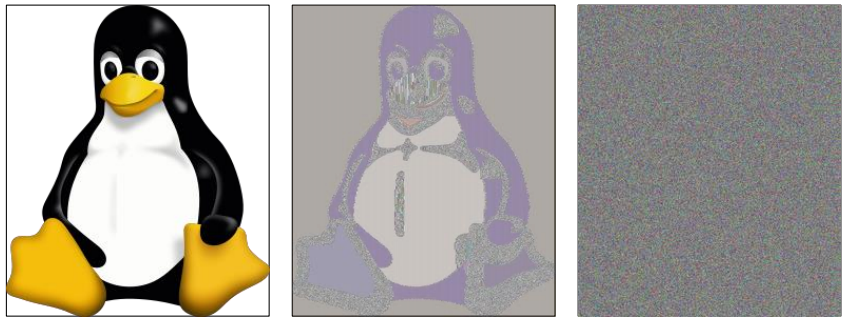


Figure 2.7 *Bitmap, bitmap encrypted using ECB mode, and bitmap encrypted using CBC or CTR modes*[1]

Even if the attacker cannot decipher the encrypted text, in the case of ECB mode, they can still launch an attack by altering the order of the individual encrypted blocks. This alteration could result in unintended consequences, such as changing the value of a transaction amount (Figure 2.6). Figure 2.7 is also noteworthy, as it illustrates an example of ECB mode encryption of an image in bitmap format, highlighting large areas of the same color.

A significant consideration is what information to include in the first encrypted block. Without including any information, two coincidentally identical texts would yield identical cipher texts. Although the attacker would not be able to decipher the original (unencrypted) text, the fact that it is the same message might not be concealed from him. Therefore, before encrypting the message, random data (known as initialization vectors) are often generated and included in the encryption of the first encrypted block.

[1]Source Wikipedia:

- Penguin Tux, the Linux Mascot; Authors: Larry Ewing, Simon Budig, Garrett LeSage; License: Creative Commons CC0 1.0 Universal Public Domain Dedication.
- Tux encrypted ecb.png and Tux encrypted ctr.png, Author: RFL890, License: Creative Commons CC0 1.0 Universal Public Domain Dedication.

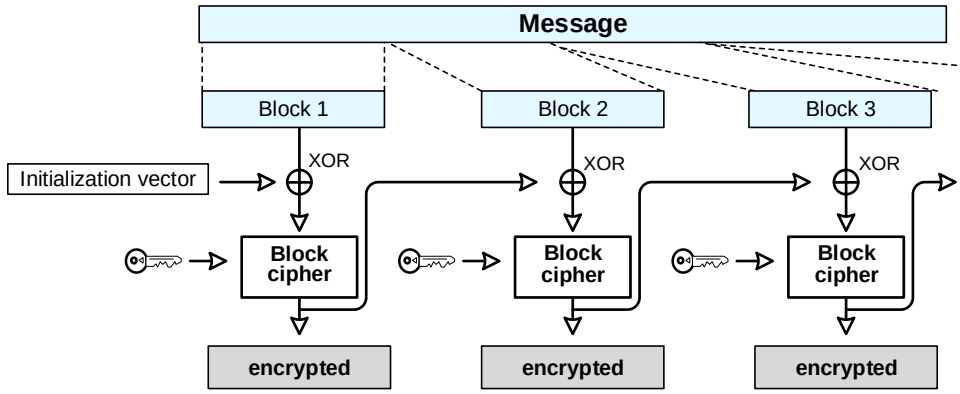



Figure 2.8 *Block cipher in CBC mode operation*

A very well-known mode of block ciphers is the CBC mode (Figure 2.8). In this case, the initialization vector is first applied to the first block through XOR operations. The result of the previous block's encryption is then applied to the next block before it is encrypted through XOR operations, and so forth. The CBC mode was published in 1976 and is not highly recommended today, as AEAD algorithms are preferred.

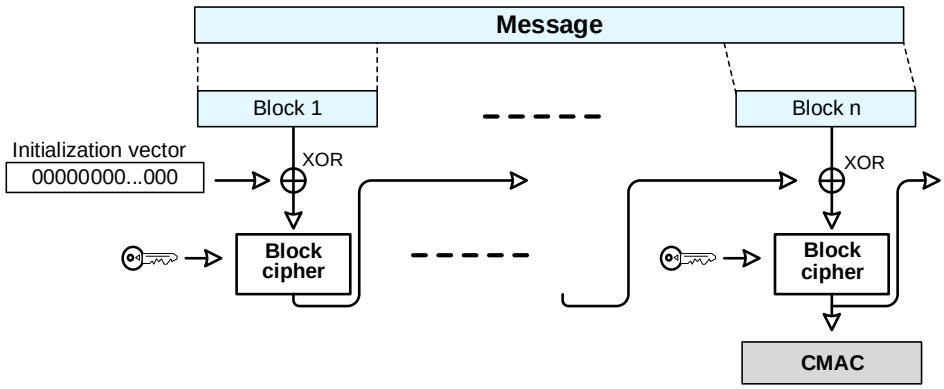



Figure 2.9 *CMAC*

### 2.6.1.2 CMAC

If we examine Figure 2.8, we realize that the last block essentially serves as a hash of the entire encrypted message, depending on the value of the secret key.

In Figure 2.9, the principle of the CMAC (the abbreviation is derived from CBC-MAC) algorithm is illustrated, which serves as an integrity protection algorithm similar to HMAC. This algorithm employs binary zeros as an initialization vector. (The figure does not address the issue of padding the last block if the message length is not divisible by the encryption block length.)

NIST has recommended the CMAC algorithm as one of the algorithms for integrity protection.

### 2.6.2 Stream ciphers

The concept of a stream cipher stems from the property of the logical XOR operation:

$$(A \oplus B) \oplus B = A$$

The foundation of a stream cipher lies in a generator of pseudo-random data flow, whose output is deterministically dependent on the value of the secret key.

The output from the generator (A) is then XORed bit by bit with the message (B) that we want to encrypt. The result is the encrypted message.

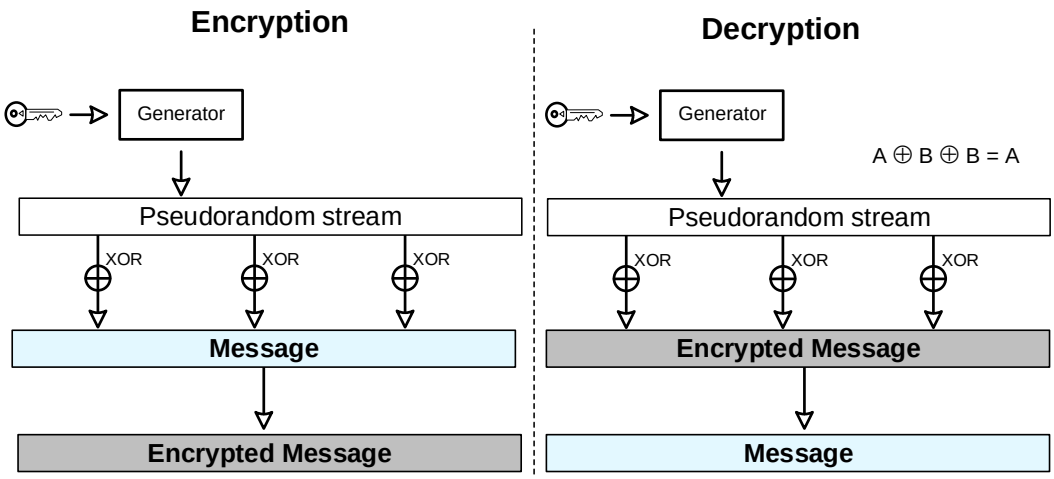



Figure 2.10 *Stream cipher*

During decryption, a pseudo-random flow (B) is applied to the encrypted message (i.e., A ⊕ B). The result is the decrypted message (A). It is crucial to use the same generator with the same secret key.

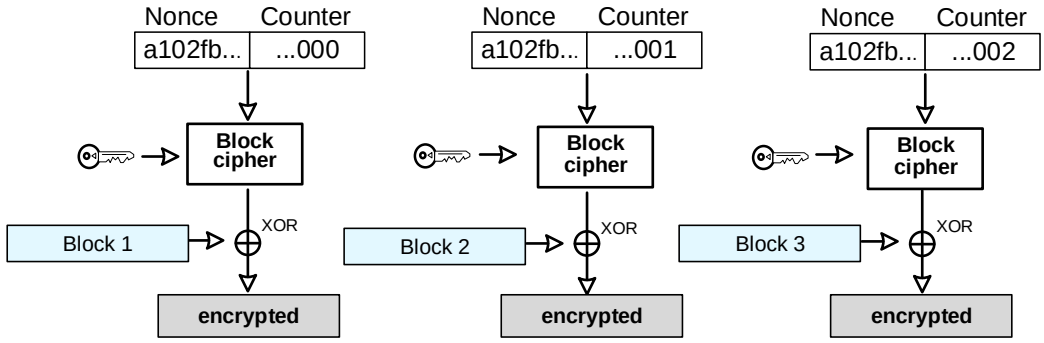



Figure 2.11 *CTR mode of operation*

A slightly different approach utilizes the CTR (Counter) mode of block ciphers (see Figure 2.11). In this case, a pseudo-random stream of data is generated using a block cipher. First, a nonce is generated—a random number serving as an initialization vector.

The encrypted message is then divided into blocks. A block cipher-encrypted nonce + counter

pair is applied to each block using the XOR operation (the + operation is a function specified by the standard). A random Nonce number is also referred to as an Initialization Vector (IV).

A nonce is a random unique string for a given encryption, while the counter is typically incremented by one as individual blocks enter the encryption process.

The AES algorithm in CTR mode is used, for example, in the encryption of multimedia communication between a mobile device and a base station in LTE networks.

### 2.6.3 AEAD

Current versions of network protocols (e.g., TLSv1.3) abandon classical modes of block ciphers in favor of AEAD (Authenticated Encryption with Associated Data). AEAD is an encryption scheme that ensures:

- Confidentiality: without knowledge of the secret key, the encrypted text cannot be understood.
- Authenticity and integrity: encrypted data contains an authentication tag (similar to HMAC or CMAC).
- Some schemes allow for associated data, which is not confidential but is necessary to ensure its integrity. A typical example is a network protocol packet (e.g., TLSv1.3), where its integrity must be ensured, but the header must also be readable by some active network elements on the way for optimal packet routing (e.g., for HTTP/3 load balancing).

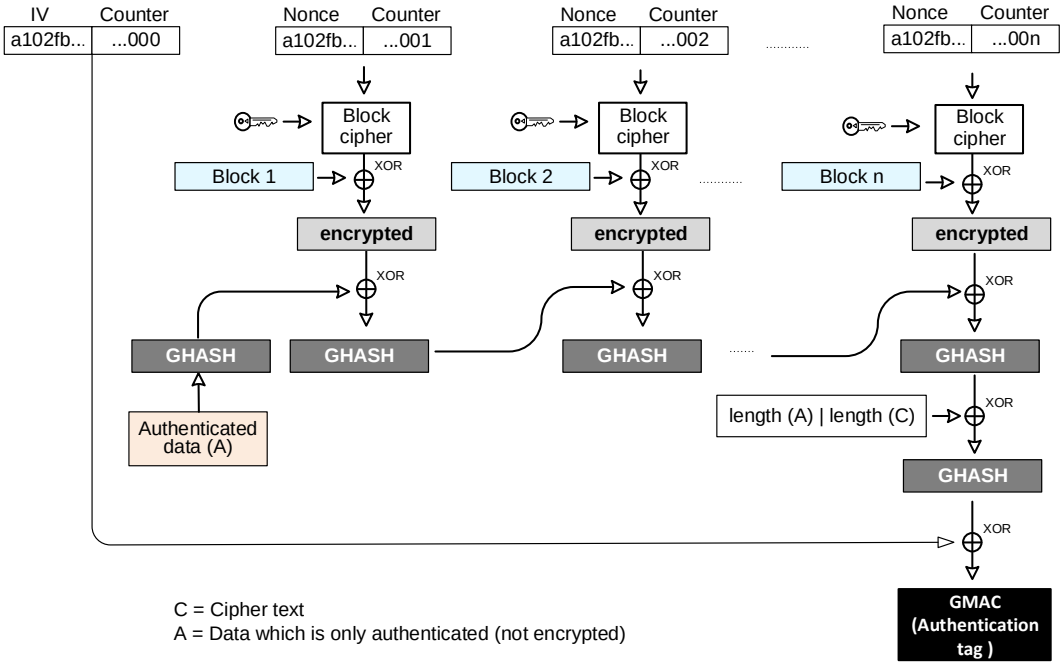



Figure 2.12 *GCM*

An example of AEAD is GCM mode (*Galois/Counter Mode*) which defines its own hash func-

tion called GHASH. The process (Figure 2.12) includes: data to be encrypted (C) and associated data (A) that will only be authenticated and have their integrity protected. It is based on the principle of a stream cipher. However, a GHASH of the authenticated data is additionally XORed onto the first encrypted block. The result enters the next step of the process. After encrypting the last block, the length of the authenticated data and the length of the encrypted data (Cipher text C) are added to the result using the XOR operation. Finally, the nonce creates an Authentication tag GMAC.

In table 2.3 you can see overview of most commonly used modes of operations for block ciphers. For each mode you can see, whether it functions similarly to stream ciphers and whether it is AEAD mode.

Table 2.3 *Block cipher modes of operations overview*

| Mode | Full name | Stream-like | AEAD |
|---|---|---|---|
| `ECB` | Electronic Code Book | no | no |
| `CBC` | Cipher Block Chaining | no | no |
| `CTR` | Counter | yes | no |
| `GCM` | Galois/Counter | yes | yes |
| `CCM` | Counter with CBC/MAC | yes | yes |

## 2.7 Asymmetric ciphers

Another type of cipher is asymmetric cipher. These ciphers do not utilize a single secret encryption key shared between the sender and the receiver. Instead, they always use a pair of keys:

- A private key that the user protects as a highly valuable asset (e.g., stored on a smart card).
- A public key that the user publishes to facilitate communication with partners.

The fundamental feature of cryptography based on asymmetric algorithms is that it is relatively easy to perform an operation using a public or private key. However, without the private key, reversing the operation performed with the public key is extremely difficult.

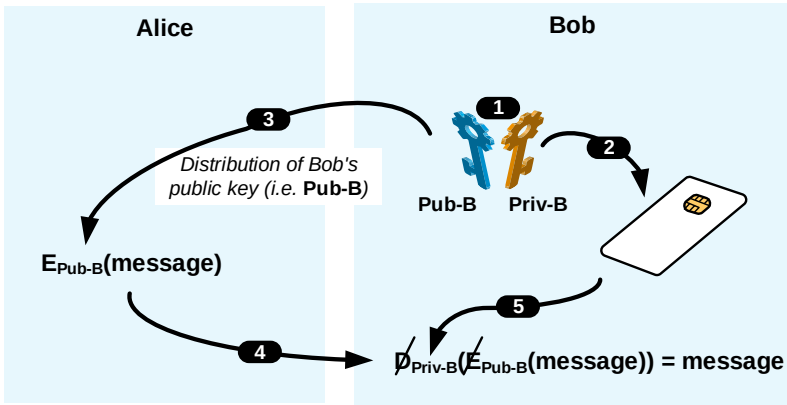



Figure 2.13 *Asymmetric encryption*

### 2.7.1 RSA

Probably the most famous asymmetric algorithm is the Rivest–Shamir–Adleman algorithm (RSA algorithm). If Alice wants to encrypt a message to Bob with an asymmetric RSA cipher, then:

- Bob, the recipient of the message, must generate a pair of keys: a public key (Pub-B) and a private key (Priv-B).
- Bob stores his private key in a trusted key store, such as on a disk or a chip card. The private key is an asset of Bob's that must be securely protected.
- Bob distributes his public key (Pub-B) worldwide. He can easily send his public key to Alice via Chuck.
- After receiving Bob's public key, Alice encrypts the message to Bob using Bob's public key (Pub-B).
- Bob, the receiver, decrypts the received encrypted message with his private key (Priv-B) and obtains the original message ("Decryption cancels an encryption").

The length of the encryption keys for the RSA algorithm is considered secure, if it is greater than or equal to 2048 bits today. It is generally understood that arithmetic operations with such long keys are extremely computationally intensive, which is why the RSA algorithm is being phased out as the length of the keys grows.

### 2.7.2 Key exchange algorithms

Another asymmetric algorithm is the DH (*Diffie-Hellman*) algorithm (Figure 2.14). It is not suitable for electronic signatures or asymmetric encryption, but rather for the secure establishment of shared secrets from which secret keys can be derived.

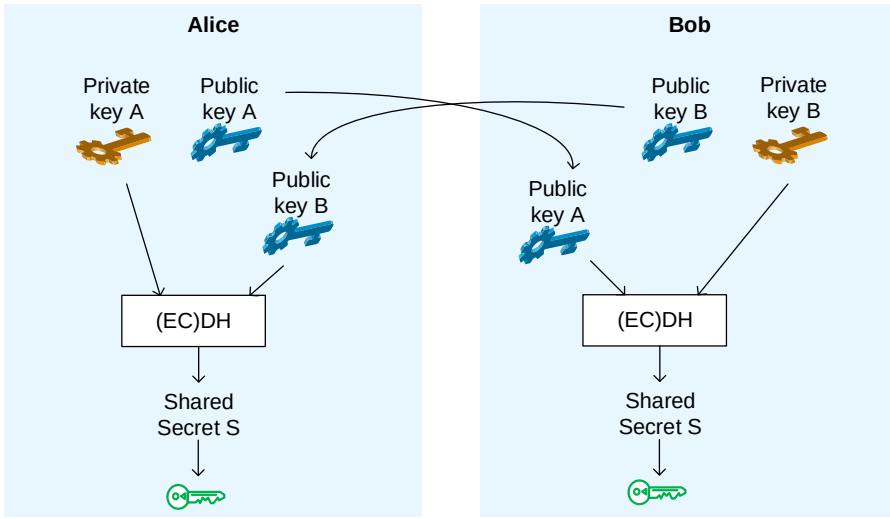



Figure 2.14 *Secure key establishment*

In order to communicate using symmetric cipher, Alice and Bob first agree on a secret key using the (EC)DH algorithm without the need for a secret meeting. Both parties first generate a pair of keys: a public and a private (EC)DH key. They then exchange their public DH keys with each other (e.g., freely via the Internet).

Subsequently, both parties are able to calculate the same shared secret (S) from the knowledge of their own public and private DH keys, as well as the public (EC)DH key of their counterpart. From this shared secret, it is then possible to derive a symmetric (secret) encryption key through some transformation, which is used for encrypting their communication, e.g., with the AES algorithm.

### 2.7.3 ECC

Currently, cryptography based on elliptic curves (*Elliptic Curve Cryptography*) is widely used. Its advantage is that it employs significantly shorter keys.

These algorithms do not facilitate encryption but enable key exchange. As this key exchange uses same principle as Diffie-Hellman (DH) algorithm, we refer to these algorithms as Elliptic-curve Diffie-Hellman or ECDH for short. Later, we will discover that they also support electronic signatures.

There are various algorithms available. When specifying the algorithm, we must include not only the algorithm but also the parameters of the curve used.

## 2.8 Hybrid encryption

Regardless of the length of the keys, it is generally true that asymmetric encryption algorithms are computationally much more demanding than symmetric algorithms.

*For example, an RSA key with a length of only 1024 bits is represented in the decimal system as a number with more than 300 digits, making it impractical for standard microprocessor operations.*

The solution to this problem is hybrid encryption (Figure 2.15). The sender encrypts the message with a random (symmetric) master secret key, which is a fast operation. Then, for each recipient, they enclose a Recipient structure containing, among other things:

- In the case of the RSA algorithm, the master secret key encrypted by the recipient's public key.
- In the case of the (EC)DH algorithm, the master secret key encrypted by a secret key derived from the shared secret of the (EC)DH algorithm. Therefore, the structure of the Recipient info must contain the originator's and recipient's public keys (or public keys IDs).

So, only the master secret key is encrypted asymmetrically. This process is fast and effective. It has another positive effect. If we send a message to several recipients, we encrypt it only once with a random master secret key, and for each recipient, we enclose an encrypted master secret key. This is useful, for example, for email encryption (S/MIME).

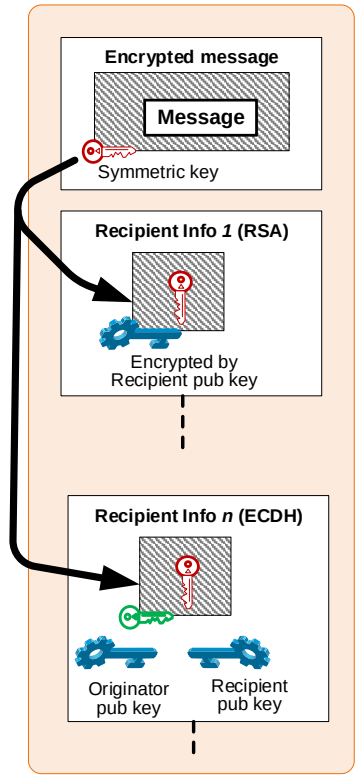



Figure 2.15 *Hybrid encryption*

## 2.9 Electronic signature

An electronic signature provides proof of authenticity based on ownership of a private key.

An electronic signature (also called digital signature) is a mechanism that ensures proof of non-repudiation of data (authenticity of documents).

### 2.9.1 Signature based on RSA algorithm

Electronic signatures based on the RSA algorithm are created and verified in the following steps (see Figure 2.16):

1. The signer calculates the hash of the document.
2. The signer encrypts the resulting hash with his private key. A hash of a message encrypted with a private key is called an electronic signature of the message.
3. The signer sends the document and its electronic signature to the verifier.

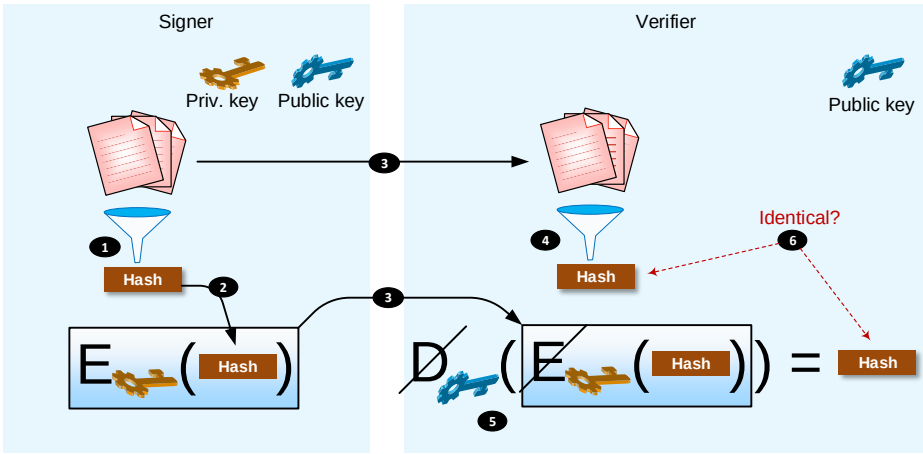



Figure 2.16 *Creation and verification of an electronic signature using the RSA algorithm (E - encryption, D - decryption)*

The verifier then confirms the authenticity of the electronic signature:

4. The verifier independently calculates the hash from the received message.
5. The verifier checks the received electronic signature using the signer's public key.
6. The verifier compares the result obtained in step 4 with the result obtained in step 5. If they are the same, then the electronic signature could only have been created by the one who owns the signer's private key – that is, the signer. Additionally, this fact proves that the message was not altered during transmission, ensuring message integrity.

Unlike encryption, the RSA electronic signature uses the key of the person signing the message (not the recipient, as in encryption). Therefore, we tacitly assume that our algorithm allows ”decrypting” with the private key and then ”encrypting” with the public key interchangeably. An algorithm, which enables such a process, is the RSA algorithm.

***Note:*** *It is not considered secure to apply encryption directly to digest of data (so called textbook RSA). Before anything else, this data digest must be processed in specific way. For example you can take sequence of bytes* `02 00 [some random bytes] 00` *and prepend it to data digest. This preprocessing is called signature scheme. This particular signature scheme is referred to as RSA-PKCS#1 v1.5. However, it is worth noting that the RSA-PSS signature scheme is now recommended. In RSA-PSS, instead of encrypting the hash itself with a private key, we encrypt a hash masked by a random function. This ensures that the result varies each time, making it impossible to deduce the original hash. Naturally, signature verification in RSA-PSS is more intricate than depicted in Figure 2.16*

### 2.9.2 Signature based on (EC)DSA algorithms

Thus, we can say that the signature is a function $F()$ with parameters consisting of the signed data, private key, and the signature scheme. We verify the signature using the inverse function $F^{-1}$ (Figure 2.17):

$$Signature = F(hash, privatekey, signature\ scheme)$$

$$Verification = F^{-1}(signature, publickey, signature\ scheme)$$

Today, as function F, we employ, for example, EC algorithms such as ECDSA (*Elliptic Curve Digital Signature Algorithm*) on Weierstrass curves P-256 and P-384 (the number also denotes the length of the keys). Additionally, TLSv1.3 also supports EC with the EdDSA algorithm on Edwards curves ed25519 and ed448.

***Note:*** *ECDSA algorithm keys can also be used for ECDH key exchange. However, in practice, separate key pairs are always generated for ECDSA and for ECDH. The only exception permitted is for signing the certificate request.*

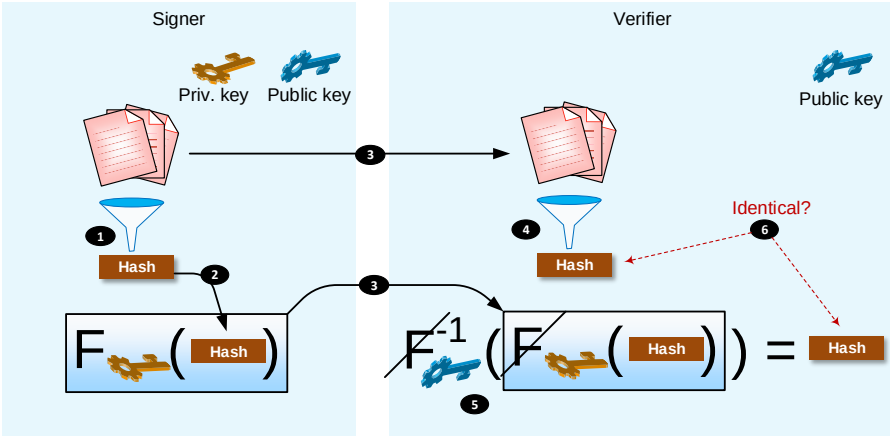



Figure 2.17 *Creation and verification of electronic signature in general*

### 2.9.3 Why is MAC not considered an electronic signature?

HMAC, CMAC or GMAC (hereafter MAC) are not algorithms for creating an electronic signature; they do not provide unique proof that was created by Alice (signer).

The explanation is simple: Because MAC uses the same key for computation and verification, the same key has to be known by both Alice and Bob. if Bob were an imposter, he could change the message from Alice himself and recalculate MAC from it. He would then deceive Alice. If Alice questioned it, he could show her the altered message and say, ”See, this is what you sent me. Don’t you remember?” Alice would be unable to claim justice because she could not prove whether HMAC was genuinely calculated by her or forged by Bob.

Thus, we do not refer to MAC as an electronic signature but more often as an **authentication tag**.

### 2.9.4 Composite electronic signature

The signature described so far is used rather exceptionally (for certificate or CRL signing). However, when signing documents, a composite signature is typically employed (Figure 2.18). A composite signature is created in two steps:

1. In the first step, the so-called Signed Attributes are generated. One of these signed

attributes is the Message Digest, which contains the hash of the signed document.

2. In the second step, a hash is calculated from the Signed Attributes, to which the function $F()$ is then applied.

The most common signed attributes are:

- Content type (data, timestamp, etc.).
- Message digest (hash).
- Signing time.

An example of a composite electronic signature is CMS Signed Data described in the chapter 6 CMS.

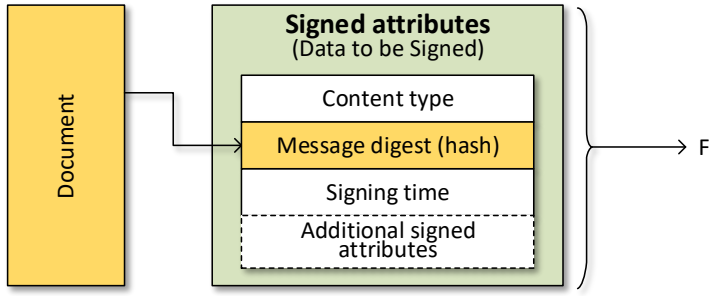



Figure 2.18 *Composite electronic signature*

***Note:*** *It is essential that we safeguard our private keys carefully. Losing a private key is similar to forging a portrait on an ID card or mixing up fingerprints in the offenders' register. Negligence in protecting your private key can be likened to signing blank checks.*

# 3 Some other algorithms

## 3.1 Base64

Base64 encoding [23] is intended for encoding general binary data that may not be human-readable. The encoded data is only one-third longer than the original data.

The coding algorithm is simple. It uses a base64 table (Table 3.1) of 64 characters (and the special "=" character). Six bits are needed to encode these 64 possible characters ($2^6 = 64$).

From an encoding point of view, a message is not viewed as a stream of eight bits blocks (bytes) but rather as a stream of six bits blocks. Each six bits block is then encoded according to the base64 table (Table 3.1).

At the beginning of encoding, the encoded text is divided into sequences of 24 bits (triples of bytes). Each triplet (24 bits) is divided into 4 groups of six bits each. Each six bits represent one character in the Base64 table. Every 6 bits are replaced by the corresponding character from the Base64 table.

The "=" character is used for a special purpose to indicate padding at the end of text.

Table 3.1 *Base64 Table*

| Value | Encoded | Value | Encoded | Value | Encoded | Value | Encoded |
|---|---|---|---|---|---|---|---|
| 0 | A | 17 | R | 34 | I | 51 | z |
| 1 | B | 18 | S | 35 | J | 52 | 0 |
| 2 | C | 19 | T | 36 | K | 53 | 1 |
| 3 | D | 20 | U | 37 | L | 54 | 2 |
| 4 | E | 21 | V | 38 | M | 55 | 3 |
| 5 | F | 22 | W | 39 | N | 56 | 4 |
| 6 | G | 23 | X | 40 | O | 57 | 5 |
| 7 | H | 24 | Y | 41 | P | 58 | 6 |
| 8 | I | 25 | Z | 42 | Q | 59 | 7 |
| 9 | J | 26 | a | 43 | R | 60 | 8 |
| 10 | K | 27 | b | 44 | S | 61 | 9 |
| 11 | L | 28 | c | 45 | T | 62 | + |
| 12 | M | 29 | d | 46 | U | 63 | / |
| 13 | N | 30 | e | 47 | V | | |
| 14 | O | 31 | f | 48 | W | | |
| 15 | P | 32 | g | 49 | X | | |
| 16 | Q | 33 | h | 50 | Y | padding 00 | = |

If invalid characters appear in Base64-encoded text, the encoded text should be rejected.

Notably, some implementations, such as MIME, are more permissive.

If less than 24 bits remain at the end of the text after division, the zero bits from the right are padded. Adding to the end is indicated by the ”=” character.

If the text entering the encoding is divided into groups of three bytes, then the last group:

- Can also have three bytes. Then there is no complication, and no ”=” characters are added to the end.
- Can have two bytes (16 bits). Then the first twelve bits are encoded normally according to the Base64 table. The remaining 4 bits are supplemented with two binary zeros to form 6 bits, and the result is also Base64 encoded. However, a single ”=” character is added to the end to signal to indicate padding of two bits.
- Can have one byte (8 bits). Then the first 6 bits are encoded normally according to the Base64 table. The remaining two bits are supplemented with four binary zeros, and the result is coded according to the Base64 table. Two ”=” characters are added at the end to signal to indicate a 4-bit padding.

The principle is best understood with examples:

**Example:** the length of the input text is divisible by three

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Input 8-bit: | 01101101 | 01001000 | 01111011 | 11100011 | 10101010 | 11110001 | |
| Input 6-bit: | 011011 | 010100 | 100001 | 111011 | 111000 | 111010 | 101011 | 110001 |
| Decimal: | 27 | 20 | 33 | 59 | 56 | 58 | 43 | 49 |
| Base64 (output): | b | U | h | 7 | 4 | 6 | r | x |

**Example:** the last group has two bytes

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Input 8-bit: | 01101101 | 01001000 | 01111011 | 11100011 | 10101010 | | |
| Padding: | | | | | | | 00 |
| Input 6-bit: | 011011 | 010100 | 100001 | 111011 | 111000 | 111010 | 101000 |
| Decimal: | 27 | 20 | 33 | 59 | 56 | 58 | 40 | |
| Base64 (output): | b | U | h | 7 | 4 | 6 | o | = |

**Example:** the last group is one byte

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Input 8-bit: | 01101101 | 01001000 | 01111011 | 11100011 | | | |
| Padding: | | | | | | 0000 | |
| Input 6-bit: | 011011 | 010100 | 100001 | 111011 | 111000 | 110000 | |
| Decimal: | 27 | 20 | 33 | 59 | 56 | 48 | |
| Base64 (output): | b | U | h | 7 | 4 | w | == |

### 3.1.1 Base64/MIME

MIME[24] is more liberal and permits line breaks in Base64-encoded text to enhance human readability: “The encoded output stream must be represented in lines of no more than 76 characters each. All line breaks or other characters not found in Base64 Table must be ignored by decoding software.”

**Note:** The protocols described in this book use Base64/MIME or Base64URL encoding.

### 3.1.2 Base64URL

In case we need to insert Base64 encoded data into the URI, then the characters '+', '/', and '=' are a problem because the URI uses them for other purposes. The encoding that addresses this is called Base64URI. Data are Base64 encoded with the following:

- "+" is replaced by "-" (minus).
- "\" is replaced by "_" (underscore).
- "=" is omitted.
- Line separators are disabled.

## 3.2 Shamir's secret sharing

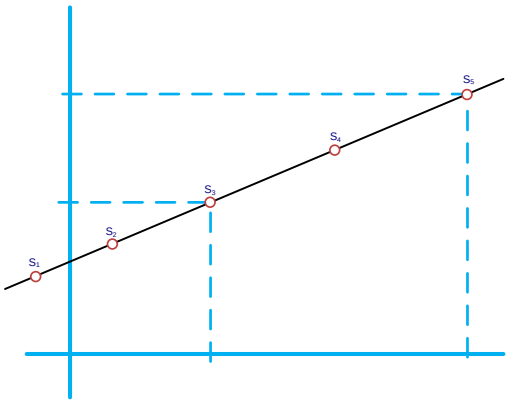



Figure 3.1 *Shamir's secret sharing ($k = 2$)*

Shamir's algorithm is designed to protect an asset (encryption key, shared secret) in such a way that in order to obtain this asset, it is necessary to compile a secret, the individual parts of which are distributed among $n$ users. Not all $n$ users need to meet to compile the entire secret, but only any $k$ of them are sufficient (thus: $0 < k \leq n$).

In practice, this means that we store the clues, for example, on $n$ chip cards, which we distribute to $n$ different holders. If we need to reconstruct the secret, then at least the holders must meet with their cards.

The principle of Shamir's algorithm is straightforward (Figure 3.1. Let us imagine that we want to distribute some secret among 5 administrators so that every time at least two meet, they piece together secret $B$. Then we choose an arbitrary (but fixed) number $A$ and construct the equation of the straight line:

$$y = Ax + B$$

We will then reveal the coordinates of a point on this line to each of the five administrators. Whenever two get together, they can determine $B$, i.e., put together a secret. For $k = 3$, we use a parabola, etc. We often encounter the use of this algorithm by administrators of Hardware Security Modules (HSM) when we want to ensure that more eyes are always present during the administration of these modules.

## 3.3 Photuris

[1]Source Wikipedia: Photuris lucicrescens.jpg; Author: Bruce Marlin; License: Creative Commons Attribution-

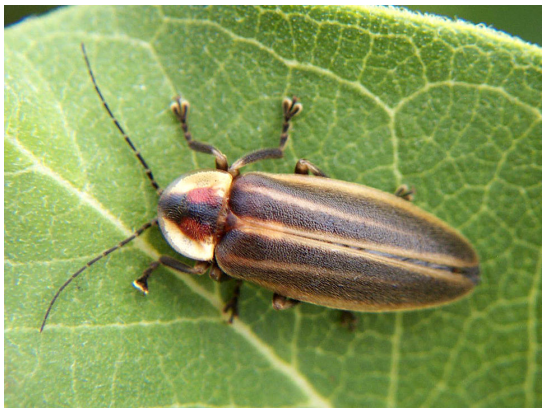


Figure 3.2 *Photuris*[1]

Photuris (Figure 3.2) is the Latin name for a genus of North American fireflies whose adult females imitate the light signals of other species of fireflies to attract their males and then eat them.

Photuris: Session-Key Management Protocol RFC 2522 [25] was created for now-forgotten devices. What remained of it was an idea for a way to defend against DDoS attacks using cookies. This mechanism is used by today's versions of the TLS, DTLS, QUIC, and SCTP protocols. It is not absolute protection against attackers but a principle by which we make life difficult for attackers and prevent attacks from fake IP addresses.

A classic attack on the TCP protocol is the SYN flood attack. In this attack, the attacker attacks the server with TCP packets with the SYN flag set. The server immediately allocates data structures to establish communication. If there are a large number of packets with the SYN flag set, the server may collapse by exhausting the allocated resources.

In the case of datagram services, the possibility of attack is significantly greater than in the case of the TCP protocol. So it would be good to slow down the attacker somehow before the appropriate resources are allocated.

The Photuris protocol came up with a solution where the server generates a cookie that it sends back to the client, who has to repeat his request again, but with a cookie. Only then is the request, for example, to establish a session, accepted.

This is pretty good protection against DoS attacks carried out from non-existent or fake IP addresses. It is possible to respond correctly from existing IP addresses, but it requires the attacker to understand the protocol in question.

The system can be improved so that the cookie is not generated by a simple random number generator, but the server first generates a secret. A structure is created that contains the secret, IP address of the client, and other parameters of the connection with the client. A MAC (Message Authentication Code) is calculated from this structure. The resulting MAC is included in the cookie.

The specific implementation is protocol dependent.

---

Share Alike 2.5 Generic.

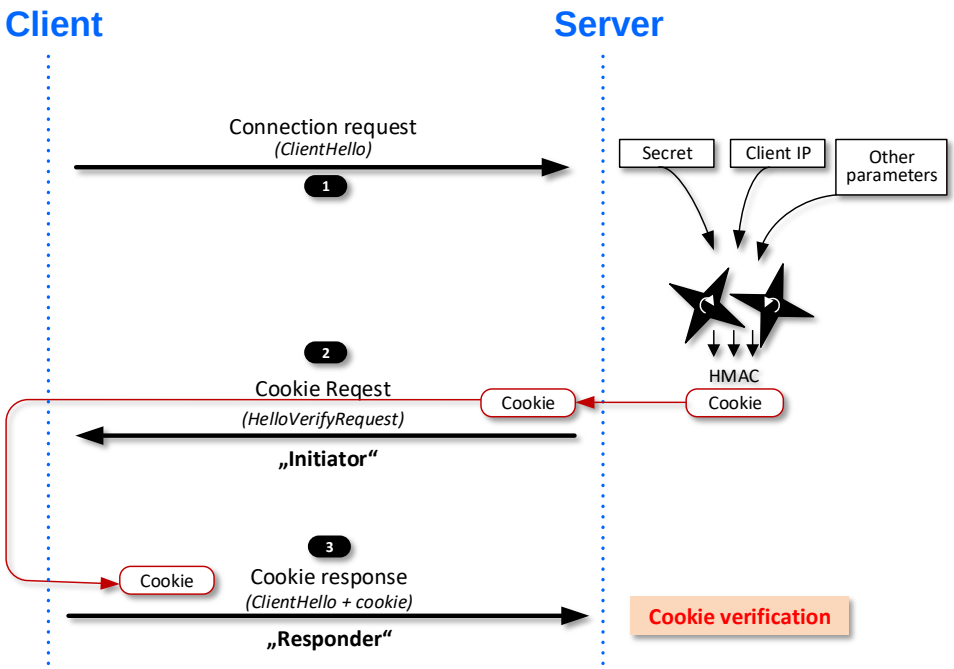



Figure 3.3 *Photuris: Session-Key Management Protocol implemented in DTLS protocol*

## 3.4 AKA

AKA (*Authentication and Key Agreement*) is a security protocol used in the 3G/4G/5G mobile networks for mutual authentication and cryptographic material (Figure 3.4) agreement.

The Extensible Authentication Protocol Method for the 3rd Generation called Authentication and Key Agreement (EAP-AKA) was defined by RFC 4187 [26]. The standard RFC 5448 [27] improves the EAP-AKA to EAP-AKA'. The EAP-AKA' is a small revision of the EAP-AKA. The change is a new key derivation function that binds the keys derived within the method to the name of the access network. In addition, the EAP-AKA' employs the SHA-256 hash algorithm instead of SHA-1.

5G networks use EAP-AKA' and new 5G AKA which is another revision of AKA [28].

In this publication, we present the AKA mechanism because it is very often used to authenticate subscribers in SIP-based networks.

There are three communication parties:

- A (mobile equipment), usually equipped with the USIM/ISIM smartcard containing a shared secret $K$.
- B (network), e.g., A-SBC (P-CSCF) in IMS.
- Authentication Centre (AuC), which is part of Home Subscriber Service (4G) or Unified Data Management (5G).

Parties A (mobile) and AuC:

- Share a secret $K$ (shared secret) different for each USIM/ISIM smartcard.

- Maintain sequence number SEQ of authentication.
- Both parties support the AKA mechanism using one way functions $f1, f2, f3, f4,$ and $f5$ (Figure 3.5), which are defined in standard.
- There is the well-known AMF string.

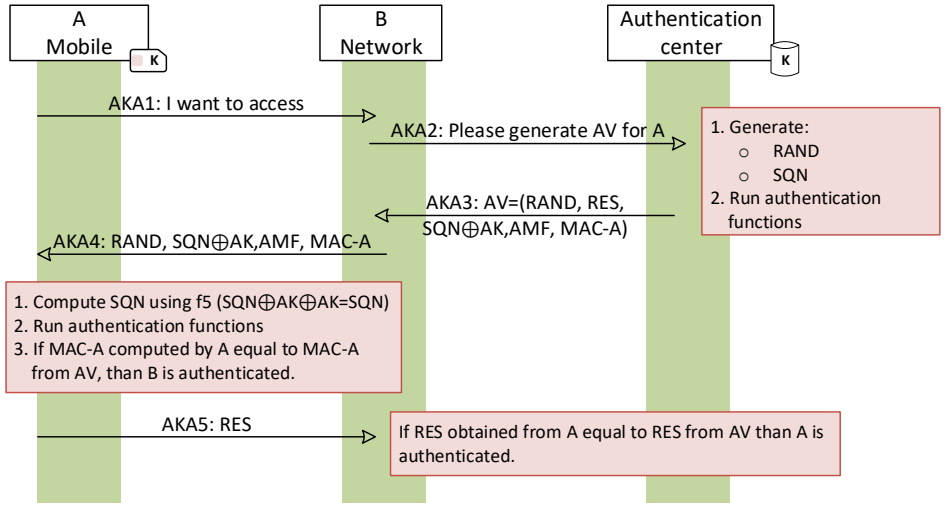



Figure 3.4 *AKA mechanism*

AKA mechanism (Figure 3.4) is the following:

**AKA-1:** A (Mobile) will grant access, then sends its identity to B (Network). Concrete: A wants mutual authentication – its own authentication and also the authentication of B (network).

**AKA-2:** B (network) sends identification of A (mobile) to the Authentication center (AuC).

**AKA-3:** The Authentication center on behalf of B (network):

- Generates a random number `RAND`.
- Generates the next SEQ sequence number of authentication.
- Runs Authentication functions $f1, f2, f3, f4,$ and $f5$ (Figure 3.5) and generates the Authentication vector `AV = (RAND, RES, CK, IK, SQN` $\oplus$ `AK, AMF, MAC-A)`. While `RES` is a one-time password for authentication of A (mobile) and `MAC-A` is a one-time password for authentication of B (network).
- Sends `AV` to B (network).

**AKA-4:** B (Network) store `RES` from Authentication vector `AV`. Next, `RAND, SQN` $\oplus$ `AK, AMF` and `MAC` sends to A (Mobile).

**AKA-5:** A (Mobile) knows `SEQ` and `AMF`, and next from B (Network) receives `SQN` $\oplus$ `AK, AMF` and `MAC-A`. A at first by the function $f5$ verifies `SQN`. Next, A runs the rest of authentication functions and:

- If the MAC−A, computed by A (Mobile), is equal to the MAC−A from AV, then B (Network) is authenticated.
- Generates RES and sends it to B (Network).

**AKA-6:** B (Network) compares the RES, obtained from A (Mobile), with saved RES from AV (step **AKA-4**). If equal, A (Mobile) is authenticated.

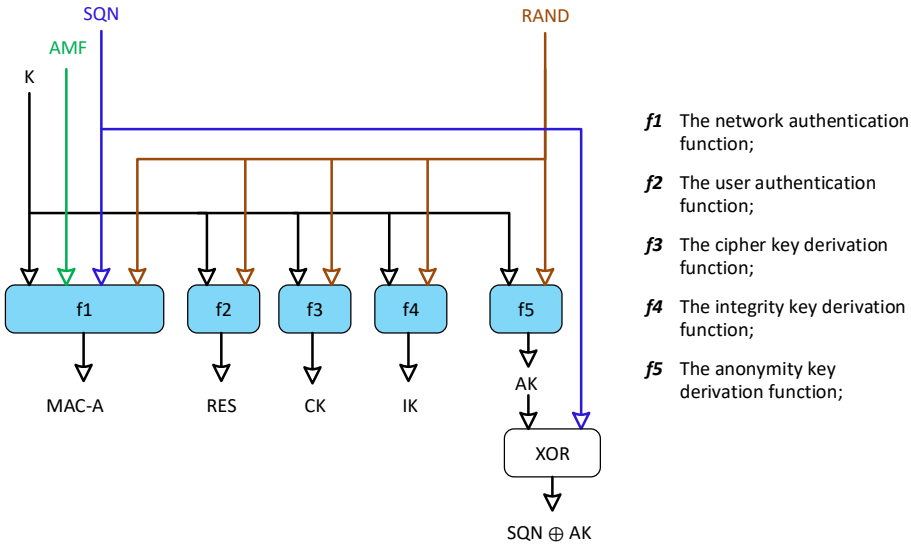



Figure 3.5 *AKA Authentication function* $f1 - f5$

Table 3.2 *Notation of AKA*

| Symbol | Meaning |
|---|---|
| K | Shared secret (permanent key stored in USIM/ISIM and at the same time in AuC) |
| AMF | Authentication Management Field. Well-known string defined by standard |
| SQN | Sequence number of authentication |
| RAND | Random number |
| MAC−A | One-time password of network authentication |
| RES | Response that is the output of the function $f2$ (one-time password of user equipment authentication) |
| CK, IK | Ciphering key, Integrity key (both keys derived in AuC and on USIM/ISIM during AKA authentication) |
| $K_{ASAME}$ | A key derived by mobile equipment and AuC during an AKA authentication. From this key, symmetric keys are subsequently generated for encryption of communication. |
| AK | Anonymization key (anonymized SEQ) |
| $f1$–$f5$ | One-way functions |

AKA should be used for mutual “silent” authentication. The word “silent” means, that the user single sign on (after switching on the mobile equipment, a user inserts his/her PIN for opening access to the shared secret K) and the application runs subsequent authentication without any user intervention, e.g., when entering the visited network.

Cryptographic key $\mathrm{K}_{\mathrm{ASAME}}$ is a result of AKA authentication. It is known by both A (Mobile) and B (Network). Cryptographic material is then derived from $\mathrm{K}_{\mathrm{ASAME}}$ to ensure communication between A and B.

It is also necessary to define the functions $f1, f2, f3, f4$ and $f5$. A concrete example of functions $f1, f2, f3, f4,$ and $f5$ is defined by the "MILENAGE algorithm set" [29] [30] protocol.

# 4 Certificates and certification authorities

Let us revisit our acquaintances: Alice, Bob, and Chuck (Figure 4.1). Bob generated a pair of asymmetric keys (1); he sent the generated public key Pub-B to Alice via Chuck (2). Bob hopes that Alice will encrypt her answer with Bob's public key, thereby preventing Chuck from reading the message. Even if Alice sent this encrypted message with Chuck's help, Chuck would not be able to read it.

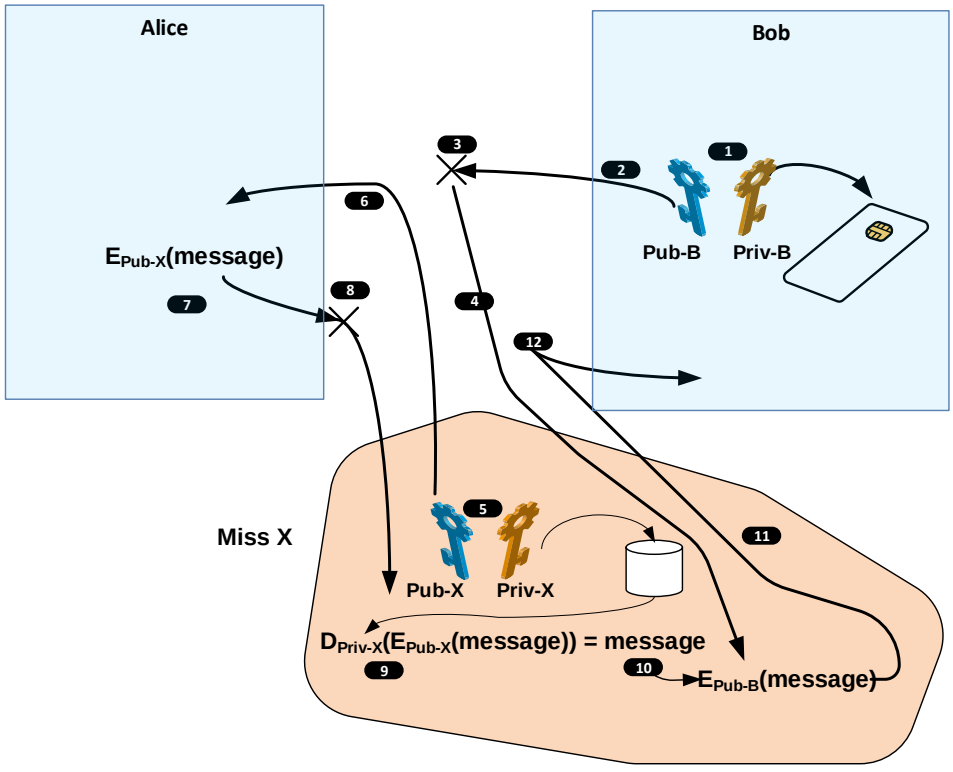



Figure 4.1 *An attack on Bob's public key distribution (RSA algorithm)*

Chuck slowly makes his way to Alice, and then a plan forms in his head (3). He stops by his girlfriend – a hacker calling herself Miss X. Maybe she can't give him some advice, but it will not hurt to try anyway. Miss X examines Bob's public key and immediately tells Chuck that breaking the RSA algorithm is beyond her power. But there is a very simple way to help Chuck! She takes Bob's public key from Chuck and keeps it for later use (4). Now she generates her own key pair: a public key Pub-X and a private key Priv-X (5). She gives the

newly generated public key Pub-X to Chuck and tells him to bring it to Alice, claiming that it is Bob's public key. Chuck complies immediately (6).

Now Alice, believing that she is encrypting her reply with Bob's public key, encrypts the reply with Pub-X's public key and sends the encrypted message with Chuck's help to Bob (7). Chuck immediately hurries after Miss X (8). She is already impatiently waiting. She snatches Alice's reply encrypted with her Pub-X public key, decrypts it (9) with her private key Priv-X, and gets a message in clear text. She shows this to a surprised Chuck, who even allows her to change the message in his favor. She then encrypts the result with Bob's public key Pub-B (10) and sends it to Bob (11) through Chuck. The unsuspecting Bob decrypts the message with his private key Priv-B (12).

Alice encrypted the message in good faith. Bob received a message from Alice, which he deciphered as he was supposed to. Chuck not only read the content of the message; he could even change it to his advantage. And Miss X also enjoyed the outcome. So, everybody is happy! This type of attack is called a Key distribution attack.

## 4.1 What is the defense?

So, what is a defense against public key distribution attacks? Before Alice uses any public key, she should obtain some proof that the public key really belongs to the intended owner, i.e., that it is not forged. Alice has the following options for verifying the authenticity of Bob's public key:

- Alice receives the public key personally, directly from Bob at their mutual meeting. In this case, it is clear beyond any doubt that the public key belongs to Bob.
- Alice verifies that the public key indeed belongs to Bob. For example, before using the key for the first time, she calls Bob, authenticates herself to ensure she is really talking to him and not some attacker, and asks him to recite the public key hash or part of it to her. Authentication may be based on shared experiences, such as discussing recent events they both attended. For instance, Alice might ask, "How did you like the movie we went to together yesterday?" and Bob replies, "What are you talking about? Yesterday we went to the park."
- Bob has the authenticity of his public key confirmed by an independent third party (e.g., a notary).

### 4.1.1 Does Bob have the corresponding private key?

It would be most appropriate for Bob to also provide Alice proof that he owns the corresponding private key. Such a **proof of ownership** of the private key can be, for example, an electronic signature of the public key, or of some data structure that contains the public key, created using the corresponding private key. However, this type of proof is only possible when an electronic signature can be created using the corresponding private key.

### 4.1.2 Did Bob generated his key pair using secure device?

We have proof that public key belong to Alice and another proof that Alice have corresponding private key. We also can have proof that the key pair was generated and is secured by device with sufficient level of security.

Private key is asset, that can be very valuable. We are trying to sufficiently secure this asset. Often we keep private key in specialized devices, e.g. smart cards, USB tokens or HSM. This devices generate key pair inside them and they never show/output private key. It would be unfortunate if we mistakenly generated key pair outside this device and saved private key on unsecured drive, because we would open ourselves to possibility of key compromise.

As a proof, that private key is secured in specialized device, we usually use one-time passwords, that are generated together with key pair. How is it possible, that one one-time password is enough? We usually only need to proof this once - during certificate creation.

### 4.1.3 Conclusion

Before Alice uses Bob's public key, she should have:

1. Public key.
2. Proof, that this public key really belongs to Bob.
3. Proof, that Bob really have corresponding private key.
4. When securing data of significant value, proof that key pair was generated on specialized device, that securely protects private key.

If Bob let independent third party (e.g. notary) assess validity of his private key, then this party do this 4 steps and after that he issues certificate of validity of Bob's public key. Bob that gives Alice his public key and this notarized certificate.

## 4.2 Public key certification

However, it is more practical to defend against forgery of the public key by certifying the public key via an independent third party – a certification authority (hereafter CA). Bob generates a public/private key pair, carefully storing and guarding the private key as his secret asset. The public key is not sent to Alice by itself, but only as part of the certificate issued by the certification authority (Figure 4.2). But let us not get ahead of ourselves.

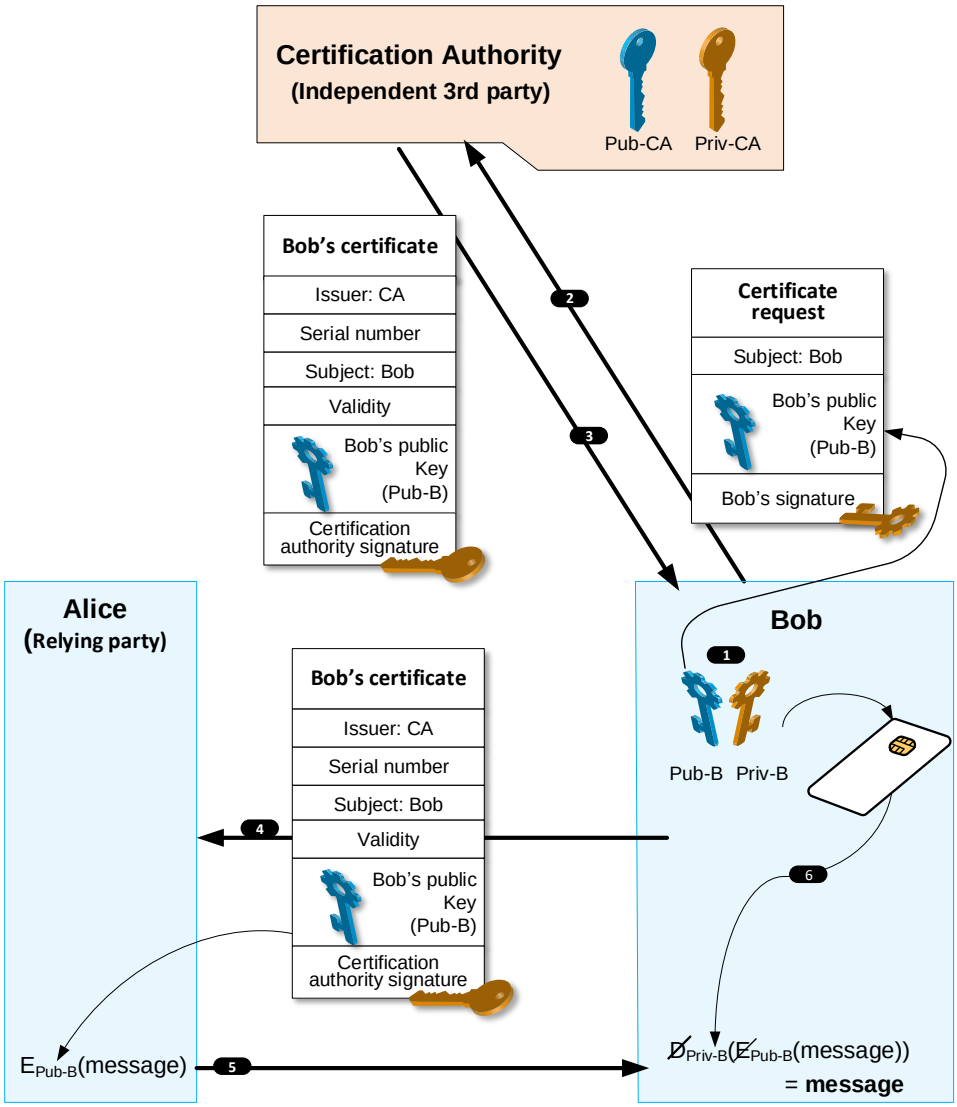



Figure 4.2 *Certificates and Certification Authority*

Initially, Bob generates a pair of keys (1) and creates a structured CSR *(certificate signing request)*. This structure contains Bob's identification data, his public key, and possibly other data, which we will discuss later. Bob then digitally signs this CSR with his just-generated private key (= proof of ownership of the private key) and passes the result to the certification authority (2). The certification authority can verify Bob's identity. In any case, however, it verifies the electronic signature on the certificate request so that the CA can confirm that Bob really has the relevant private key. If the request is found to be acceptable by the CA, then the CA issues the certificate.

The certificate is a data structure containing Bob's public key and his identification data. Furthermore, the certificate contains the name of the certificate issuer (Issuer Name of the certification authority), the serial number of the issued certificate, the validity of the certificate, etc. The binding of Bob's identification data and his public key is confirmed by the

certification authority with its electronic signature.

The issued certificate (3) is returned by the certification authority. Now Bob can send his public key to Alice as part of the just-issued certificate (4). Alice verifies this certificate and, if everything is OK, extracts the public key from this certificate, which she then uses to encrypt the message to Bob (5). Bob then uses his private key to decrypt the message (6) and thus obtains the original message.

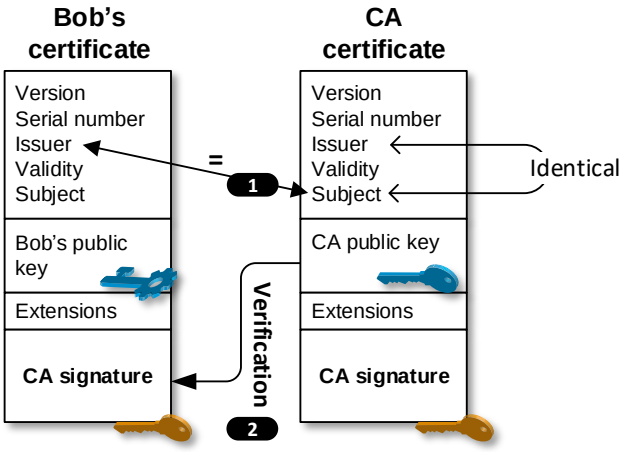



Figure 4.3 *Simplified Bob's certificate verification*

What does Alice verify on Bob's certificate? We will discuss later that certificate verification is quite a complicated procedure. At this point, let us just remind you that Alice needs to verify that Bob's certificate was issued for him by a trusted certification authority (whose identification is in the certificate issuer item). Since the certificate is a digitally signed structure, Alice must verify the electronic signature on Bob's certificate.

Figure 4.3 shows the link between Bob's certificate and the certificate of the CA which issued Bob's certificate. Alice must find the appropriate CA certificate in her Trusted CA certificate repository or by some other mechanism. She proceeds by first searching her repository of trusted certificates and looking for the certificate of the certification authority that issued Bob's certificate. She recognizes it by the fact that the Issuer in Bob's certificate (1) is equal to the Subject in the CA certificate. She then extracts the public key from the CA certificate and uses it to verify the electronic signature in Bob's certificate (2). Certificate authority certificates are similar to user certificates. The certification authority can have its certificate issued by:

- Another certification authority, in which case the certificate of this CA has different entries for Issuer and Subject. We call such a CA certificate a cross-certificate.
- Itself, i.e., the CA issues a certificate to itself, signing it with its private key. In this scenario, the certificate of this CA has the same Issuer and Subject. We refer to such a certificate as a **self-signed** or **self-issued certificate**.

## 4.3 Proof of ownership of the private key

Now, we will explain why the previously mentioned proof of ownership of the private key is important. The goal of the certificate holder is to distribute the certificate as widely as possible. However, it would be problematic if the certification authority issued a certificate for a public key to one user, and then another user soon appears with a request for certification

of the same key. This situation arises because when these users possess a key pair with the same public key, they also possess the same private keys, effectively revealing the private key.

In any case, the certification authority should monitor whether it has already certified the same public key and reject another request for certification of the same key. This illustrates how practical it is to supplement the certificate request with proof of ownership of the private key. Without this proof, anyone who obtains a certificate could apply for the certification of an already certified public key.

To prevent the generation of identical key pair (meaning public/private keys), high-quality random number generators are used to generating keys. True random number generators are employed for this purpose. This minimizes the likelihood that two different users would generate the same pair of data. However, a software error can cause the random number generator to fail in generating numbers completely randomly. There have also been documented cases where errors in the random number generator were induced artificially using so-called side-channel attacks.

Table 4.1 *Comparison of certificate and ID card entries*

| **Certificate entry** | **ID card entry** |
|---|---|
| Version of certificate structure | Version of the ID card format (booklet, card, etc.) |
| Serial number | ID card Serial number |
| Signature Algorithm | Types of protective elements |
| Issuer | Issued |
| Validity | Validity |
| Subject: name, address, … | Name and address |
| Public key | – |
| – | Photo |
| Certificate extension | Optional data |
| Electronic signature | Protective elements |

## 4.4 Certificate

The certificate is often compared to an identity card. While the ID card is issued in printed form, the certificate is a digitally signed data structure, binding the holder's public key to their identity (see 4.1).

We can also compare the individual items on an identity card with those on a certificate (we will learn later that this comparison forms the basis of the Registration authority operators' work when verifying the identity of the certificate applicant).

There are several standards that define the structure of the certificate (such as X.509, EDI, EMV, etc.). The Internet primarily relies on the ITU-T standard X.509, specifically version 3 of the certificate structure. To cater to the requirements of the Internet, an Internet profile of the X.509 standard has been developed and documented in relevant RFCs. The current standard for Internet certificate profiles is RFC 5280 [31].

Next we will describe some certificate entries.

### 4.4.1 Version

The Version determines whether the certificate is derived from version 1, 2, or 3 of the X.509 standard structure. The Version entry is zero for version 1, one for version 2, and two for version 3. Essentially, only version 3 certificates are used today.

### 4.4.2 Serial number

The Serial number is defined as a positive integer up to 20 bytes long (not 20 digits!), which must be unique within a specific certification authority. In other words, the certification authority shall not issue two certificates with the same serial number. The pair of Serial Number + Issuer items uniquely identifies the certificate.

**Note:** It is recommended that at least part of the Serial number is generated randomly.

### 4.4.3 Signature Algorithm

The Signature Algorithm entry specifies the signature scheme used to create the electronic signature of the certificate, optionally including its parameters.

### 4.4.4 Validity

The Validity item determines the validity period of the certificate from 'Not Before' to 'Not After'. A common question is why the validity period of the certificate is limited. There are two reasons:

- **Organizational:** The application has a certain lifetime. Undoubtedly, it is also commercially interesting to issue certificates more often.
- **Security:** The lifetime of the certificate should be significantly shorter than the expected time required to crack the certified public key. This is especially crucial for certificates of certification authorities, which should be issued for a period at least five times longer than the lifetime of user certificates. With a shorter period, the overhead of renewing user certificates increases significantly.

It must be emphasized again and again that the certificate is not useless after the expiration date and therefore cannot be carelessly discarded. After the validity period has expired, we simply cannot sign new messages using the private key corresponding to the public key specified in the certificate. However, to verify the electronic signature of messages created during the validity period of the certificate, we will always need expired certificates in the future, as long as verification is necessary.

Similarly, if we archive messages encrypted with the public key of the certificate, we will again need the private key corresponding to the certificate that was valid at the time the message was created to decrypt them. When processing a message, it is therefore practical to decrypt the message and possibly re-encrypt it before storing it in the archive, but this time with the encryption key of the archive.

**Note:** Although it may seem unbelievable, some IoT applications (e.g., smart meters) require certificates with practically unlimited validity (up to 9999 years) to be issued for some sensors. The reason for this is the fact that the sensor can be turned off for a very long time, and if it were to be turned on after the expiration of its certificate, it would no longer authenticate to the IoT network, thus losing connectivity.

### 4.4.5 Distinguished Name

Both the Issuer and Subject items use the same data format known as the Distinguished Name.

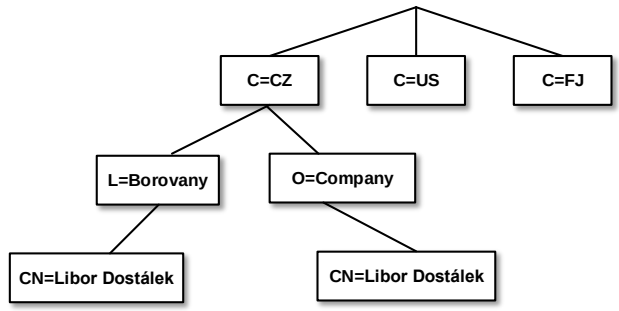



Figure 4.4 *Hierarchical structure of RDN*

The Distinguished Name was originally introduced in the X.500 series of ITU standards, specifically in the X.501 standard. The goal of the X.500 series of standards is to create a worldwide directory structure. The term ”directory” here does not refer to a directory of files, but rather a directory akin to a list of addresses in a phone book. The objective is to establish a global equivalent of a telephone directory. Each entry in such a directory corresponds to a Distinguished Name.

A Distinguished Name in such a list is composed of partial information about the subject: the country name, the organization name, the organizational unit, the common name, and so on. Each specific partial information that contributes to forming a unique name is called a Relative Distinguished Name (RDN).

A Distinguished Name is thus formed by a sequence of RDNs. Additionally, the same RDN can be repeated within a single Distinguished Name, e.g., with a different value.

The RDN itself comprises a set of attributes. An attribute consists of an object identifier (e.g., `Country`, `Organization`, `Common Name`, etc.) and a value (e.g., `CZ`). We write a relative unique name in the format (attribute=value). E.g.:

```
Common Name=Libor Dostálek
```

A Distinguished Name describing an individual is then a sequence of RDNs, for example:

```
Common Name=Libor Dostálek, Organization=Company, Country=CZ
```

This notation is often abbreviated using abbreviations for RDN object identifiers:

```
CN=Libor Dostálek, O=Company, C=CZ
```

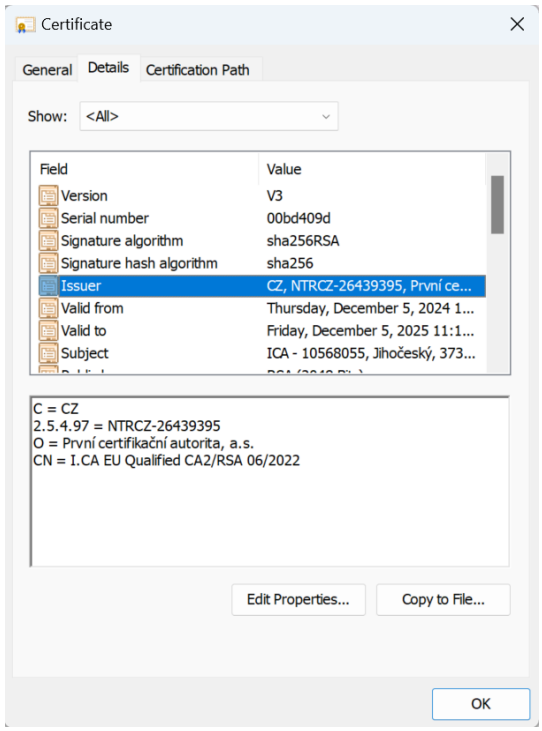



Figure 4.5 *Displayed certificate in Windows*

Although an RDN is a set of attributes, in practice, this set usually consists of only one attribute (identifier and value). Distinguished Names are always formed by branches in the tree of RDNs.

Interestingly, the holder Libor Dostálek can be listed in many ways in the structure (each time it forms a different unique name!).

For example:

As a resident of the Czech Republic, Borovany locality:

```
CN=Libor Dostálek, L=Borovany, C=CZ
```

As an employee of the Company:

```
CN=Libor Dostálek, O=Company, C=CZ
```

We use a Distinguished Name to specify a person, a system, or an entity in general. An interesting situation arises with the names of natural persons. Different nations have their own customs for naming their citizens; therefore, the specific use of individual attributes depends on the certification policy of the specific certification authority.

The user provides his/her Distinguished Name in the certificate request, and the CA copies this Distinguished Name into the certificate. In general, the CA should not modify the Distinguished Name when copying it from the certificate request to the certificate. The only exceptions are the `DNQualifier` and `SerialNumber` attributes, which the certification authority will most likely add to the certificate. These two attributes serve to distinguish

Table 4.2 *An overview of the RDN attributes used by PKI*

| Attribute | Abbreviation | Importance |
|---|---|---|
| Common Name | CN | The name of the object by which it is known locally. For example, for persons, it can be first and last name. For servers, their DNS name, etc. |
| Surname | SN | Surname |
| Country | C | Country according to ISO 3166 (CZ = Czech Republic, SK = Slovakia, FJ = Fiji...) |
| Locations | L | Location (e.g. city) |
| State or Province | S | Higher territorial administrative unit |
| Organization | O | Organization name |
| Organizational Unit | OU | Organizational unit (e.g. department) |
| Title | T | Position (e.g. governor, company executive, director, etc.) |
| Name | | Name |
| Given Name | G | Given Name |
| Initials | | Initials |
| Generation Qualifier | | For example, "Jr." or "III" for Charles the III. |
| DNQualifier | | See above |
| Serial Number | | See above. Not to be confused with the certificate Serial number! |
| Pseudonym | P | Pseudonym |
| Email Address | E | **Do not use!** |
| Domain Component | DC | Individual strings from a domain name (e.g. Windows domains). E.g. domain Controller info.pvt.net is DC=info, DC=pvt, DC=net. *Attention (!) this attribute is not directly related to DNS name – it is a different namespace.* |

two different persons who would otherwise have the same unique name.

The usage of `DNQualifier` and `SerialNumber` is not standardized. `DNQualifier` is often not used by CAs, while the `SerialNumber` attribute is used in Europe for ID card (or passport) numbers. Therefore, the specific meaning of the DNQualifier and SerialNumber attributes should be determined according to the relevant certification policy of the specific certification authority.

Another common practice of CAs is to move the Email Address attribute from the subject of the certificate request to the appropriate certificate extension, as today's standards directly call for CAs not to include the Email Address attribute in the subject of certificates.

### 4.4.6 Issuer

Issuer entry contains the Distinguished Name of the certification authority that issued the certificate. It is necessary for the certification authority to have a unique identification (unique name) among all certification authorities.

An attacker may attempt to create a CA with the same name but using their spoofed pub-

lic/private key pair. Particularly, if our certificate authority uses a root certificate, we must be especially careful about its certificate distribution.

### 4.4.7 Subject

If we use certificates according to X.509 version 3, then the subject of the certificate must be unique within all objects certified by the given certification authority. In other words, the certification authority may not issue a certificate with the same subject to two different objects. On the other hand, it is very practical for a certificate authority to issue certificates to the same person repeatedly with certificates having the same subject.

As a rule, we utilize a wider range of Distinguished Name attributes in the subject of the certificate than in the Distinguished Name of the CA, where we should be more moderate, even though the software should support a variety of attributes.

Many identification data that do not ”fit” in the subject of the certificate can be stored in the Subject Alternative Name extension of the certificate.

An interesting aspect is that the subject of the certificate can also be empty (an empty sequence of RDN). In such a case, the certificate must contain the Subject Alternative Name extension, which must be marked as critical.

### 4.4.8 Public key

The Public Key entry is a sequence of two pieces of information: the identifier of the algorithm for which the public key is intended, and the public key itself (including parameters of the public key).

Unlike the Signature Algorithm entry, the mentioned algorithm specifies the algorithm for which the certified public key is intended.

### 4.4.9 Certificate extensions

Even though the certificate extension is defined quite generally, it also faces a problem similar to the certificate subject attributes, wherein some applications may not understand certain extensions and their purpose. This issue is addressed by using extension parameters to mark extensions as critical or non-critical.

Table 4.3 *An overview of the certificate extensions*

| Certificate extension | CA certificates | End user certificates |
|---|---|---|
| **Standard internet extensions** | | |
| Authority Key Identifier | Mandatory in all non-root certificates. Must not be marked as critical. | Same as CA certificates. |
| Subject Key Identifier | Mandatory; must not be critical. | Should be; must not be critical. |
| Key Usage | Mandatory in certificates that verify the electronic signature of certificates and CRLs; should be critical. | Same as CA certificates. |
| Extended Key Usage | It is mandatory for some certificates (TSA, OCSP, etc.). | Same as CA certificates. |
| Private Key Usage Period | It should not be used. | Same as CA certificates. |
| Certificate Policies | Optional. | Optional. |
| Policy Mappings | If used, then it must not be critical. | n/a |
| Subject Directory Attributes | If used, then it must not be critical. | Same as CA certificates. |
| Subject Alternative Name | Optional and may not be critical. | Required and critical if subject is empty. |
| Issuer Alternative Name | Should not be critical. | Same as CA certificates. |
| Basic Constraint | Must be used and marked as critical. | n/a |
| Name Constraints | If used, it may or may be critical. | n/a |
| Policy Constraints | If used, it may or may not be critical. | n/a |
| CRL Distribution Points | Should not be critical. | Same as CA certificates. |
| Inhibit Any-Policy | If used, it may or may not be critical. | n/a |
| Freshest CRL | Must not be critical. | Must not be critical. |
| **Private Internet Extensions** | | |
| Authority Information Access | Must not be critical. | Must not be critical. |
| Subject Information Access | Must not be critical. | Must not be critical. |

Software working with the certificate must understand all extensions marked as critical, as they contain crucial information. If any of the extensions in the certificate is marked as critical and the software will not recognize its purpose, it must reject the entire certificate.

RFC 5280 [31] specifies the standard extensions listed in the following table. Interestingly, not every one of these standard extensions needs to be supported by specific applications.

Table 4.4 *Examples of extensions created for specific use cases*

| Certificate extension | CA certificates | End user certificates |
|---|---|---|
| **Qualified certificates** | | |
| Qualified Certificate Statements | It may or may not be critical. | It may or may not be critical. |
| **Microsoft** | | |
| Certificate Template Name | Can be used. | Can be used. |

## 4.5 Guide to some certificate extensions

In the rest of this chapter, we will consider some certificate extensions. The main goal of this chapter is to provide the reader with a basis to understand these extensions when viewing the content of the certificate. An example of a certificate is shown in Figure 4.6, where critical extensions are marked with an exclamation mark in a yellow triangle, while other extensions are marked with a green arrow in a white bullet. The thumbprint contains the hash of the certificate.

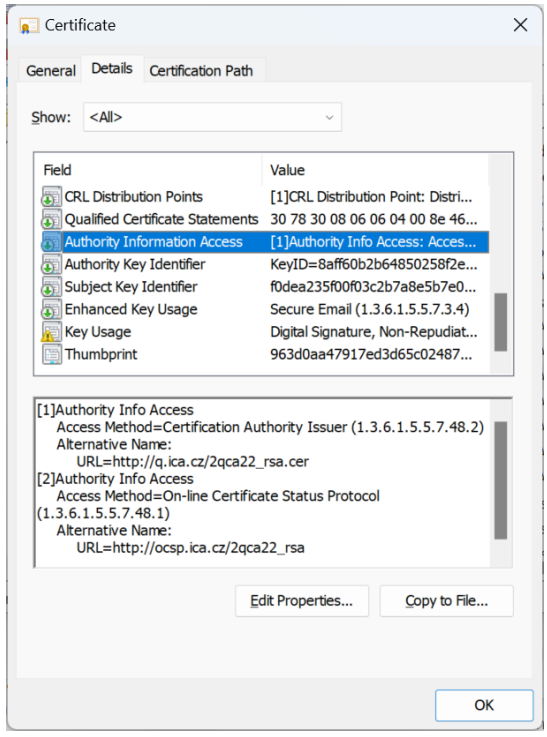



Figure 4.6 *Displayed certificate extensions in Windows*

### 4.5.1 Subject Key Identifier and Authority Key Identifier

As previously mentioned, it is possible for the same person to have multiple certificates issued (Figure 4.7). In this context, the certification authority itself is also considered a person. So, what would it be, if we had a scenario where a CA had two certificates with the same subject but different public keys?

Imagine that Alice wants to verify Bob's certificate. She would encounter two CA certificates. Without clarity on which certificate from the certification authority to choose for verification of Bob's certificate, Alice faces a dilemma. One might think to try one certificate first, and if verification fails, try the other. But what if both CA certificates fail? What does that signify? Is Bob's certificate forged, or does the certification authority possess a third certificate suitable for Bob's verification?

This is where the Subject Key Identifier extension, coupled with the Authority Key Identifier extension, comes into play. The certification authority assigns unique identifiers to its individual public keys, for example a hash (or parts of a hash) of the public key. Then, the certification authority includes the Subject Key Identifier extension in its certificate, containing this identifier.

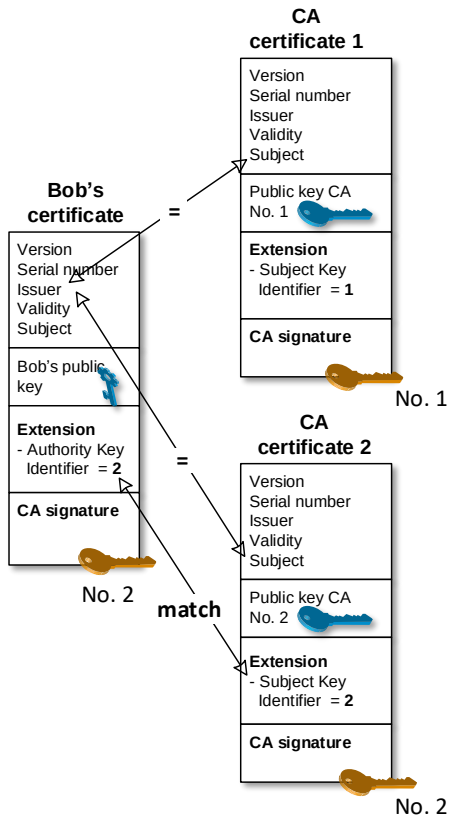



Figure 4.7 *Alice finds the correct CA certificate by matching the contents of the Authority Key Identifier and Subject Key Identifier extensions*

When the certification authority issues certificates to users, it includes the Authority Key Identifier extension in each issued certificate. This extension contains the same identifier as the Subject Key Identifier extension in the CA certificate. This linkage allows for seamless verification of certificates, resolving the ambiguity that could arise from multiple CA certificates.

Let us return to Alice. When Alice searches her repository of trusted CAs to find the public key to verify Bob's certificate, she finds a CA certificate that has a Subject matching the Issuer in Bob's certificate. Now, she additionally checks whether the contents of the Authority Key Identifier extension in Bob's certificate match the content of the Subject Key Identifier extension in the CA certificate.

In other words, the pair of the Subject Key Identifier extension and the Authority Key Identifier extension is used to identify the key of the certification authority with which the certificate was signed. This is particularly important if the CA has multiple public/private key pairs. If you believe it is unnecessary for a CA to have multiple key pairs, you may not have realized that even a CA certificate has a validity period. Certificates of certification authorities also need to be renewed. For a certain period of time, the certification authority has two certificates: old and new.

The Subject Key Identifier extension can also be useful for end-user certificates. The content of this extension can effectively identify the certificate. In the next chapters, we will see that the certificate is identified either by the pair Issuer + Serial number or by the content of the Subject Key Identifier extension, which has the great advantage of having a fixed length.

### 4.5.2 Private key usage period

This extension is usually not present in certificates, but knowledge of the Private Key Usage Period mechanism is absolutely essential, especially for administrators. If they fail to issue a new CA certificate in time, it can result in job failure, potentially costing their employer a significant amount of money.

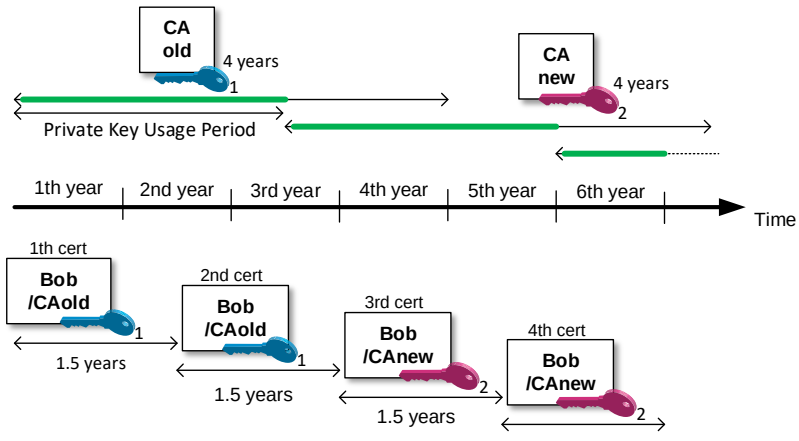



Figure 4.8 *Private key usage period*

The Private Key Usage Period is shorter period than the validity of the entire certificate CA. This allows for digitally signing with a key marked in this way for, for example, 2.5 years, while still being able to verify this signature without additional timestamps for, say, 4 years.

Now let's consider slightly different ques-

tion: why should a CA have two valid certificates for a certain period of time during rekeying (or renewing) a certificate?

Let us imagine how Bob's favorite certification authority works (see Figure 4.8), whose certificates are valid for 4 years. It issues certificates to its users (such as Bob) for a maximum of 1.5 years. The upper part of the figure shows the validity of the certification authority's certificates, while the lower part shows the validity of the user certificates.

Let us focus on Figure 4.8. The certification authority will issue the first certificate to Bob. When this certificate is about to expire, Bob's certification authority will issue a second certificate. Both certificates are verified by the CAold certificate. However, when Bob's second certificate is about to expire and he wants to renew it again, the certification authority cannot issue a certificate that is verified by the CAold certificate. Why? If it did so, then after one year of validity of Bob's certificate, Bob's certificate would be formally valid, but its verification would fail because the CAold certificate would already be invalid.

So, the conclusion is that the certificate authority must renew its certificate no later than the maximum duration for which it issues certificates to its users before the expiration of its old certificate. This is because the CA cannot issue a certificate signed by the CA key to the user, which would expire during the validity period of the issued certificate. In other words, a situation would arise where the user has a valid certificate but is signed with an expired certificate.

Thus, the CA has two overlapping certificates for a relatively long time. Some users have their certificates signed with the "old" CA certificate, and others with the "new" CA certificate. Both CA certificates will have the same subject. They will differ in certificate serial number and public. In the user's certificate, the subject of the CA certificate with which the user's certificate is signed is listed in the Issuer item. This is the same for both the new and old certificates of the certification authority. Once again, this problem is solved using the Authority Key Identifier and Subject Key Identifier extensions mentioned in the previous paragraph.

In the upper part of the image in Figure 4.8, the validity period of the private key is marked green. During this time, the key of the certification authority is used to sign issued certificates. After this period, the certification authority will start using the new certificate. It is a good idea to start the ceremony to dispose of the private key belonging to the old certificate of the certification authority after this period.

It is also possible to enter the Private Key Usage Period extension in the certificate, but it is not recommended. Interestingly, this extension may limit the validity of the private key even at the beginning of the validity of the certificate.

### 4.5.3 Key Usage

This extension can be used to restrict the ways of using the private key matching to public key contained in the certificate, i.e., to restrict the use of the certificate. The extension contains a bit string. Each bit in the string corresponds to a specific way of using the certificate. If the relevant bit is set to TRUE, then the certificate can be used for the given application. (If the given bit does not appear in the string, it can be assumed that it is set to TRUE.) Meaning of

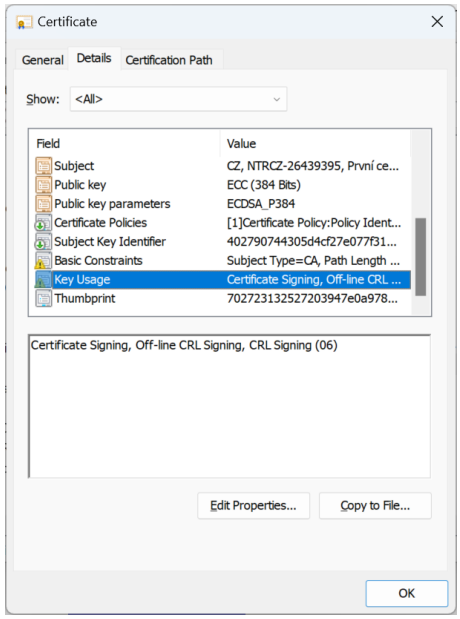
individual bits:

- **Digital Signature (bit 0)** – The certificate is intended for the electronic signature of data (as an algorithm). Setting this bit **does not** authorize:
  - Content commitment (authenticity) of document digital signature.
  - Verification of certificate and CRL signature.

  You may be wondering about the purpose of such a certificate. It can be used for:
  - User authentication service.
  - Data origin authentication.
  - Data integrity verification.
- **Non-Repudiation (bit 1)** (*recent editions of X.509 and EU standards have renamed this bit to **Content Commitment***) – The certificate is intended to verify non-repudiation or, if you want, authenticity. The names of the Digital Signature bit and the Non-Repudiation bit are completely misleading. The Digital Signature bit signals any use that is based on the electronic signature algorithm, not a specific use for verifying authenticity in the case of an electronic signature. The Non-Repudiation bit signals the specific use of the electronic signature algorithm as a verification of authenticity. So, if, for example, we want to use a certificate to verify the electronic signature of a document, which is supposed to replace a handwritten signature, then this bit must be set, and the Digital Signature bit does not necessarily have to be set.
- **Key Encipherment (bit 2)** – The certificate is designed to encrypt keys. A classic case is hybrid encryption based on the RSA algorithm, where the data is encrypted with a random symmetric encryption key, which is enclosed with the message and itself encrypted with the public key from the certificate marked in this way.
- **Data Encipherment (bit 3)**– The public key of the certificate marked in this way is intended for data encryption (other than encryption keys).
- **Key Agreement (bit 4)** – The certificate is intended for key exchange (e.g., (EC)DH key exchange).
- **Key Certificate Sign (bit 5)** – The public key specified in this certificate is intended for certificate verification, i.e., the private key belonging to this public key can be used for signing certificates.
- **CRL Sign (bit 6)** – The public key specified in this certificate is intended for CRL verification.
- **Encipher Only (bit 7)** – This bit is used in conjunction with the Key Agreement (key exchange) bit. The resulting agreed symmetric key can only be used for encryption.
- **Decipher Only (bit 8)** – This bit is used in conjunction with the Key Agreement bit (key exchange). The resulting agreed symmetric key can only be used for decryption.

The extension is marked as critical. This prevents the certificate from being used for purposes other than those indicated in the certificate.

**Example:**




On the left, we can see a list of certificate items, including a list of certificate extensions in the Microsoft environment. Extensions marked with a green arrow inside a circle are non-critical, while those marked with an exclamation mark inside a yellow triangle are critical.

The content of the **Key Usage** extension is displayed in the lower section: "Certificate Signing, Off-line CRL Signing, CRL Signing (06)." Let us set the text aside and focus on the hexadecimal value 06. When converted to binary, it becomes 00000110, meaning bits 5 (Certificate Signing) and 6 (CRL Signing) are set.

This indicates that the public key associated with this certificate can be used to verify the signatures of certificates and Certificate Revocation Lists (CRLs).

### 4.5.4 Extended Key Usage

It is a more general solution for determining the purposes for which the certificate is intended. This extension may contain a sequence of object identifiers specifying specific ways of using the public key.

This extension is prescribed for use by some other authorities. For example, timestamping authorities (TSA) or OCSP responders require that their servers' certificates explicitly state that they can be used for this purpose.

### 4.5.5 Subject Alternative Name

This extension allows additional names of the certificate holder to be inserted into the certificate, for example, an email address; DNS names; IP addresses; another Distinguished Names of the same form as used for the subject of the certificate, etc. It is important that multiple alternative names can be specified.

**Notice:** When issuing the certificate, the checking of the data specified in this extension must not be omitted. We should drop the bad habit of putting email addresses and DNS names in the subject of the certificate, and only include them here in the Subject Alternative Name extension.

In this extension, one or more occurrences of the following may be stated:

- **Other Name** – other identification data.
- **RFC822 Name** – email address according to RFC 822 [32] (e.g. dostalek@example.org).

- **DNS Name** – DNS name (e.g. server name *.example.org).
- **X.400 Address** – e-mail address according to the standards of the X.400 series.
- **Directory Name** – directory name according to X.500 series standards, i.e., it has the same format as the subject or certificate issuer.
- **EDI Party Name** – name according to EDI standards.
- **Uniform Resource Identifier** – URI (e.g. http://www.example.org).
- **IP Address** – Ipv6 or IPv4 addresses which contain exactly 4 bytes of the IP-address version 4 or 16 bytes for IPv6. The individual bytes are not separated by separators or converted to the decimal system.
- **Registered ID** – object identifier.

### 4.5.6 Certification policies

This extension contains the identifier of the Certification Policy document. Additionally, it may also contain a hyperlink to this document, which is not part of the certificate. That is, only this extension and not the entire document is included in the calculation of the electronic signature of the certificate. Therefore, it is not guaranteed that the certification authority will not change this document after the certificate is issued.

A certification policy is a document specifying the procedures, practices, and goals used during the certificate's lifetime, that is, the rules under which the CA issues certificates and, in particular, how they guarantee the issued certificates. A certification policy is a document issued by a certification authority. Unlike some other CA documents, the certification policy is a public document usually displayed on the Internet (on the website of the operator of the certification authority).

Extension Certification Policy in the certificate:

- May not be specified. This is, for example, the case of certification authorities that are part of the Microsoft server. These certification authorities do not assume the creation of a laborious certification policy during the installation of the certification authority. Instead of the certification policy, they use a private extension to the certificates: Certificate template.
- May contain the identifier of the certification policy and a possible hypertext link to this policy, which is published on the Internet. An elaborate certification policy is several orders larger than the size of the certificate itself, so placing it in the certificate would be inefficient.
- May contain max. 200 characters Notice of issuer's statement that extends or supersedes the certification policy (e.g.: "For testing purposes only", "This is a qualified certificate for electronic signature according to Regulation (EU) No 910/2014", etc.).

**Note**: Before creating Certification policy should be studied first [33].

### 4.5.7 Constrains

With its electronic signature, the CA guarantees the data specified in the issued certificates. Figure 4.9 illustrates the danger when the CA issues certificates to its users: Alice, Bob, and Chuck. Chuck arbitrarily declares himself a certification authority and issues certificates to other users: Miss X and Hacker Y. The problem is that the CA is indirectly responsible for the certificates issued in this way to users X and Y. For example, Miss X receives a certificate chain containing: Miss X's certificate, Chuck's certificate, and the CA's certificate. Miss X's certificate is verified using Chuck's certificate, and Chuck's certificate is subsequently verified using a CA certificate. So, there is no problem verifying the validity of the certificate issued in this way by Chuck, but the problem lies in the CA's responsibility for the certificates issued in this manner.

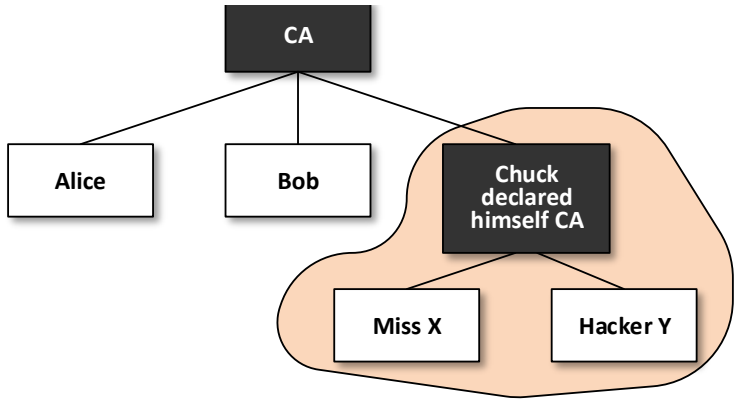



Figure 4.9 *Chuck arbitrarily declared himself a CA*

We have already described one mechanism to prevent such user behavior using the Key Usage extension. With this extension, in certificates issued to users, we limit the use of their key so that it cannot be used to verify certificates. However, this extension is not mandatory.

#### 4.5.7.1 Basic Constraints

The Basic Constraints extension may be used in the CA certificate or an end-entity certificate. In the case of a CA certificate, it allows you to specify how many child CAs this CA can have. For example, if this item is set to zero, then this CA can only issue certificates to end users and not to subordinate certificate authorities. In an end-entity certificate, this extension indicates that it is an end-entity certificate.

This extension is marked as critical for CA certificates.

#### 4.5.7.2 Name Constraints

The certificate of the certification authority is limited by this constraint to issuing certificates to users meeting the conditions specified in their certificates for Distinguished Names or Subject Alternative Names. For example, you can restrict issuance to DNS names belonging

to the *.example.org domain. Mailbox names can also be restricted to email addresses belonging to a certain domain. It is also possible to restrict IP addresses, etc. This extension can only be used in CA certificates.

### 4.5.8 CRL Distribution Points (CDP)

The certificate revocation list (CRL) can be issued either by the certification authority itself, or it can authorize another authority to issue the CRL (e.g., the authority for issuing the CRL).

This extension contains a list of distribution points where a Certificate Revocation List (CRL) is exposed. A distribution point is usually represented by a URI.

**Note:** If the CRL Distribution Points extension contains multiple values, then each name describes a **different mechanism** to obtain the same CRL. For example, the same CRL could be available for retrieval through both LDAP and HTTP.

### 4.5.9 Authority Information Access (AIA)

This extension can serve several functions. In practice, the following two options are usually used most often:

- If we want to verify the validity of the certificate, then we need to have the certificate of the certificate authority that issued it. We may, for example, already have it stored on our computer. However, if we don't have it, we find ourselves in a difficult situation. Where can we get it? It is in the AIA extension that there can be a link (URL) where the necessary certification authority certificate can be found. However, it is not possible to establish a trusted anchor in this way.
- The second option is to include a link to the OCSP server, with which we can check online if the certificate has been revoked.

### 4.5.10 Certificate template

Certificate authorities supplied by Microsoft issue certificates for various purposes, such as CA certificates, CA subordinate certificates, certificates for user login to Windows, certificates for user signatures only, etc. In general, it would probably be possible to write a certificate policy for each purpose and mark it here in the certificate. However, Microsoft went the other way – they created a list of many possible purposes. And they have a specific Certificate Template for each purpose from the list.

### 4.5.11 Qualified Certificate Statements

This extension indicates that it is a qualified certificate according to the EU regulation eIDAS. The extension contains a sequence of certificate qualification flags.

A qualified certificate is a special type of certificate in terms of law (not by format) used by EU legislation. Among other things, qualified certificates can be used to create a qualified electronic signature, which replaces a handwritten signature within the EU.

## 4.6 Certificate chain and trusted anchors

So far, we have assumed that we have a root CA that issues certificates to its end users (Figure 4.10). The Root CA possesses a self-signed certificate, which exhibits several special features:

- It is signed by the private key corresponding to the public key published in this CA certificate.
- The Issuer field is equal to the Subject field.
- Any verification of the electronic signature of the root certificate will therefore only serve to verify the integrity of the data structure of the certificate.

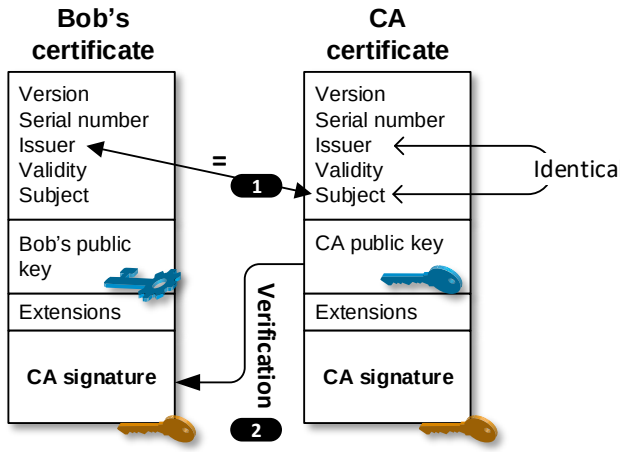



Figure 4.10 *Verifying the end-user certificate issued by the root CA*

Therefore, it is relatively easy to forge a root certificate. An attacker only needs to generate an arbitrary public/private key pair and create a root certificate with an arbitrary content for the public key generated in this way. Consequently, root certificates must be distributed in a trusted manner. One such method is the distribution of certificates within the operating system.

CA certificates (or public keys) whose validity we verify in another way and therefore explicitly trust are referred to as **trusted anchors**. An example of a trusted anchor is a root certificate that we verified using an alternate method. However, we can declare any CA certificate as a trusted anchor.

### 4.6.1 Tree of certification authorities

However, reality can be more complicated. Our CA can only be the root of the CA tree.

In addition to end-user certificates, the CA certification authority can also issue certificates to subordinate certification authorities, such as CA A and CA B. Similarly, for example, the certification authority CA B can create its subordinate certification authorities, CA C and CA D, by issuing certificates to them. The result is a tree structure of certification authorities (Figure 4.11).

This tree structure is commonly referred to as **Public key Infrastructure (PKI)**.

What does a subordinate CA certificate look like? The most important change compared to the root CA certificate is that this certificate has different values in the Issuer and Subject fields.

It can certainly have a root CA certificate as a trusted anchor, but it is often more practical to designate another CA's certificate as a trusted anchor. Certificate verification is then performed against this trusted anchor.

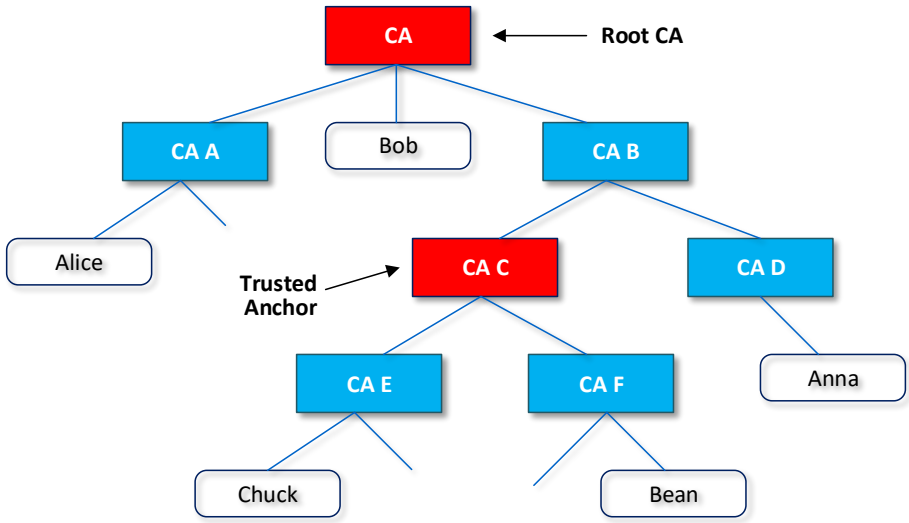



Figure 4.11 *CA tree*

At first glance, the certificate of the subordinate CA (cross-certificate) appears similar to the certificate of the end user. However, upon closer examination, we find that the cross-certificate differs considerably, especially in the extensions of the certificate. Why? The reason is simple. By issuing a cross-certificate to someone, we grant them the authority to issue other certificates. And if we somehow guarantee the issued certificates, we will also be responsible for the certificates issued by the individual to whom we issued the cross-certificate. We must be cautious when issuing cross-certificates.

### 4.6.2 Certificate chain

In the case of a CA tree, verifying an end user's certificate is more complex. It requires verifying not only the end user's certificate but also the certificate of the certification authority that issued it. If this certification authority also has a cross-certificate, then that too must be verified. This entails verifying a chain of certificates (Figure 4.12).

The certificate chain terminates with a trusted anchor, typically in the form of a root certificate. In theory, the public key itself (including any parameters and the name of its issuer) can also serve as a trusted anchor.

When given a set of certificates, the first task is to construct a certificate chain up to the trusted anchor. Usually, the process begins with the end user's certificate. We search for

the certificate of the certification authority that issued it by locating all certificates with a Subject field identical to the Issuer field of the end user's certificate. There may be multiple such certificates, from which we select the one whose Subject Key Identifier extension value matches the Authority Key Identifier extension value of the authenticated certificate. We then check if we have the certificate of this certification authority among the trusted anchors. If so, we halt the search for the next element of the certificate chain; otherwise, we continue similarly until we reach a trusted anchor.

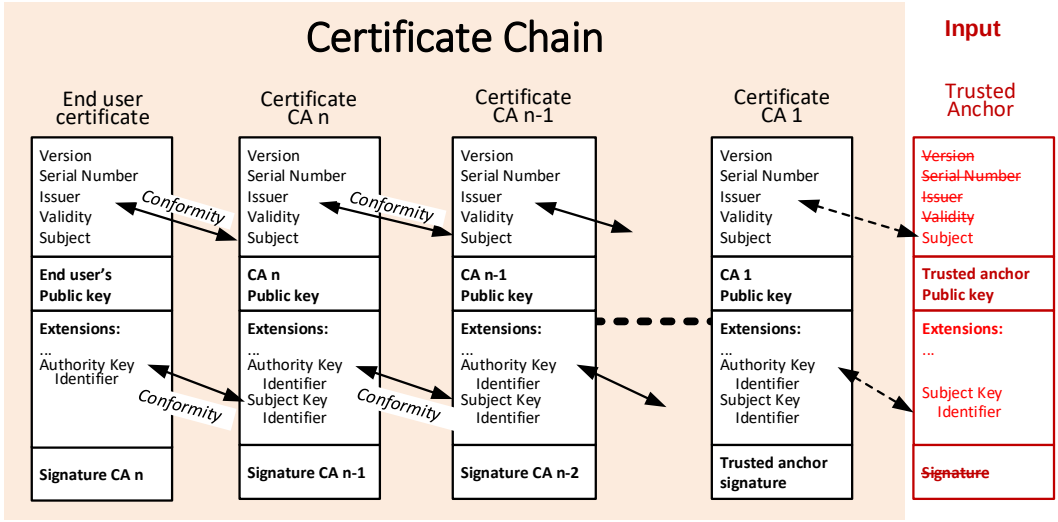



Figure 4.12 *Certification Chain. We proceed from "left to right" when compiling it, but when verifying it, we proceed "from right to left"!*

### 4.6.3 Verifying the validity of the certificate

In the previous paragraph, we compiled Certificate chain from the certificate we want to verify to the trusted anchor. Now our goal is to verify the already compiled Certificate chain, i.e., we will deal with the verification of the specific compiled Certificate chain.

We implicitly assume that we want to verify the certificate's validity at the present time.

**Note:** Another task is verifying if the certificate was valid at some point in the past. This task poses a significantly more complex problem, especially when authenticating archived electronically signed documents.

### 4.6.4 Chain validation starts from a trusted anchor!

When compiling the certificate chain, we proceed from the certificate to be verified. However, during verification, we proceed in the opposite direction—from the trusted anchor to the verified certificate. Since we trust the trusted anchor and therefore do not need to verify it, we gradually verify the validity of the first certificate under the trusted anchor, then the second, until we reach the certificate whose validity we need to verify.

### 4.6.5 We are verifying the Certificate chain

To validate the certificate, we verify that the public key specified in the certificate is valid and indeed belongs to the holder specified in the subject of the certificate. Additionally, we ensure that this key can be used for the intended purpose. This is why we have created a certificate chain up to the trusted anchor and will now proceed to validate it. We designate the certificate issued by the trusted anchor as the first certificate in the certificate chain. Subsequently, we refer to the certificate issued by the first certificate as the second certificate in the certificate chain, and so on. The trusted anchor is often denoted as certificate number 0.

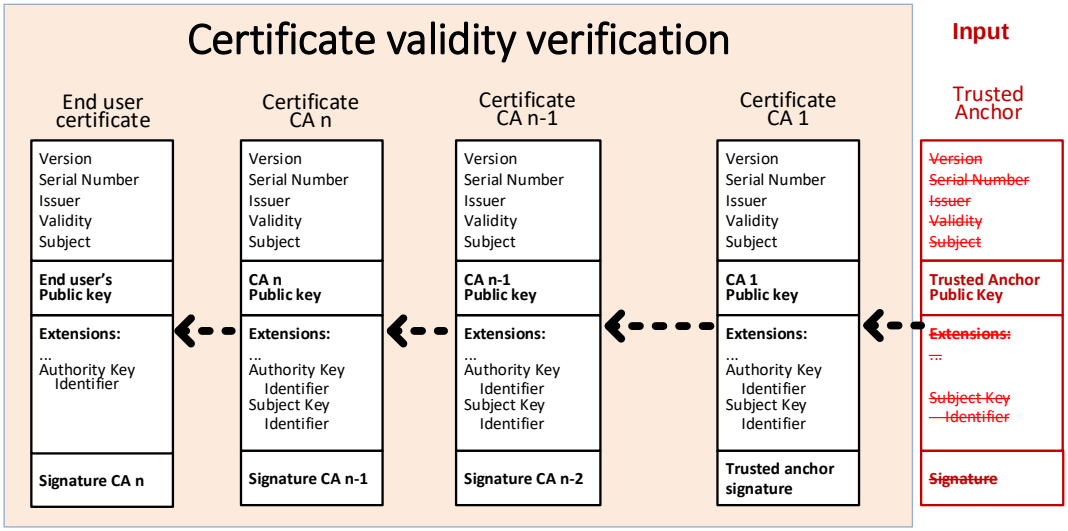



Figure 4.13 *Verification of the Certificate chain takes place from the trusted anchor to the certificate being verified*

The trusted anchor may or may not be a root certificate. From this certificate, the following is taken for verification:

- The public key including the public key algorithm and any public key parameters.

Since other information is ignored in the trusted anchor during certificate verification, trusted anchors, if they are in the form of a certificate at all, usually contain minimum certificate extensions. If the issuer of the trusted anchor has issued multiple certificates with the same subject, then the only extension having a real effect is the Subject Key Identifier extension, which has an effect for building the Certificate chain (not for verifying the certificate).

It enters the verification process:

- Compiled Certificate chain.
- A trusted anchor.
- Current time.
- Initialization information regarding certificate policies in case we intend to check certificate policies when validating a certificate.

The use of many certificate extensions is optional. So, if the application does not support these extensions, then it does not need to use them and can skip their verification. However, if an unsupported extension is marked as critical, then the certificate must be rejected.

When verifying a certificate, we will go step by step along the Certificate chain from certificate number 1 to the certificate we want to verify. Let us imagine that we are validating certificate number ***x*** in the certificate chain, then we verify (see RFC 5280 [31] for details):

- That the certificate issuer is the subject of certificate ***x-1***.
- The electronic signature of certificate ***x*** using certificate ***x-1***.
- That the current time falls within the validity period of the certificate.
- That the certificate has not been revoked (see the next paragraph).
- That the names listed in the subject of the certificate and in the Subject Alternative Names extension correspond to the restrictions set forth in the Name Restrictions extension in the previous certificates of the Certificate chain.
- Extensions associated with certification policies.

If the verification of any certificate of the Certificate chain up to the verified certificate did not fail, then the certificate is valid.

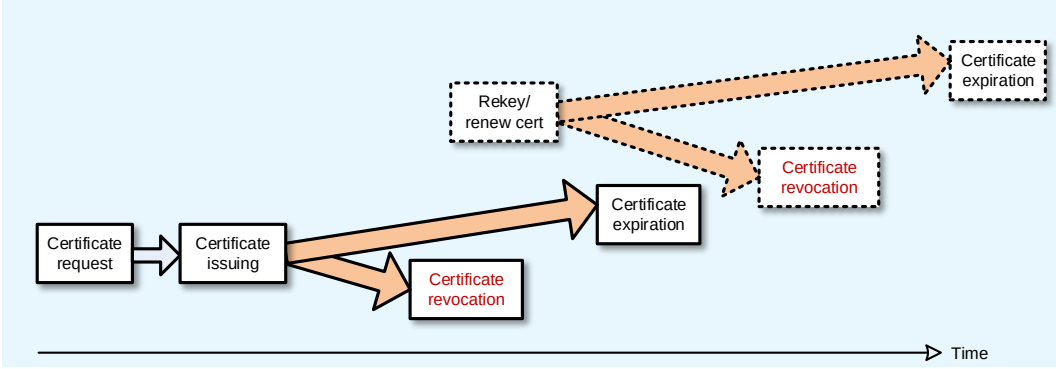



Figure 4.14 *Certificate Lifecycle*

## 4.7 Certificate Lifecycle

Over time, the certificate goes through several phases that make up the life cycle of the certificate (Figure 4.14). The certificate lifecycle consists of the following phases:

1. **Creation of a certificate request**. The creation of the request may or may not precede the generation of key pair.
2. **Issuance of the certificate and its possible publication**.
3. **Certificate validity**. After the certificate has been issued, it may not be automatically valid. The validity of the certificate begins at the time specified in the entry Not Before and ends either with the expiration (Not After) or with the revocation of the certificate.
4. **Expiration of the certificate** occurs in Not After time specified in the certificate.

5. **Revoking the certificate** before the expiration of its originally declared period of validity. A certificate is revoked by the certification authority by publishing the certificate's identification on the certificate revocation list (CRL). The revoked certificate is listed on all CRLs for the period of its original validity.

**Notice:** In the case of qualified certificates, the EU requires that they be listed on the CRL even after the original validity period has expired.

CA revokes the certificate:

- From the decision of the CA:
  - Another user requested the certification of an already certified public key.
  - The certification authority found that the information in the certificate is no longer true (e.g., the person has changed their name).
  - The certificate holder violated the terms of the certification policy.
- At the request of the certificate holder. Reasons for this can be:
  - The employee resigned (in case of certificates for employees).
  - The user no longer wishes the certificate to be valid for personal reasons.
  - The user's private key has been compromised.
  - The user's private key has been destroyed.
  - The equipment has been replaced by new equipment.

There also exists a suspension of certificate validity, known as certificate hold. Difference between suspension and revocation is that suspension is reversible, but revocation is not. The suspended certificate remains listed on all Certificate Revocation Lists (CRLs) until the original certificate expires. However, during the suspension period, it is listed in the CRL with the ”certificateHold” Reason Code, and during the post-revalidation period (“reincarnation”), it is listed in the CRL with the ”removeFromCRL” Reason Code.

### 4.7.1 Certificate renewal

If the certificate is about to expire, we can:

- Certificate **rekey** – we generate new key pair and create a new request for a certificate.
- Certificate **renew** – we do not generate new key pair; we use the old certificate request.

**Note:** The problem is that administrators often say ”renew” and mean ”rekey”.

**Note:** Key pair may only be generated by the certification authority after receiving the certificate request. Such request is not in the CSR format.

If the certificate is still valid, then we can use it for renewal. What does this mean? When applying for a certificate, we typically need to verify our identity so that the certification authority can confirm the information in the application. If we have a valid certificate, the certification authority has already verified our identity. Therefore, it suffices to use the valid certificate to confirm our identity again for renewal.

For instance, if we have to physically appear to prove our identity when obtaining the initial certificate, it is usually unnecessary for certificate renewal—we can electronically validate ourselves using the valid certificate.

The certification authority typically maintains the same subject for the renewed certificate as the original one. (However, the specific procedures for issuing and renewing certificates are outlined in the certification policy of the relevant certification authority.)

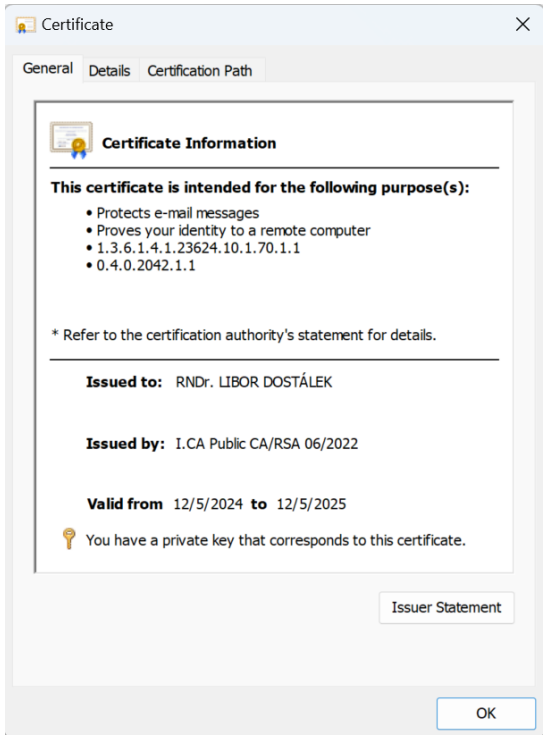



Figure 4.15 *Displaying certificate in Windows*

If we utilize multiple certificates concurrently (e.g., for encryption, signature, authentication, etc.), obtaining a signature certificate suffices, and we can acquire additional certificates with the understanding that we no longer need in-person identity verification but can electronically verify ourselves, such as through an electronic signature using a valid signature certificate.

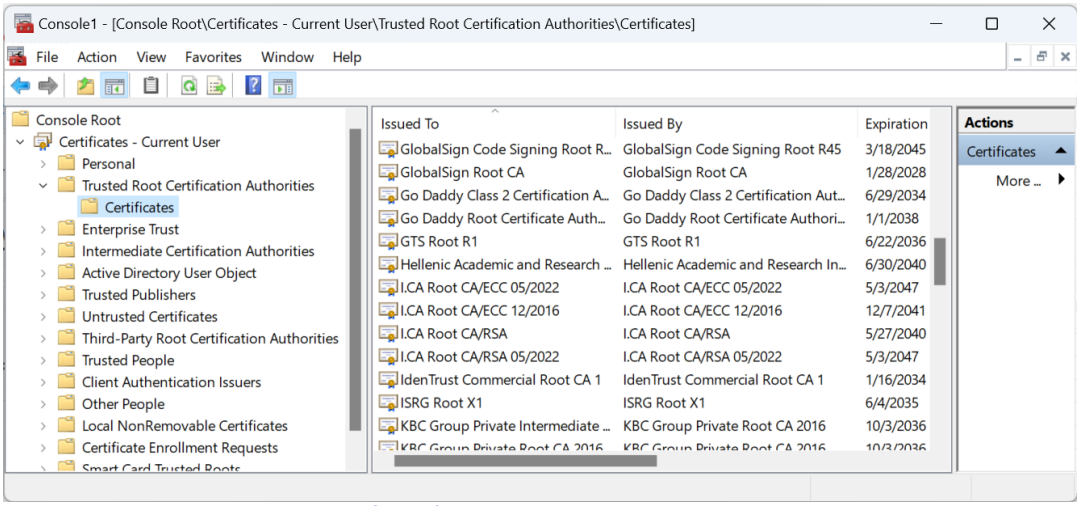



Figure 4.16 *List of certificates in Windows using the program mmc*

### 4.7.2 Was certificate revoked?

We can get information about certificate revocation from CRL or some other way (e.g. using OCSP protcol). Where (URI) can this information be found is written in following extensions:

- `(CRL Distribution Points)`: Shows URI, where CRL is published.
- `(Freshest CRL)`: Shows URI, where delta CRL is published.
- texttt(Auhtority Info Access) Can show URI of relevant OCSP server.

## 4.8 Certificate in Windows

In Windows, file extensions such as .cer, .crt, .der, etc., are typically associated with the Certificate manager. Clicking on a file with one of these extensions will display the contents of the certificate (Figure 4.15).

Another method to view certificates in Windows is by running the mmc program. By selecting ”Add or remove snap-in module,” we can add the ”Certificates” module, as shown in Figure 4.16. This image displays the certificate stores of the currently logged-in user. Additionally, administrators can view the local computer certificate store and the service store. These repositories are particularly important for servers operating on this computer.

Here is the significance of some certificate repositories from Figure 4.16:

- **Personal:** This contains the personal certificates of the currently logged-in user. Importantly, we usually have the corresponding private keys for these personal certificates. The presence of the private key is marked by a yellow target in the form of a key (see Figure 4.15 - before text ”You have a private key ...”).
- **Trusted Root Certification Authorities:** This contains trusted anchors from third-party Root Certification Authorities, Microsoft trusted anchors, and enterprise trusted anchors.

- **Intermediate CAs:** This holds certificates of Intermediate CAs, including their CRL.
- **Active Directory User object** This contains certificates that the current user has registered in Active Directory.
- **Other People:** This contains certificates of other users issued by trusted CAs. This repository can also be used for email to search for recipient certificates.
- **Certificate Enrollment Request:** This stores certificate requests.

Regarding personal certificates, we typically have private keys for them. Private keys in Microsoft operating systems are stored:

- On disk: They can be stored locally or as part of the User Profile (ActiveDirectory).
- On a smart card, USB token, or TPM chip on the computer motherboard (or its virtualization).

For server certificates, private keys are stored on the server disk, on the Trusted Platform Module (TPM) or in the Host (or sometime Hardware) Security Module (HSM).

## 4.9 Certification and registration authorities

A Certificate Authority (CA) is an independent third party responsible for issuing certificates. However, the term ”certification authority” can be interpreted in two ways: either as a software application (responsible for issuing certificates) or as an organization (overseeing the certificate issuance process). As an organization, a CA can be implemented either as a standalone company or as a department within a larger organization.

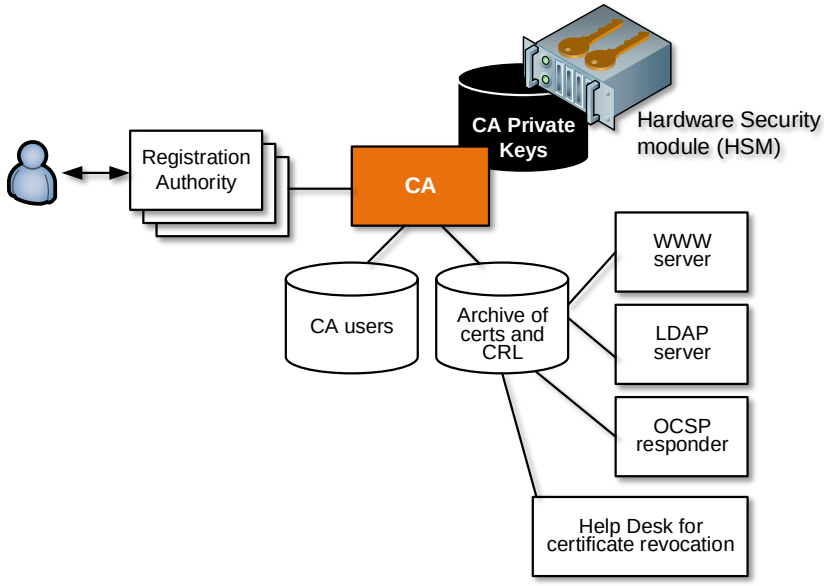



Figure 4.17 *Certification authority architecture*

As an institution, a CA typically consists of several fundamental components (Figure 4.17):

1. **Registration Authorities (RA):** These are often structured similarly to bank counters. Certificate applicants visit the RA to submit their applications, where the RA may verify their identities. The RA facilitates the certificate issuance process (which may require applicants to appear in person), and the issued certificate is usually provided to the applicant (now the certificate holder) through the RA. RAs can also be set up as servers for electronic communication with users.
2. **Certificate Authority Application:** At the core of a CA is the application responsible for issuing certificates, which are electronically signed using the CA's private key. The CA's private key is its most valuable asset and must be adequately protected, often with the use of Hardware Security Modules (HSMs).
3. **User Database and Audit Records:** The CA maintains a database of users and audit records of its activities, including billing information for invoicing services provided.
4. **Certificate and CRL Archive:** The CA maintains an archive of issued certificates and Certificate Revocation Lists (CRLs).
    - The archive of issued certificates and CRLs can typically be accessed through a web interface.
    - The optional archive of issued certificates can also be accessed using the LDAP protocol.

In addition to issuing certificates, a CA may also provide:

5. **Certificate Revocation Mechanism:** A mechanism to revoke certificates when necessary.
6. **OCSP Responder:** An Online Certificate Status Protocol (OCSP) responder for providing online revocation information.

## 4.10 Certificate request

Figure 4.18 schematically illustrates the data structure of a certificate. The individual fields of the certificate must be accurately completed by the certification authority before digitally signing the certificate. An applicant can request a certificate in two ways: by submitting a data structure known as a certificate request or by not submitting any request at all.

Let us first consider the scenario where the applicant does not submit any data structure, and the certification authority handles the entire process. This approach is common today, particularly in cases such as remote electronic signature issuance (”Server Signing”). In such cases, the certificate is issued based on, for instance, two-factor authentication. However, this requires prior identification of the applicant and agreement to appropriate two-factor authentication methods.

In the first case, where the applicant submits an electronic certificate request, there are several options available: a self-signed certificate, PKCS#10, or CRMF.

### 4.10.1 Information related to the certificate request

The certificate application should include:

- **Applicant's identification data:** These details, as listed in the resulting certificate, will be included in the subject field of the certificate or in the Subject Alternative Name extension.
- **The public key**, including its parameters.
- **Proof of possession of the relevant private key**.
- **Additional data** that the user wishes to include in the certificate (e.g., key usage, extended key usage, etc.).
- It may contain evidence of the generation of key pair by a secure device (e.g., a smart card). Certificate applications do not have a separate item for this evidence. This evidence is often referred to as **"Proof of Possession of eID means"** in the EU.
- The data required for invoicing must also be provided (invoicing address, ID number, possibly bank account number for collection, etc.).
- **Passwords** for communication with the certification authority. These typically include:
  - A one-time password for issuing a certificate: This is especially useful if the certificate issuer wants to streamline the process and implement a "one visit is enough" strategy. During the first visit to the registration authority, the user provides all necessary documents. Instead of issuing the certificate immediately, the authority provides a one-time password for issuing the certificate. The user can then submit a request online from their home or office, including the one-time password. The CA verifies the provided data against the records from the first visit and checks the one-time password. If everything aligns, the CA issues the certificate and sends it to the user via the internet.
  - A one-time certificate revocation password: This is valuable in cases where a user's private key is compromised, such as when their PKI chip card is stolen along with the PIN written on it. In such situations, using a one-time password to revoke the certificate promptly is crucial. The user can contact the CA through various communication channels (phone, email, website, etc.) and request the certificate revocation, authenticating with the one-time password.
  - Permanent password for non-electronic communication with the CA: This password enables users to interact with the CA for various support needs.
  - In addition to a permanent password, users often set up a "phrase" as an additional security measure. This phrase, such as the mother-in-law's maiden name, serves as a backup if the user forgets both the one-time and permanent passwords. If they forget even the phrase, they typically have someone they can ask for assistance, provided they proceed cautiously.

### 4.10.2 Proof of possession of the relevant private key

The first question is why should the certificate applicant provide proof of possession of the relevant private key? At first glance, it may seem nonsensical why someone would ask for a public key certificate to be issued when they do not have a corresponding private key!

Imagine a certificate requester generating key pair identical to that generated by another requester that has already been issued a public key certificate. If the certificate authority issued certificates with the same public key to two different applicants, then it would be very unpleasant. Both would have the same private key and could, for example, sign documents for each other.

Cases where two users generate the same pair of data are almost impossible. However, this only applies if users use quality random number generators. Due to low-quality software, the impossible can become a reality. Certification authorities usually check whether they have already certified the submitted public key (the algorithm must also be identical). If so, and the certificate is valid, then a reasonable CA will immediately revoke the original certificate and reject the new request.

So, if the CA did not control the possession of the relevant private key, then the attacker would appear with a request for a certificate with the public key of the user he wants to harm, and the CA would revoke the certificate to the detriment of the user.

The most common type of proof of possession of the relevant private key is an electronic signature from a structure containing the public key. A certificate request, signed with a private key whose corresponding public key is included in the request, is referred to as a **Certificate Signing Request (CSR)**.

### 4.10.3 Self-signed certificate

A self-signed certificate is one issued by the certificate applicant himself. It is important to note that the root certificate is also a form of self-signed certificate. However, there is a significant security distinction between a root certificate and a self-signed certificate used for requesting a certificate.

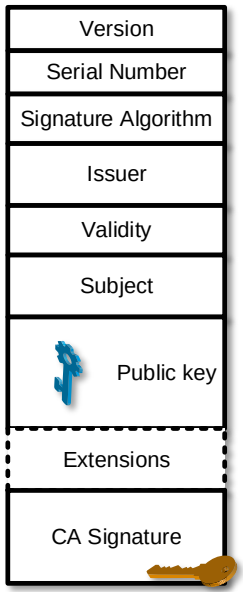



Figure 4.18 *Certificate Structure*

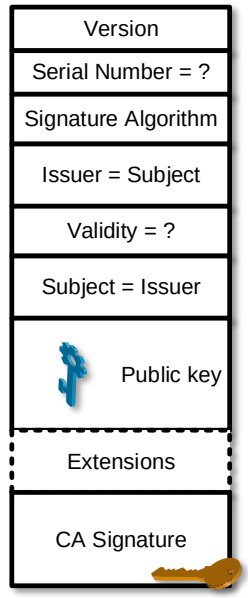



Figure 4.19 *Self-signed certificate*

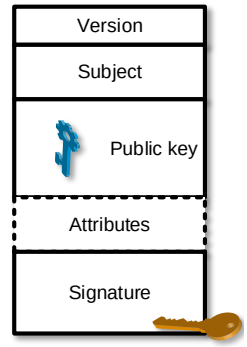



Figure 4.20 *PKCS#10*

Self-signed certificates are distinguished by having the Subject identical to the Issuer. Since these certificates are self-issued, the user includes their proposed subject information in both the Issuer and Subject fields.

Creating a certificate request in the form of a self-signed certificate is not straightforward. Certain fields, such as the certificate serial number and validity period, may pose challenges as the user may not have all the necessary information. In such cases, any placeholder values may be used.

To prove the user's possession of the private key, the self-signed certificate includes an electronic signature generated by the private key corresponding to the specified public key. Verification of this signature is performed using the public key specified in the self-signed certificate itself.

### 4.10.4 PEM

PEM (*Privacy Enhancement for Internet Electronic Mail*) stands out as the earliest internet email security system that utilized X.509 certificates, at the time version 1. Its inception dates back to RFC 989 [34] in February 1987 and underwent successive revisions until RFC 1421 [35] to RFC 1424 [36].

In PEM, X.509 self-signed certificates of version 1 were employed as certificate requests. RFC 1424, a longstanding reference in this context, specifies the placehoder values to populate the fields of the root certificate.

However, X.509 certificates are binary structures, whereas the PEM protocol was designed for communication via electronic mail, typically utilizing seven-bit ASCII. Consequently, all

binary data, including certificate requests, underwent Base64 encoding (refer to chapter 3.1) to convert them into a seven-bit format. The resulting output was delineated by lines containing a group of dashes, indicating the type of structure:

```
-----BEGIN PRIVACY-ENHANCED MESSAGE-----

<text>

-----END PRIVACY-ENHANCED MESSAGE-----
```

Despite the PEM protocol not gaining widespread adoption (with the PGP system being more prevalent at the time), the term ”PEM format” endured. It signifies that binary data has been encoded using the Base64 algorithm into a seven-bit form. Therefore, when requesting data ”in PEM format,” it does not necessarily imply Base64 encoding of the binary data.

### 4.10.5 PKCS#10

PKCS#10 (Figure 4.20) represents a certificate request. Simply put, PKCS#10 resembles a root certificate, except for omitting fields where we were uncertain what to fill in.

The remaining fields include (PKCS#10 entries):

- **The version**, where we specify the PKCS#10 structure (typically set to 0).
- **Subject**.
- **Public key**.
- **Attributes:** This section contains various attributes, including extensions that the applicant wishes to incorporate into the certificate.

Within the Attributes section, we can propose the extensions desired in the resulting certificate. However, it is subject to the CA’s certification policy how these attributes are handled. Notably, attributes like a one-time certificate revocation password will not appear in the issued certificate for obvious reasons.

A PKCS#10 request is signed with a private key corresponding to the public key specified in the request. Similar to root certificates, this serves as proof that the applicant possesses the relevant private key.

PKCS#10 originates from the RSA enterprise standard. It is closely associated with the RSA algorithm, known for enabling both encryption and electronic signature. However, PKCS#10 has a fundamental limitation: it may not be suitable for a public key that is not intended for electronic signature. In the case of a request for certification of the public key of the ECDH algorithm, it is important to remember that the same key pair can also be used for the ECDSA algorithm, and once again, PKCS#10 can be utilized.

Moreover, PKCS#10 requests may not be suitable in scenarios where the key pair is solely generated by the CA, meaning the user lacks the key pair at the time of the request.

Renewing PKCS#10 request certificates can also be challenging. During renewal, proof of holding both the new and old private keys is required to ensure identity continuity. However, this challenge can be addressed by embedding a PKCS#10 request into a CMS message signed

with the old key.

Initially adopted as RFC 2314 [37], the PKCS#10 standard was later revised and reissued as RFC 2986 [38].

### 4.10.6 CRMF

CRMF (Certificate Request Message Format) RFC 4211 [39], offers a significantly richer request format compared to PKCS#10. It addresses some of the limitations encountered with PKCS#10 requests. Notably, CRMF allows for the inclusion of billing information as part of the request.

### 4.10.7 CMC

The CMC (*Certificate Management over CMS*) protocol, as outlined in RFC 5272 [40], provides a comprehensive solution for certificate issuance dialogues. It facilitates the issuance and revocation of certificates. However, it does not define a specific transport protocol, a gap addressed by protocols such as EST and ACME.

The CMC protocol is also employed by Microsoft CA (MSCA) in certain scenarios, particularly when the certificate request requires countersigning by the RA operator.

### 4.10.8 CMP

Protocol CMP (*Certificate Management Protocol*) [41] is similar to CMC, but these procols are not compatible.

It mainly uses certificate requests in CRMF format, but can use PKCS#10 as well.

It is worth noting that **locomotives using ETCS** (*European Train Control System*) at least on level 2 should support CMP protocol [42].

## 4.11 Exercises

The goal of the exercise is to create a simple PKI for issuing certificates in IoT using OpenSSL. This PKI will consist of:

- Root CA.
- Issuing CA.
- One Sensor for which two certificates fill be issued (ECDH and ECDSA certificate).

Elliptic curve cryptography will be used, specifically the P-256 curve according to the FIPS PUB 186-3 standard (i.e., the X9.62/SECG curve over a 256-bit prime field). If we list, the supported curves in OpenSSL with the command:

```
$ openssl ecparam -list_curves
```

then we find that OpenSSL marks this curve as prime256v1.

The following paragraphs provide examples of OpenSSL commands used to create these certificates. Each certificate was generated in a separate directory. For this purpose, the following directory structure was created:

- `RootCA`
- `sub-CA`
- `Sensor`

**Note:** A text editor can change hyphens to other types of hyphens than OpenSSL accepts. This should be taken into account when copying the following commands to the command line.

### 4.11.1 Root CA

**Key pair on curve P-256 generation:**

```
$ cd RootCA
$ openssl ecparam -out ecparam.pem -name prime256v1
$ openssl genpkey -paramfile ecparam.pem -out ecdhkey.pem
```

The generated key pair can be printed with the command:

```
$ openssl pkey -in ecdhkey.pem -text -noout
```

**Root Certificate issuing:**

Create file root-256.cnf with the following contents:

```
[ req ]
default_keyfile = ecdhkey.pem
distinguished_name = req_distinguished_name
x509_extensions = v3_ca
prompt = no

[ req_distinguished_name ]
C = CZ
O = IoT
CN = SM-Test-RootCA

[ v3_ca ]
basicConstraints = CA:TRUE
keyUsage = cRLSign,keyCertSign
subjectKeyIdentifier = hash
```

A self-signed certificate can already be generated as a certificate request:

```
$ openssl req -x509 -new -key ecdhkey.pem -out root.cer
    -config root-256.cnf -days 73000
```

The certificate can be printed with the command:

```
$ openssl x509 -in root.cer -text
```

### 4.11.2 Sub-CA certificate

**Key pair on curve P-256 generation:**

```
$ cd ../sub-CA
$ openssl ecparam -out ecparam.pem -name prime256v1
$ openssl genpkey -paramfile ecparam.pem -out ecdhkey.pem
```

**Certificate request:**

Create file Sub-CA-256.cnf with the following contents:

```
[ req ]
default_keyfile = ecdhkey.pem
distinguished_name = req_distinguished_name
prompt = no

[ req_distinguished_name ]
C = CZ
O = IoT
CN = SM-Test-Sub-CA

[ v3_ca ]
basicConstraints = CA:TRUE
keyUsage = cRLSign,keyCertSign
subjectKeyIdentifier = hash
```

Create certificate request:

```
$ openssl req -new -config Sub-CA-256.cnf -key ecdhkey.pem
    -out Sub-CA-256.req
```

Create file Ext-file-256.cnf with the following contents:

```
[ v3_usr ]
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid,issuer
basicConstraints = critical,CA:true,pathlen:0
keyUsage = critical,cRLSign,keyCertSign
certificatePolicies = 1.2.3.4
```

Issue certificate:

```
$ openssl x509 -req -in Sub-CA-256.req -extfile Ext-file-256.cnf
    -extensions v3_usr -CA ../RootCA/root.cer
    -CAkey ../RootCA/ecdhkey.pem -out Sub-CA-256-certificate.cer
    -days 54750 -set_serial 01
```

The certificate can be printed with the command:

```
openssl x509 -in Sub-CA-256-certificate.cer -text
```

### 4.11.3 Sensor ECDH certificate

**Key pair on curve P-256 generation:**

```
$ cd ../sensor
$ openssl ecparam -out ecparam.pem -name prime256v1
$ openssl genpkey -paramfile ecparam.pem -out ecdhkey.pem
```

Create file sensor.cnf with the following contents:

```
[ req ]
default_keyfile = ecdhkey.pem
distinguished_name = req_distinguished_name
prompt = no

[ req_distinguished_name ]
C = CZ
O = IoT
OU = " Data central 1"
CN = 123456789ABCDEF

[ v3_usr ]
authorityKeyIdentifier = keyid,issuer
keyUsage = keyAgreement
```

**Request for a certificate:**

```
$ openssl req -new -config sensor.cnf -key ecdhkey.pem
    -out sensor.req
```

**Certificate issuance:**

Issue certificate:

```
$ openssl x509 -req -in sensor.req -extfile sensor.cnf
    -extensions v3_usr -CA ../subCA/Sub-CA-256-certificate.cer
    -CAkey ../subCA/ecdhkey.pem
    -out sensor-256-certificate.cer -days 36500 -set_serial 1001
```

The certificate can be printed with the command:

```
$ openssl x509 -in sensor-256-certificate.cer -text
```

### 4.11.4 Sensor ECDSA Certificate

**Key pair on curve P-256 generation:**

```
 $ openssl ecparam -out ecparam.pem -name prime256v1
 $ openssl genpkey -paramfile ecparam.pem -out ecdsakey.pem
```

**Certificate issuance:**

Create file sensorECDSA.cnf with the following contents:

```
[ req ]
default_keyfile = ecdhkey.pem
distinguished_name = req_distinguished_name
prompt = no

[ req_distinguished_name ]
C = CZ
O = IoT
OU = " Data central 1"
CN = 123456789ABCDEF

[ v3_usr ]
authorityKeyIdentifier = keyid,issuer
keyUsage = digitalSignature
```

**Request for a certificate:**

```
$ openssl req -new -config sensorECDSA.cnf -key ecdsakey.pem
-out sensor.req
```

Issue certificate:

```
$ openssl x509 -req -in sensor.req -extfile sensorECDSA.cnf
    -extensions v3_usr -CA ../subCA/Sub-CA-256-certificate.cer
    -CAkey ../subCA/ecdhkey.pem -out sensor-256-certificateECDSA.cer
    -days 36500 -set_serial 1002
```

The certificate can be printed with the command:

```
$ openssl x509 -in sensor-256-certificateECDSA.cer -text
```

## 4.12 Certificate revocation

A certificate can lose its validity when its originally declared period of validity expires, i.e., when the notAfter time specified in the certificate passes. Before the certificate's originally declared validity period expires, the certificate may be revoked or its validity may be suspended. However, the suspension of certificates is rarely used.

Revoking a certificate works similar to revoking a payment card. First, someone's payment card is stolen without noticing, they may later discover that the card has been stolen and subsequently report the loss to its issuer. The issuer then processes this information, resulting in the card being added to the stop list.

Every merchant is obligated to check whether a card is on the stop list before accepting payment by card (this is done automatically by the payment terminal). If the merchant fails to check and the card is on the stop list, then the costs are charged to the merchant. The discussion usually revolves around responsibility for the card between the time it is stolen and the time it is added to the stop list. This interim period may be covered by the card issuer as a service to the cardholder, or the card may be insured. For certificates, the critical moment is the initial publication of the revoked certificate on the Certificate Revocation List (CRL).

Revoked certificates are published on the Certificate Revocation List (CRL) for the entire period of their originally declared validity. Whether the certification authority allows certificates to be revoked at all, or with what period it issues the CRL, is declared by the certification authority in the ”Certification Policy” document.

**Note:** In the case of qualified certificates, it is recommended to maintain expired certificates on the Certificate Revocation List (CRL), at least to a fixed date.

We distinguish several types of CRL:

- By CRL contents:
  - The **full CRL** contains the identification of all revoked certificates whose original validity period has not yet expired.
  - A **delta (incremental) CRL** contains the identification of only those certificates revoked since the last full CRL.
  - A **restricted CRL** contains a subset of the full CRL. For example, it may include only revoked user certificates, revoked CA certificates, or certificates revoked for specific reasons.
- By CRL issuer:
  - The **direct CRL**, unlike indirect CRL, is issued by the certification authority that also issued the revoked certificates.
  - An **indirect CRL** is issued by an authority other than the one that issued the revoked certificates. This authority is referred to as a “CRL issuing authority” or ”CRL server”.

**Note:** We most often encounter full, direct CRLs. Therefore, when a CRL is mentioned without any qualifiers, it is understood to refer to a full, direct CRL.

The certificate revocation request does not have to be made electronically; the CA may require the user’s personal participation. If done electronically, the request may be electronically signed with the private key of the revoked certificate. Alternatively, a one-time password can be used for revocation.

Figure 4.21 illustrates the life cycle of a certificate, including the certificate revocation process. First, the private key is compromised, referred to as the Invalidity date. Then the user reports it to the CA, known as the Revocation date. If the CA finds the revocation request justified, it publishes the certificate on the nearest issued CRL (CRL update).

Even if the affected users do not notice the compromise of their private key for several days, they expect the CA to immediately publish their certificate on the CRL. Certification authorities usually choose one of two strategies for issuing CRLs:

- They issue CRLs at regular intervals, so a user with a compromised key has to wait until the next CRL is issued.
- CRLs are issued after a certificate has been revoked. However, this requires other users to download and process the CRL before using each certificate, which slows them down because the CRL takes time to download.

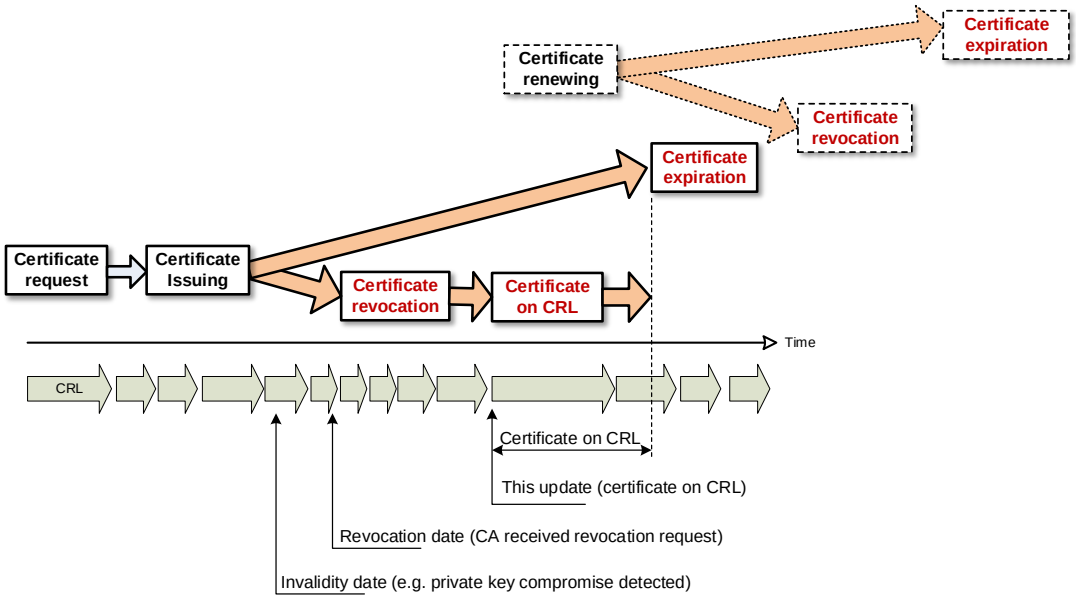



Figure 4.21 *Certificate lifecycle and CRL issuance*

However, in reality, the most common reason for revoking a certificate is not the compromise of the private key but the termination of the employment of the employee to whom the certificate was issued. Employers can revoke such certificates promptly.

### 4.12.1 Certificate revocation request

When revoking a certificate, it is not just about adhering to technical standards; speed is crucial. Therefore, the certification policies of individual CAs tend to be relatively flexible in this area. There is usually no standardized structure for a certificate revocation request. The method used to revoke a certificate varies depending on the circumstances.

## 4.13 CRL

A CRL serves as a CA's official public notice board where revoked certificates are published. Once a certificate is revoked and listed on a CRL, it should continue to appear on all subsequent CRLs until its original validity period expires.

However, there is a complication: some CAs allow for certificate validity to be suspended, which complicates the process. When a suspended certificate is reinstated, it must also be listed on the CRL until its original expiration date. The lesson here is that it is preferable to avoid certification policies that permit certificate suspensions. It is better to adhere to the principle that once a certificate has been revoked, it cannot be reinstated.

Revoked certificates are identified on the CRL by their serial numbers, which may appear abstract to the average user. The CRL is designed for computer processing. In the case of indirect CRLs, the revoked certificate must be identified not only by its serial number but also by the CA name, using the Issuer item.

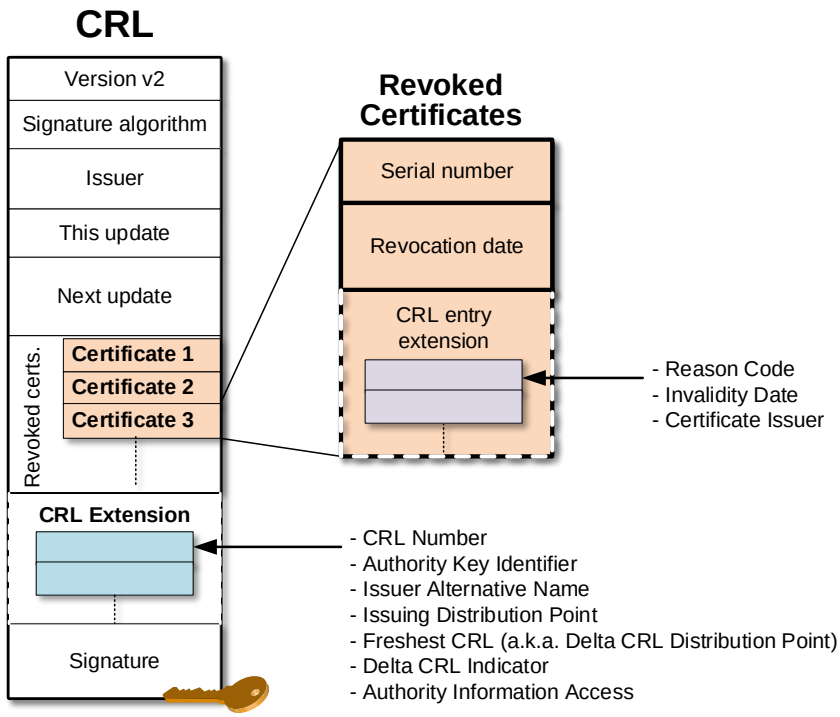



Figure 4.22 *CRL*

The structure of the CRL was specified by the X.509 standard. On the Internet, we use the CRL structure described in RFC 5280. The CRL structure is similar to that of a certificate (Figure 4.22):

- The first entry in the CRL is the **Version**, as prescribed by RFC 5280, it must be included in the CRL and must have the value 1 (i.e., version 2).
- The **Signature Algorithm** entry identifies the algorithm used for the CRL signature scheme, similar to how it is done for certificates.
- The **Issuer** entry identifies the entity that issued this CRL, akin to a certificate.
- **This Update** entry specifies the time of issuance of this CRL.
- The **Next Update** entry indicates the latest date and time for the issuance of the next CRL. A subsequent CRL may be issued before or at this specified time. However, a subsequent CRL must not be issued later than the date and time specified in this entry of the previous CRL.
- The **Revoked certificates** item contains a list of revoked certificates. For each revoked certificate, it contains a sequence of data:
  - **Serial number** of the revoked certificate, which, together with the Issuer item, uniquely identifies the revoked certificate (if it is not an indirect CRL, then the issuer is taken from the extension of the revoked certificates item).
  - **Revocation date**, date and time when the certificate holder submits the certificate revocation request.

  - The **revoked certificate extension**s - optional entry contains single extensions related to the revocation of this certificate.

- Optional item CRL extension contains the CRL extension.

Note that there are two types of extensions in the CRL:

- **CRL entry extension** which contains information about the revocation of a specific certificate.
- **CRL Extensions** – CRL extensions as a whole, which are similar to certificate extensions.

### 4.13.1 CRL Extension

CRL extensions are similar to certificate extensions. Again, these extensions may be marked as critical; when processing them, the processing software must be aware of what such a serious extension means.

### 4.13.2 CRL entry extension

In addition to the certificate revocation itself, other information related to this revocation may be of interest. This information may include CRL entry extensions. These extensions should not be marked as critical.

Revocation (CRL) reason code:

- **Key Compromise** – the private key belonging to this certificate was stolen (e.g., the entire computer was stolen or the smart card was stolen, etc.).
- **CA Compromise** – CA's private key was stolen. The consequence of revocation of a CA certificate is the revocation of all certificates issued by it, because after the revocation of a CA certificate, it can no longer be considered trustworthy.

Table 4.5 *CRL Extensions Overview*

| CRL Extension | Mandatory | Critical | Meaning |
|---|---|---|---|
| Authority Key Identifier | Yes | No | See the certificate extension of the same name. |
| Issuer Alternatives Name | No | No | See the certificate extension of the same name. |
| Authority Information Access | No | No | See the certificate extension of the same name. |
| CRL Number | Yes | No | The CA uses this extension to number the issued CRLs. Numbering is done using monotonically increasing sequences of numbers. If we find that the certificate is on the CRL, then we immediately get worried if it was not on the previous CRLs and we still used it. So, we need to find out what the previous CRL was. We can find this out just by looking at the CRL List Number extension. The previous CRL must have the previous order in the sequence of numbers assigned by the CRL. |
| Delta CRL Indicator | Only in delta CRL | Yes | This extension indicates that this is a delta CRL. A delta CRL does not contain the full CRL, but only changes from the previous full CRL. An incremental CRL also contains the number of the full CRL from which increments are taken. |
| Issuing Distribution Point | No | Yes | This extension indicates that this is not a full CRL, but a restricted CRL containing a subset of the full CRL. Optionally, it contains links to other full CRLs. |
| Freshest CRL | No | No | Provides the location at which a delta CRL for this complete CRL can be found. |

- **Affiliation Changed** – the identification data of the certificate holder have changed. For example, the content of the attribute Organization (O) ceased to be true because the employee terminated the employment relationship.
- **Superseded** – the certificate has been replaced by a newer certificate (e.g., some certificate extension values had to be changed).
- **Cessation of Operation** – the subject of the certificate has been decommissioned. For example, the web server has been replaced by another server with a new one.
- **Privilege Withdrawn** – permission revoked.
- **Certificate Hold** – the validity of the certificate has been suspended (e.g. employee takes parental leave). The peculiarity of this reason is that the certificate can be reac-

tivated (subsequently it can be declared valid).

- **Remove from CRL** - when certificate is reinstated after being on hold, it must be included on all subsequent CRLs with this reason code.

Table 4.6 *CRL Entry Extensions Overview*

| CRL Entry Extension | Mandatory | Critical | Meaning |
|---|---|---|---|
| Reason Code | No | No | Identifies the reason for the certificate revocation. |
| Invalidity Date | No | No | Contains the date and time when the loss or disclosure of the private key was detected, which is the reason for the certificate revocation. |
| Certificate Issuer | No | Yes | This indirect CRL extension contains the unique name of the issuer of the revoked certificate. If the first CRL entry does not contain this extension, then it is a direct CRL. If this extension does not contain another CRL entry, then the issuer is assumed to be the same issuer as in the previous entry. |

## 4.14 Exercises

In your browser open some website accessed via HTTPS. Show details in the server certificate by clicking on the lock (Figure 4.23). Copy the link of one CRL distribution point into the clipboard and paste it in location in your browser.

Browser download CRL. This CRL is displayed in MS Windows on Figure 4.24.

**Note:** Since CRLs are downloaded over HTTP (not HTTPS), the browser may block the download.

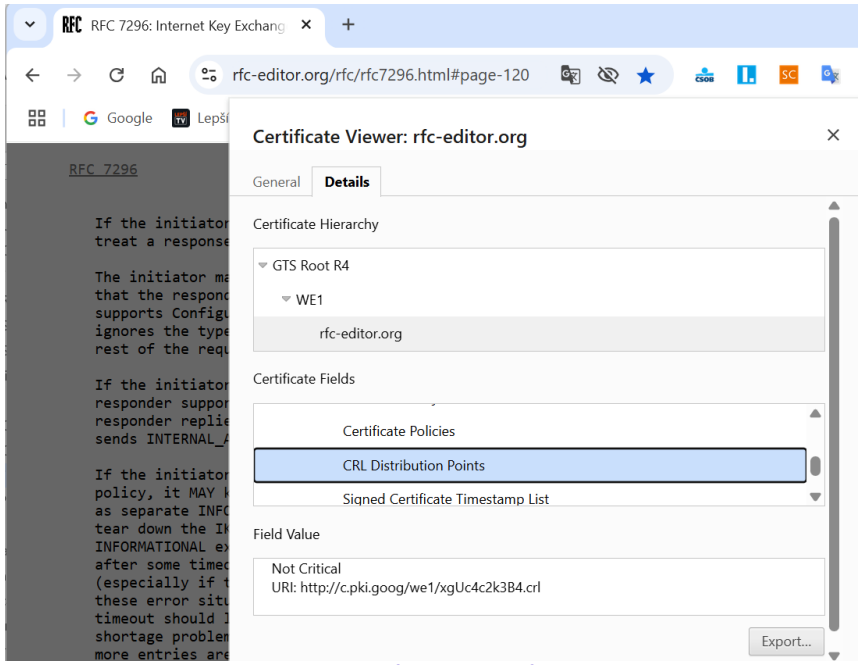



Figure 4.23 *Details of server certificate in Chrome*

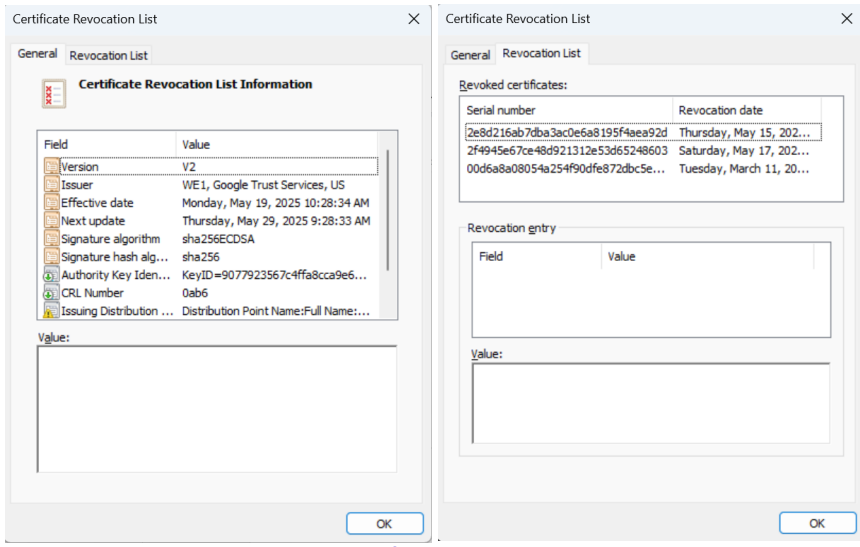



Figure 4.24 *Example of CRL displayed in MS Windows*

## 4.15 OCSP

While Certificate Revocation Lists (CRLs) are issued only after the time has passed, requiring the relying party to process the entire list to verify a certificate's status, the Online Certificate Status Protocol (OCSP) allows real-time querying of the status of individual certificates.

OCSP is specified by RFC 6960 [43]. A CA delegates the authority to provide the status of its certificates online to an entity called an OCSP responder. The CA accomplishes this delegation by issuing an OCSP responder certificate that includes the Extended Key Usage certificate

extension OID for the OCSP responder.

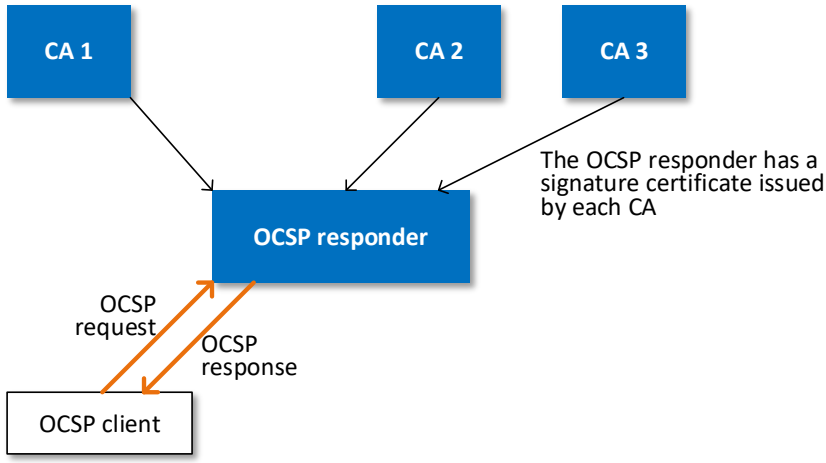



Figure 4.25 *OCSP communication*

An OCSP responder can serve multiple CAs (see Figure 4.25). However, each CA being served must issue a certificate to the OCSP responder, which the responder uses to sign OCSP responses specific to that CA.

The structure of OCSP requests and responses is illustrated in Figure 4.26.

One challenge is verifying that the OCSP responder certificate has not been revoked. The most common strategy is for the CA to include a NoCheck extension in the OCSP certificate extension, indicating that certificate verification should not be performed. Additionally, these certificates are typically issued for shorter periods, but generally longer than the CRL issuance period.

The OCSP responder has the capability to archive its responses and respond to requests for an already expired certificate. This feature is particularly useful for verifying archived electronically signed documents.

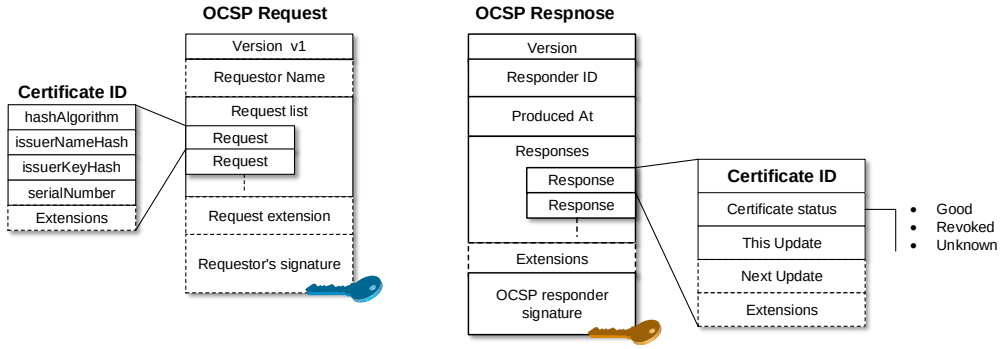



Figure 4.26 *OCSP*

# 5 ACME

ACME *(Automatic Certificate Management Environment)* is a protocol designed to simplify the lives of server and network administrators.

Specified in RFC 8555 [44], ACME is primarily intended for the automated issuance of DV (Domain Validation) certificates, especially for HTTPS servers. The ACME server acts as an online registration authority.

**Domain Validation (DV)** certificates are certificates, where when issued, the applicant must demonstrate control over an identifier, typically the DNS name of the server.

**Note:** It is essential to note that if the identifier is under the control of an attacker, the ACME server might facilitate the issuance of a certificate to the attacker.

For other types of certificates that demand more rigorous proof of identifier ownership (such as **Extended Validation (EV)** certificates or **qualified certificates for web servers**), ACME offers the External Account Binding feature.

The principle of ACME is straightforward:

1. The administrator of the HTTPS server (referred to as the ”ACME object operator”) creates an account on the ACME server using the ACME client.
2. The ACME object operator uses their account to request certificate issuance from the ACME server. This process involves the following steps:
    (a) The ACME server verifies whether the user associated with the account controls the specified identifier(s) (e.g., DNS names or URIs).
    (b) The ACME server facilitates the certificate issuance. In the ACME protocol, one or more ACME operators can request certificate issuance for specific identifiers or multiple identifiers.

## 5.1 External Account Binding

External Account Binding enhances ACME capabilities by enabling the verification of the certificate applicant’s identity. This feature is particularly suitable for obtaining Extended Validation (EV) certificates or qualified certificates for websites. It is especially beneficial for intranets, where External Account Binding can be integrated with the organization’s Identity Management System. In this scenario, the operator is designated as the manager of specific identifiers.

The process involves the individual applicant visiting the Registration Authority (RA) or Certification Authority (CA) before the issuance of the first certificate. During this visit, the applicant proves his/her identity and demonstrates authorization to apply for certificates for a particular website. After that he receives EAB code, which he use during account creation. This code enables ACME server to bind ACME account to specific individual.

## 5.2 Security of ACME communication

Communication between ACME client and ACME server uses HTTPS protocol, which means confidentiality and integrity of sent messages is secured by TLS protocol.

Client sends his request in JWS format, meaning they are signed with his private key, which means autenticity is also secured. For security against replay attack each message also contains nonce.

## 5.3 Creating an account on the ACME server

ACME object operator refers to the individual requesting the issuance of a website certificate. However, as ACME appears to be a more comprehensive protocol, it utilizes the broader term ”Identifier” instead of the specific term ”website.”

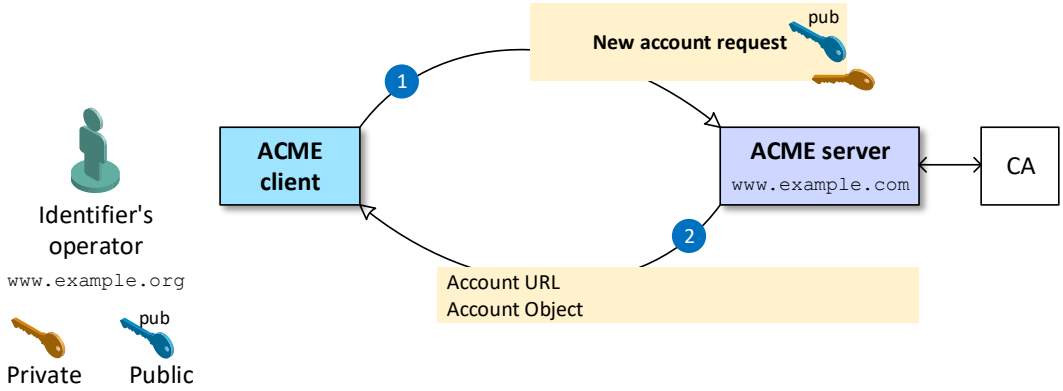



Figure 5.1 *Creating an ACME account*

The ACME object operator (the prospective certificate applicant, e.g., www.example.org) initially establishes an account on the ACME server. The ACME server does not authenticate the identity of the applicant (the account is essentially anonymous, unless EAB is used).

A single account can cater to multiple identifiers, and conversely, multiple accounts can serve the same identifier.

Procedure for creating an account on the ACME server:

1. The ACME object operator (ACME client) generates key pair for the account being created. The operator will work exclusively with the public key itself, not its certificate.
2. The object operator sends a new account request to the ACME server, including:
    - Public key.

- Contact information (optional).
- Agreement to terms of service (ToS) (optional).
- Additional data, such as payment information (optional).
- External Account Binding data (optional).
- Signature with the generated private key.

3. The server responds with:
   - Account URL: the identifier for the account.
   - Account object (used as the user's public key identifier).

## 5.4 Certificate issuance

Now, the object operator has successfully created an account on the ACME server, and the public key corresponding to this account is securely stored on the ACME server. The ACME server operates as an anonymous HTTPS server. Each client request is authenticated through an electronic signature, which is verified using the public key stored in the client's account on the ACME server.

Communication between the ACME client and ACME server involves several HTTP dialogs:

- Submitting an order for a certificate to be issued.
- Providing proof of control for all identifiers requested in the certificate.
- Finalizing the order by submitting a Certificate Signing Request (CSR).
- Awaiting issuance and downloading the issued certificate.

### 5.4.1 Order

The ACME client sends an order to the ACME server to issue a certificate for a specific identifier(s), for example, the DNS name www.example.org or *.example.org (the asterisk must be followed by a dot), i.e., not yet sending a certificate request.

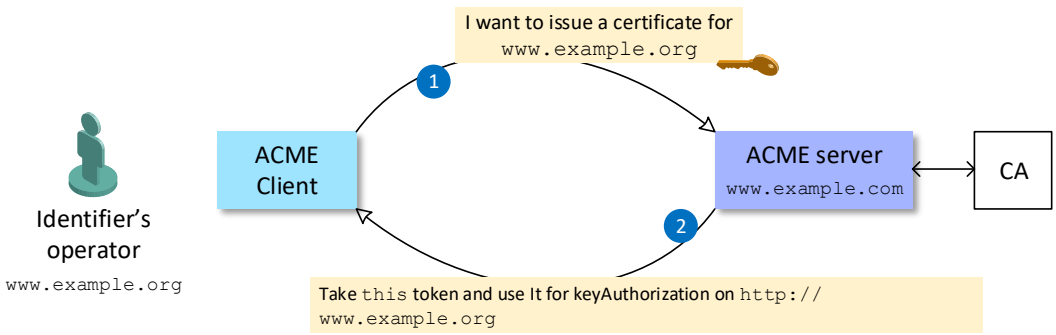



Figure 5.2 *Order*

In its response, the ACME server specifies to the requester:

1. The challenge that the applicant for the certificate has to prove that he has the identifier under his control. Figure 5.2 shows the case (referred to as HTTP-01 in the standard) where the requester proves that he has under his control the identifier www.example.org by being able to

   http://www.example.org/.*well-known/acme-challenge/'ACME_server_sent_random_number'*

   publish the keyAuthorization value, which is created by concatenating the random token number (sent by the ACME server) and the SHA-256 hash of the ACME public key of the requester's account.

2. Once the operator fulfills the challenge, they inform the ACME server, which can check the challenge status.

### 5.4.2 Proof of control

This is a verification that the operator (working with the ACME client) has control over all the identifiers for which the certificate is to be issued. If multiple identifiers should be included in the certificate (e.g., in Subject Alternative Name), then for each identifier, verification is performed separately using different challenges. Individual challenges may use different authentication mechanisms.

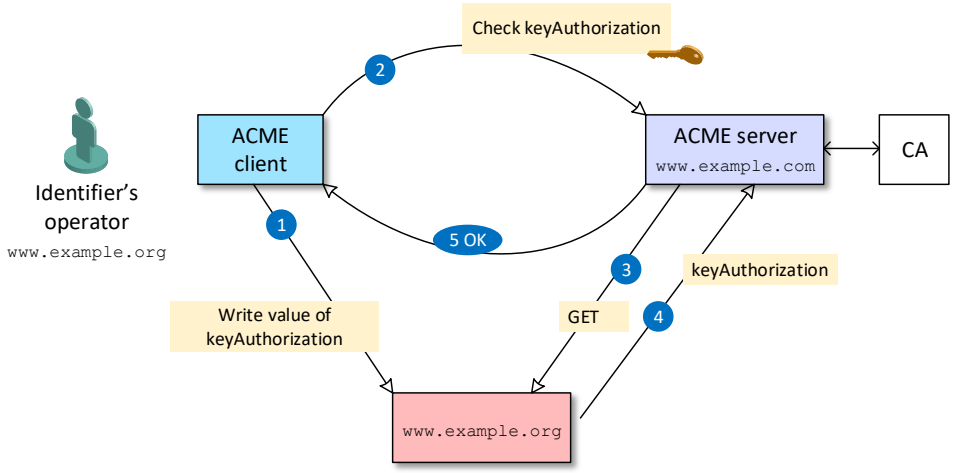



Figure 5.3 *Proof of control*

In practice, there are the following types of challenges used to verify identifiers:

- HTTP-01 - the HTTPS server is under operator control (Figure 5.3).
- DNS-01 - DNS record of identifier (e.g., web server) is under operator control.

Figure 5.3 shows authentication using an HTTP-01 challenge. The ACME client:

- Publishes the keyAuthorization value on www.example.org.
- Informs ACME server about this.

The server then:

- Verifies the electronic signature of the client's message.
- Validates that the correct keyAuthorization is published.
- Returns OK to the client (the message is more complicated, see ACME Protocol).

### 5.4.3 Sending a request for a certificate

Now the operator can send a certificate request (CSR in form PKCS#10) and the server returns the certificate (Figure 5.4).

The ACME server can either return the issued certificate directly (synchronously) or simply return an "OK" response, requiring the client to periodically inquire whether the certificate has already been issued.

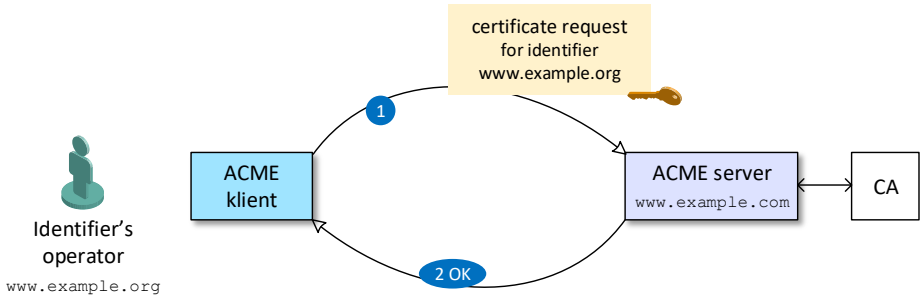



Figure 5.4 *Certificate request sending*

### 5.4.4 Certificate revocation

The client revokes the certificate with a message signed either with the private key of its ACME account or with the private key of the revoked certificate.

# 6 CMS

Cryptographic Message Syntax (CMS) [45] is a standard specifying data message security originally introduced by RSA Security Inc. as the **PKCS#7 standard**.

A data message is a block or set of data that has a clear beginning and end. The CMS, therefore, does not serve to secure the data flow (the data flow is secured by protocols such as TLS). The CMS standard does not address the internal structure of secured data messages; it treats the data message as a black box, merely as a string of bytes.

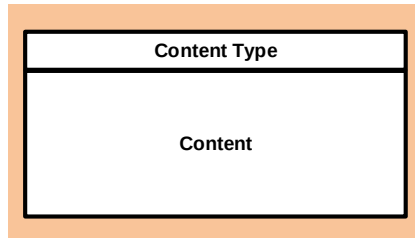



Figure 6.1 *Content Info*

CMS provides complete message security, addressing not only electronic signatures but also encryption and message authentication. Different types of CMS messages are utilized for various security purposes (refer to Table 6.1).

Table 6.1 *Content Types of CMS messages*

| **Content Type** | **Importance** |
|---|---|
| Data | Data (usually the type of message entering security) |
| Signed Data | The signed-data content type consists of a content of any type and zero or more signature values. Any number of signers in parallel can sign any type of content. |
| Enveloped Data | The enveloped-data content type consists of an encrypted content of any type and encrypted content-encryption keys for one or more recipients. |
| Digested Data | The digested-data content type consists of content of any type and a message digest of the content. |
| Encrypted Data | The encrypted data content type consists of encrypted content of any type. Unlike the enveloped-data content type, the encrypted-data content type has neither recipients nor encrypted content-encryption keys. Keys MUST be managed by other means. |
| Authenticated Data | The authenticated-data content type consists of content of any type, a message authentication code (MAC), and encrypted authentication keys for one or more recipients. |

Result of securing a data message with the CMS protocol is a Content Info structure (CMS message), which consists of two items (refer to Figure 6.1): Content Type and Content.

Thanks to this design, determining the type of content (i.e., the type of CMS message) is straightforward by examining the first item of each CMS message. The various types of CMS messages are detailed in Table 6.1.

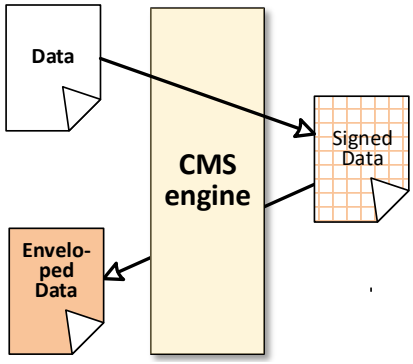



Figure 6.2 *CMS engine*

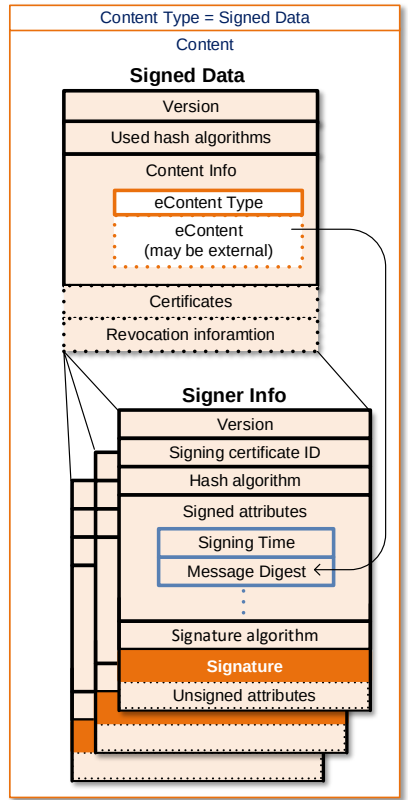



Figure 6.3 *Signed Data*

How does CMS work? Perhaps the best illustrative analogy is the CMS machine. Figure 6.2 depicts a CMS machine, which takes an unsecured message (Data) as input and outputs a secured message, with the specific security method determined by the user. The CMS machine offers options such as electronic signature, electronic envelope (hybrid encryption), data digest, message authentication, or simple message encryption.

However, in many cases, merely signing or encrypting the message is insufficient; both signing and encrypting may be desired. This can be achieved by invoking the CMS machine multiple times in succession (as shown in Figure 6.2). In secure conferences, for instance, the CMS machine may be invoked three times.

The CMS architecture is highly generic, to the extent that many current protocols do not directly address the security of their message content. Instead, they delegate security responsibilities to the CMS machine. This philosophy is employed, for instance, by time stamps (refer to Figure 7 Timestamps).

## 6.1 Signed Data structure

For the electronic signature, the CMS standard uses Signed Data structure. CMS Signed Data is an example of a composite signature mentioned in subsection 2.9.4 Composite electronic signature.

The Signed Data structure contains a set of signatures (set of Signer Info structures). Each element of this set contains one (parallel) signature.

A detailed description of the Signed Data structure is shown schematically in Figure 6.3. The meaning of individual items is as follows:

- The **Version** item contains the version of the structure. The original PKCS#7 standard used version 1, today there are a number of versions.
- The item **Hash algorithm** contains a set of hash algorithm identifiers used to calculate the hash.
- The **Content Info** item may contain data to be signed by CMS engine, e.g., a type Data. The word ”may” is important. In the case of electronic signatures, we distinguish between external and internal electronic signatures:
  - In the case of an internal electronic signature, the content of the data to be signed is placed in item eContent (encapsulated content).
  - In the case of an external signature, the data to be signed are located externally, e.g., in separate file (Content Info contains eContent Type only). Practically, this means that if the message is stored in an external file, we have the message signature stored in another separate file. In the case of electronically signed e-mail (S/MIME), the e-mail may consist of two parts: the message itself and the electronic signature. The advantage of external signatures is that the original message can be easily viewed (processed) even with software that does not support electronic signatures.
- The **Certificates** item may contain a set of certificates that will be useful in the verification of electronic signatures, i.e., to build a Certificate chain to a trusted anchor. However, nowhere is it said that there must be all the certificates that the user will need for electronic signature verification, and there may be certificates that the user does not need at all. When creating such a report, it is good to realize that it will need to be verified in, say, 10 years, and many certificates that have already expired by that time may be difficult to procure. Trusted anchors are not placed here.
- The **Revocation Information** item may contain certificate revocation information (e.g., CRL or OCSP response). The same applies to CRLs as was mentioned for certificates. Again, it is necessary to assume the role of the one who will verify the electronic signature years later, and therefore will look for the relevant CRLs valid at the time of signing the message. Even if the certificates were still valid, the certificate extension CRL Distribution Points mentioned in them refer to the current CRL, not to the CRL valid at the time of signing!
- A set of **electronic signatures**. Yes, I already wanted to write ”... contains a set of elec-

tronic signatures of input (encapsulated) data", which would not be correct, because it would often not be enough. The electronic signature does not have to be calculated not only from the input data, but also, for example, from the signing date and time – more generally from the so-called Signed attributes. So, data to be signed are often not eContent only, but Signed Attributes. One of Signed Attributes is message imprint. Each signature is of type Signer Info. The set can also be empty in the case of so-called degenerate Signed Data messages, which are used to distribute a set of certificates in .p7b format.

Generally, a Signed Data message can contain multiple electronic signatures. This is also common with handwritten signatures, for example, two signatures may also be required by the bank for payment orders above the specified limit. Even more common is the signing of contracts. A contract is a relationship between at least two parties, so the contract will also have at least two signatures, etc.

### 6.1.1 Signer Info structure

An electronic signature (Signer Info structure) consists of the following items:

- The **Version** item contains the version of this data structure.
- Item **Signing certificate ID** contains the identification of the certificate for the verification of this electronic signature. The identification can be either the subject & serial number of the certificate or the *Subject Key Identifier*.
- The item **Hash algorithm** contains the hash algorithm, which is used to calculate the hash of the data to be signed.
- The item **Signed attributes** contains information that will be included in the calculation of the electronic signature.
- The item **Signature algorithm** specify the asymmetric algorithm used to calculate the value of electronic signature, including its relevant parameters.
- The **Signature** item contains the relevant electronic signature, i.e., a hash "encrypted" with an asymmetric cipher. Perhaps the biggest surprise is the way the hash is calculated to create an electronic signature. The problem is that it is necessary to calculate the hash not only from the input data itself, but also from the *Signed Attributes*. Therefore, two cases are distinguished, and in each of them the hash is calculated from something else:
  - The Signer Info does not contain any Signed attributes. In this case, the hash is calculated from the input data.
  - The Signer Info contains Signed attributes. In this case, the hash is not calculated directly from the input data, but from the *Signed attributes*. First, the hash from the signed message is calculated and it is inserted into the signed attribute *Message digest*. Subsequently, a hash is calculated from all the *Signed attributes*, from which a value of electronic signature is only created. The hash of the signed message is thus indirectly included in the hash from which the electronic signature is created. In this case, at least the following Signed attributes must be

present: *Message Digest, Content Type and Signing Time*.

- The optional item Unsigned attributes is intended for unsigned electronic signature attributes, i.e., attributes that are not included in the calculation of the electronic signature. The syntax is similar to Signed attributes.

What about unsigned attributes? A classic case of an unsigned attribute is a Countersignature (serial signature). We know the countersignature from practice. When the president signs an international treaty with a foreign president (presidents attach parallel signatures), then the treaty is usually not yet valid. It becomes valid when it is approved by the parliaments, that is, when the leaders of the parliament countersign the signatures of their presidents, i.e., when they serially attach their signature to each parallel signature.

In the case of CMS. Each (parallel) signature can contain unsigned Countersignature attributes, i.e., it can contain other serial signatures (signature signatures).

Table 6.2 *Some signed attributes*

| Attribute | Importance |
|---|---|
| Content Type | Specifies the content type of the signed or authenticated message. |
| Message Digest | Contains a hash of the signed message. |
| Signing Time | Specifies the time the document was signed. But this is not a time stamp! This time can be set by the signatory as it is taken from the computer's system time! |
| S/MIME Capabilities | Specifies the signature algorithms, symmetric encryption algorithms, and other capabilities preferred by the software that created the S/MIME message. |
| sMIMEEncryptionKeyPreference | Specifies the certificate to use for encrypting the email response. |
| Signing Certificate | Allows the identification of the certificate used for signature verification to be included in the signature calculation. |
| Other Signing Certificate | Same meaning as the Signing Certificate attribute, but also allows using a hash other than SHA-1. |
| Content Timestamp | Contains the timestamp of the document (not of the signature). |

Table 6.3 *Some unsigned attributess*

| Attribute | Importance |
|---|---|
| Countersignature | It contains another serial electronic signature of the message. |
| Signature timestamp | Contains the timestamp from the signature, which is proof of the existence of a signature at a specific time. |

### 6.1.2 Certificates export

A degenerate case of an electronically signed CMS message is a message that lacks an electronic signature, meaning it does not contain a Signer Info structure. In essence, such a CMS message consists solely of certificates and possibly revocation information. This type of signed message is commonly used for certificate distribution. Unlike other file formats used for carrying certificates, this degenerate CMS message offers a significant advantage: it can include not just one certificate, but an entire set of certificates.

In Windows, one can export the certificate along with the certificate chain by selecting ”Cryptographic Message Standard – PKCS #7 Certificates (.P7B)”.

## 6.2 Enveloped Data

The electronic envelope contains encrypted content and, for each recipient, an encrypted content-encryption key (sometimes referred to as session key) with which the content of the message was encrypted.

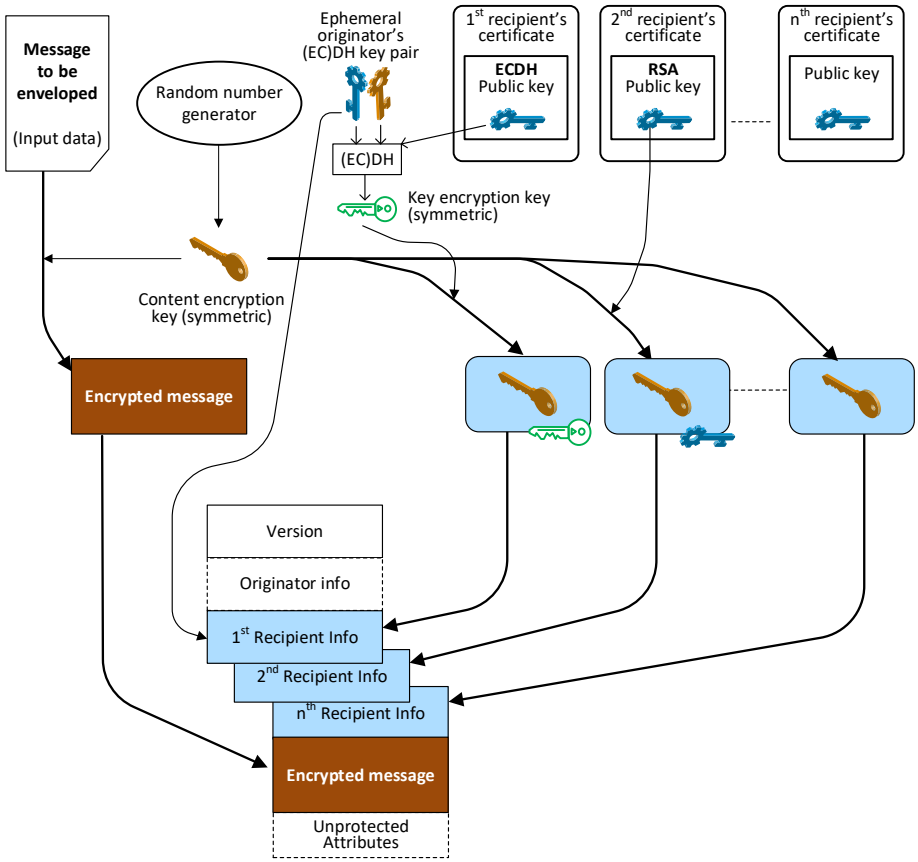



Figure 6.4 *Enveloped Data*

The principle of the electronic envelope:

- In the case of the RSA algorithm the content encryption key is encrypted by recipient's

public key.

- In the case of the (EC)DH algorithms content encryption key is encrypted by symmetric key calculated from the (EC)DH key exchange. This key exchange includes private keys of the signer and the public key of the recipient. Typically, the sender generates ephemeral key pair. The sender's public key is not in included in the form of certificate but only in its raw form. On the contrary, recipient's public key is extracted from his certificate.

Enveloped Data message has the following structure (Figure 6.4):

- The Version item contains the version of the message format.
- The optional Sender Information item is for the sender's certificates and CRLs.
- Recipient Info items contain for each recipient one instance of the Recipient Info structure, i.e., it contains either the encrypted content-encryption key or at least its identification.
- The Encrypted data item contains its encrypted message.
- The item Unprotected attributes is similar to Unsigned attributes in the case of a digitally signed message.

# 7 Timestamps

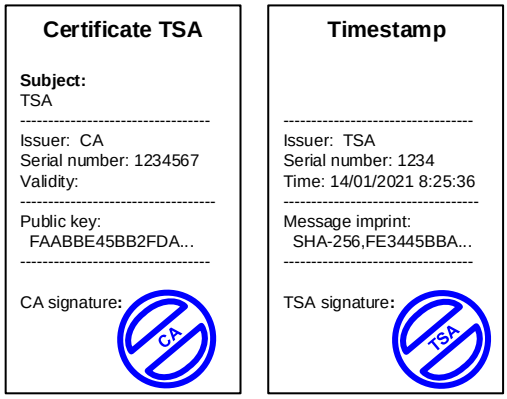



Figure 7.1 *Certificate and Timestamp*

A public key certificate is a data structure that binds the certificate holder's identification with his public key. This binding is confirmed by the electronic signature of an independent third party, which we call a certification authority. A timestamp (or time stamp or time-stamp), on the other hand, is a similar data structure in some way, but it binds the hash of the message (document) to a certain time. Timestamps can be used as proof that a message (document) existed at a certain time.

A timestamp is a data structure that contains, among other things, the time, an imprint of the document, the issuer of the time stamp and a serial number. All of this is confirmed by signature of an independent third party – the Timestamping Authority (TSA). The timestamp thus serves as proof that the document (more precisely, an imprint from the document) existed at a given time. The document is represented in the timestamp using the so-called message imprint, which consists of a pair: a hash of the document and the hash algorithm used to calculate it.

Since the TSA does not verify the identity of the document owner, a timestamp does not include the identification of the applicant (or owner). Therefore, a timestamp does not prove that a document was in the possession of a specific person at the time of issuance; it only indicates that the document existed at a specific time.

Let us see what we can use the timestamp for in practice.

**Example 1 – timestamping of documents**

The application that immediately comes to mind is document stamping: we create a timestamp for an existing document and a document + timestamp pair is created. The timestamp not only serves us as an indication of the existence of the document in time, but also encapsulates the document in time. Any subsequent change to the document would invalidate the timestamp.

Most TSAs provide software with which we can create timestamps for files, which we then save in separate files (usually with the .tsr extension). However, the practical significance of

stamping the documents themselves is limited.

**Example 2 – timestamping of audit records** Stamping audit logs is one of the more complicated applications of the previous example.

If a hacker gets into a computer and does something there, s/he will try to cover his/her tracks by fixing the audit trail. By regularly stamping previous audit records, we make it impossible for an attacker to change records without being noticed. The only option for the attacker is to remove the timestamps as well, which, of course, also leaves traces because the period of inserted audit records with a timestamp is violated.

**Example 3 – timestamping electronic signature**

If we send a digitally signed payment order to the bank, which the bank archives for, e.g., 10 years, then a problematic situation occurs after the certificate that was used for the signature expires (e.g., after a year). The sender may declare that he did not send such an order. Why? He can claim that the bank had his public key at his disposal for a long enough time and during this time was looking for the corresponding private key, with which he signed the false payment order. However, if the bank, before archiving the payment order, has a timestamp created from the electronic signature of the payment order, it has proof that the payment order was actually executed during the period of validity of the user's certificate and thus by the user himself.

From this fictitious example, several conclusions emerge for digitally signed documents:

- A timestamp from an electronic signature is often a more significant piece of evidence than a timestamp from the document itself. This is because the document could have been created significantly before it was signed, and what is important is when it was digitally signed and not when it was created.
- It is possible that one of the parties will want to challenge the validity of a document that has been signed by both parties. After signing the document, it therefore reports the loss of the private key (revokes the certificate). Without the existence of the timestamp of the given document, it is no longer possible to prove whether the document was signed before the relevant certificate was revoked or after it.
- In the case of documents with a relatively long period of validity, the situation is even worse. The electronic signature is verified using the relevant certificate. But it is valid in the interval notBefore after notAfter. If we do not process the electronic signature in any way, the validity of the electronic signature is questionable after the expiration of this interval. The mentioned argument can be used to show that the attacker had the public key available for such a long time that he was technically able to find the corresponding private key. The timestamp from the electronic signature is then proof that the document was created at a time when the attacker did not have the calculated private key available, so the document signature is genuine.

## 7.1 What is time?

A timestamp contains a time value—but what exactly does that mean, and how accurate is it? The question of whether a Timestamp Authority (TSA) can prove the reliability of the

time it provides in its timestamps should not be overlooked.

Time is an abstract quantity that determines the order of events within a particular time frame. It is measured based on periodically repeating natural phenomena, i.e., oscillators. An oscillator is a generator that produces a precise frequency within a defined tolerance. Common examples include astronomical oscillators based on celestial motion in the Solar System (such as Earth's rotation on its axis, its orbit around the Sun, or the Moon's orbit around Earth). Other types of oscillators include pendulums, quartz crystals, or atomic resonances. A clock consists of such an oscillator paired with a counter that counts the number of cycles—for example, one complete rotation of Earth defines a day, and one swing of a pendulum might last 1.3 seconds.

We express time using calendar dates, hours, minutes, seconds, and fractions of a second. Because we derive time from the Earth's rotation, we must also specify a time zone.

Historically, both the calendar and the definition of the second have undergone changes.

The historical time system known as **Greenwich Mean Time (GMT)** is based on the **mean solar time** at the Greenwich meridian (0° longitude). Originally developed for navigation and astronomy, GMT was not an official standard.

### 7.1.1 Length of the Day and the Second

Time is defined in terms of intervals, such as the **mean tropical year**, the time it takes Earth to complete one revolution around the Sun relative to the vernal equinox. This value is determined from long-term astronomical observations of the Sun, Moon, and planets.

In 1956, the **ephemeris second** was defined as 1 / 31,556,925.9747 of the mean tropical year at 0h UT on January 1, 1900. This corresponded to a tropical year of approximately 365.2421987 days. However, because celestial mechanics are affected by various forces (e.g., tidal effects), the tropical year's length slowly increases—by about 5.3 ms per year—with an accuracy of about ±50 ms.

**Universal Time (UT)** was originally based on the mean solar day rather than the tropical year. Nevertheless, both UT and the tropical year are derived from Earth's motion—albeit on different scales (daily vs. annual).

Earth and other celestial bodies can be thought of as massive gyroscopes with relatively stable rotations, but these rotations are not perfectly uniform. Internal and external influences cause irregularities in Earth's rotation, leading to fluctuations in UT.

In contrast, modern electronics use atomic oscillators, which offer far greater stability and precision. In 1967, the definition of the second was changed to be based on atomic time: "the duration of 9,192,631,770 periods of the radiation corresponding to the transition between the two hyperfine levels of the ground state of $^{133}$Cs."

Since 1972, the global standard for time and frequency has been **International Atomic Time** (**TAI**, from its French name **Temps Atomique International**), which is based on a network of cesium clocks with accuracy better than one microsecond per day. However, solar time does not precisely align with atomic time.

### 7.1.2 Leap Seconds and UTC

A day consists of 24 hours, 1440 minutes, or 86,400 seconds. If the base unit is the second, each day should logically have exactly 86,400 seconds. But astronomical observations do not match this exactly. Thus, astronomical time (UT) must be reconciled with atomic time (TAI).

Precise atomic time is adjusted to better match less precise astronomical time because human biological rhythms still align with the latter. After all, one is expected to drink tea at five, according to one's internal clock.

When the discrepancy approaches 0.7 seconds, a correction is made by inserting or removing a **leap second**. This adjustment is only made on the last day of June or December. When added, the leap second appears as 23:59:60, following 23:59:59. When removed, the second 23:59:59 is skipped. The first leap second was introduced on January 1, 1972. The resulting timescale is called **Universal Time Coordinated** (UTC).

### 7.1.3 Time Zones and Daylight Saving Time

Until now, we've considered time as observed in Greenwich, London. Because time is derived from Earth's rotation, we must specify the observer's location. This leads to the concept of time zones. The Earth's surface is administratively divided into 24 main time zones, each shifted by one hour. Some countries opted for shifts of ¼, ½, or ¾ hours instead of whole hours, resulting in unusual local offsets.

Time zones are expressed relative to the zone in which Greenwich lies. Time increases as you move east (positive offset) and decreases to the west (negative offset).

Greenwich lies on the Prime Meridian, historically defining GMT. Today, the zero-time zone is aligned with UTC (UTC+0), which has largely replaced GMT as the international standard. In this context, time zones can also be expressed using the letter **Z (Zulu time)** to indicate UTC+0. For example, local time in Kathmandu is 5 hours and 45 minutes ahead of UTC.

Daylight saving time (DST) further complicates the time zone issue. A practical rule is to store times in UTC in data structures and convert to local time only for display purposes. For instance, both **certificates and timestamps generally record time in UTC+0**.

### 7.1.4 Computer Time

Time is measured using an oscillator connected to a counter that tracks the number of cycles. Many computer protocols—and the system time of most operating systems—do not internally use human-readable formats (calendar date, time, and zone). Instead, they store time as a continuously increasing counter value. Applications then convert this value to a readable format.

For example, the Network Time Protocol (NTP) [46] uses a 64-bit unsigned integer to represent the number of seconds since 00:00:00 on January 1, 1900. This 64-bit number is split into two 32-bit halves: the first holds whole seconds, and the second holds fractional seconds (as though there were a decimal point in the middle of the 64-bit value).

A 32-bit counter for seconds can represent about 136 years, so this counter will overflow in the year 2036. We will eventually have to deal with the "2036 problem." Converting this

counter to a calendar date is nontrivial—leap seconds must be considered. For example, the first leap second on January 1, 1972, had an NTP counter value of 2,272,060,800.

Unix systems use a 32-bit signed counter, which overflows after 68 years. However, Unix time starts from January 1, 1970, so the overflow will not occur until early 2038. Not long ago (on January 14, 2021, at 08:25:36), Unix time reached its round birthday of 0x60000000 seconds! It is no longer a young lad—it has reached maturity.

Other protocols measure time differently. For example, the ICMP timestamp protocol counts milliseconds since the previous midnight.

### 7.1.5 Time resources

The primary source of time must be an extremely precise oscillator, such as the previously mentioned transition between two states of an atom. Available oscillators are based on transitions in hydrogen, cesium, or rubidium atoms. These oscillators have approximately the following levels of stability:

- Hydrogen – 0.086 nanoseconds per day
- Cesium – 8.6 nanoseconds per day
- Rubidium – 432 nanoseconds per day

In comparison, a quartz crystal oscillator has a fixed stability determined by its design. A temperature-controlled crystal oscillator (also TCXO (*Temperature-Compensated Crystal Oscillator*)) exhibits high stability, typically around 43 milliseconds per day. In contrast, an ordinary, non-temperature-controlled crystal oscillator (often referred to as XO (*Crystal Oscillator*)) has a stability of about 4.3 seconds per day per degree Celsius of temperature change.

An equally important parameter is the price of the time resource. It is quite understandable that a national laboratory dealing with time will pay for the best quality time sources, whereas a regular non-temperature-controlled crystal will be enough for us on a PC. The Timestamping Authority (TSA) must be provided with an internal time source having an accuracy consistent with its timestamping policy. Business TSAs typically use rubidium oscillators.

### 7.1.6 Time Providers

One cannot use only two sources of time. If one of these sources breaks up, it will be impossible to find out which one is functioning well and which one is not.

There is a whole network of time measurement laboratories around the world. These laboratories coordinate their time with each other. The resulting time is therefore an “average” value, which includes the times of individual laboratories. However, not every laboratory has the same weight for this time calculation. Labs with more stable times are given more weight than labs whose time has deviated more from the set time in the past.

We now know that there is a worldwide network of laboratories equipped with high quality oscillators, which are also coordinated worldwide. These labs have the exact time available from the perspective of us – the users. Now it remains to solve the problem of how to deliver

the exact time to us.

Many time labs donate their time to the public. Time can be provided in a variety of ways, such as:

- Through regular radio/television broadcasts. This information is intended for human use. While looking at the clock displayed on the TV screen, one can adjust his pocket watch or other clock.
- It is broadcast at special radio frequencies. Information from these frequencies is directly processed by the computer. We used the DCF77 transmitter from Mainflingen, Germany. The radio receiver can be directly connected to the computer and the computer's time source can be directly coordinated according to the radio signal. However, TSA needs a more stable time than an alarm clock or station clock. Even so, this time can be used to check otherwise provided time.
- Satellite navigation systems (GPS, Galileo, GLONASS or BeiDou) also provide time. Satellite time is directly intended for computer processing; it is significantly more accurate than radio time. Satellite time is gradually displacing the use of radio time when synchronizing computer time. Today, there are 1U rackmount boxes or PCI-Express cards (e.g., from Meinberg) for servers that can process the time provided by all four satellite systems at the same time and signal the event of a time deviation.
- Over a computer network by application protocols (e.g., NTP [46]).
- National Time Standard, e.g., in Czech Republic is the standard maintained by the Institute of Photonics and Electronics of the Academy of Sciences.

## 7.2 TSA

Figure 7.2 shows a typical Timestamp Authority (TSA) architecture. The client uses the transport protocol to access the TSA FrontEnd, which forwards the request to a specific **Timestamp Unit (TSU)** for processing. The TSA itself usually consists of several TSU units. TSU is a set of hardware and software designed for issuing and especially signing timestamps under a specific TSA policy. Each TSU has its own key pair; the private keys are usually stored in the Hardware Security Module (HSM).

The client requests the issuance of a timestamp using a data structure called a timestamp request (Figure 7.3 and Figure 7.5). It sends this request to the TSA FrontEnd. Then the request is forwarded to a specific TSU according to the timestamping policy. Unless there is a formal error, the TSU returns a response that contains the requested timestamp. This communication specifies Timestamp Protocol (TSP) [47] which also defines the format of the timestamp and timestamp request.

The TSP is not concerned with the specification of data transfer between the client and the TSA. So, the timestamp request can also be stored on a flash drive and taken to TSA. In the case of network communication, the TSP embeds both the request and the response in the corresponding transport protocol (e.g., HTTP).

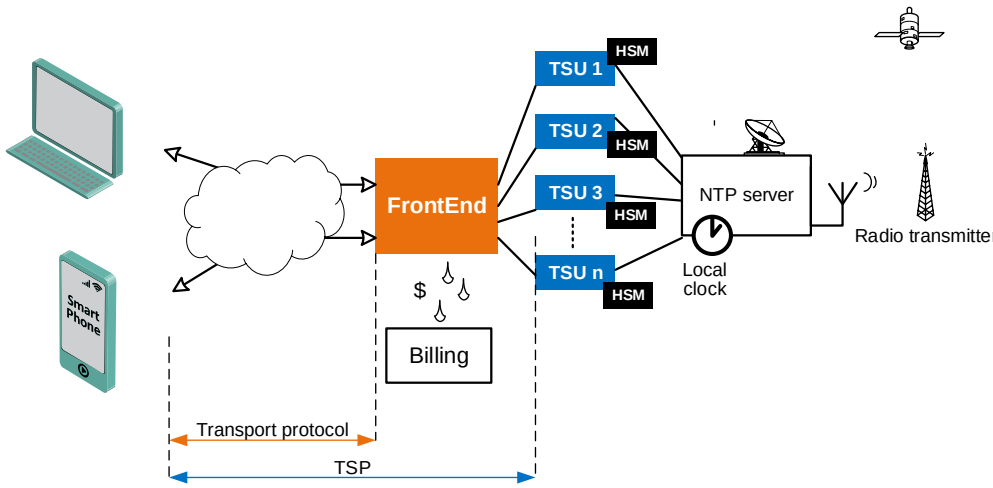



Figure 7.2 *TSA architecture*

**11 commandments of RFC 3161**

- Use a trustworthy source of time.
- Include a trustworthy time value for each time-stamp token.
- Include a unique integer for each newly generated time-stamp token.
- Produce a time-stamp token upon receiving a valid request from the requester, as soon as possible.
- Include within each time-stamp token an identifier to uniquely indicate the security policy under which the token was created.
- Only time-stamp a hash representation of the data, i.e., a data imprint associated with a one-way collision resistant hash-function uniquely identified by an OID.
- Examine the OID of the one-way collision resistant hash-function and verify that the hash value length is consistent with the hash algorithm.
- Do not examine the imprint being time-stamped in any way (other than to check its length, as specified in the previous bullet).
- Do not include any identification of the requesting entity in the time-stamp tokens.
- Sign each time-stamp token using a key generated exclusively for this purpose and have this property of the key indicated on the corresponding certificate.
- If asked by the requester using the extensions field, include additional information in the time-stamp token only for the extensions that are supported by the TSA. If this is not possible, you SHALL respond with an error message.

So far, we have presented the TSA as a crazy clerk who just mechanically timestamps anything he can get his hands on. However, this is only in theory, in practice TSA must live (must earn money). So, in practice, the TSA will only issue timestamps to paying clients, i.e., the client will have to authenticate to the TSA in some way, for example, with a personal certificate in TLS communication (i.e., the TSP is inserted into the HTTPS channel).

It is very important that TSA uses a stable time source. Time management uses time from trusted time providers for this purpose. It is advisable to take time from at least three sources (voting is already possible in the case of three sources).

TSA is a trusted third party. The TSA operator creates one or more documents called a timestamping policy (TSA policy). Each policy is then assigned a unique object identifier (OID). The identifier of the TSA policy object, under which the timestamp was issued, is always inserted into the timestamp. (This is different from a certificate, for which the Certificate Policy extension is optional, and may also contain multiple policies!)

TSA must sign the timestamps with an exclusive dedicated key for this purpose. TSA can have multiple key pairs (one for each TSU), e.g., for different policies, different algorithms, different key lengths, etc. The TSA public key is stored in the TSA certificate, which must have exactly one instance of the "Extended Key Usage" extension (value 1.3.6.1.5.5.7.48.3), which is marked as critical.

A very important part of the TSA is its documentation, which in many ways is similar to the documentation of the certification authority. Before you start building a TSA, and especially before you start writing TSA policies, read RFC 3628 [48]: Policy Requirements for Time-Stamping Authorities (TSAs ).

### 7.2.1 Timestamp Protocol (TSP)

The Time-Stamp Protocol (TSP) is specified by RFC 3161 [47]. The TSP describes a timestamp request and a response.

Communication is simple: the client sends a request to which TSU returns a response. The response either contains the issued timestamp or a specification of the error. After encapsulating the TSU into the transport protocol (e.g., HTTP), a client-server protocol is created.

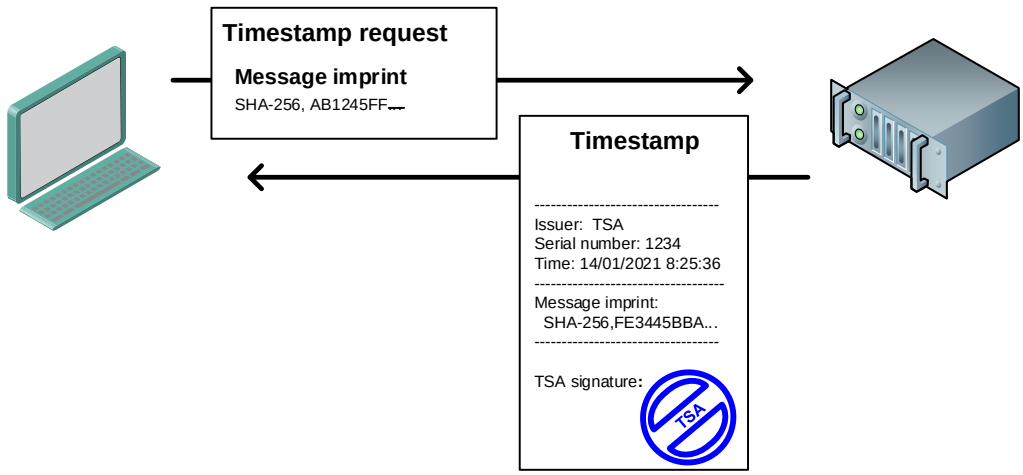



Figure 7.3 *Format of timestamp request and timestamp*

Importantly, the timestamp request is not digitally signed, i.e., it is anonymous. TSU does not investigate the identity of the applicant. TSU will issue a timestamp whenever the timestamp application is formally correct. In contrast, a timestamp is a digitally signed structure that contains, among other things, a TSA identification.

### 7.2.2 Transport protocols

If the request is not stored on a data carrier and is not personally delivered to the TSA, then the TSP communication is inserted into the transport protocols that ensure data transfer between the client and the TSA (i.e., the server).

As transport protocols, it is possible to use TCP protocols directly or, theoretically, UDP as well. The well-known ports 318/tcp and 318/udp are reserved for TSA for these purposes. However, it is more common to embed this communication in the HTTP protocol.

### 7.2.3 Timestamp request

A timestamp request is a sequence of items. These items are explained below:

- The **Version** item contains the version of the TSP protocol. The current version is 1.
- The **Message imprint** item contains the imprint of the document being stamped and the algorithm by which it was created.
- The optional **Policy** item may contain the object identifier of the policy under which the client wishes to issue a timestamp.
- The optional **Nonce** field contains a random number that TSA copies into the response.
- If Certificate requesting is set to TRUE, the requester wants TSA to return the timestamp along with a TSA Certificate in its response.

### 7.2.4 Timestamp

A timestamp is an electronically signed data structure (CMS format) consisting of:

- The **Version** item that contains the version of the TSP protocol. The current version is 1.
- The Policy item which contains the object identifier of the TSA policy under which the timestamp has been issued and can be used.
- **Message imprint** item that contains the imprint of the stamped document and the algorithm by which it has been created (copied from the request).
- The **Serial number** item contains the serial number of the issued timestamp. Logically we would assume that the serial numbers of the issued stamps would increase after each issued stamp for example by 1. But this is not necessary the case, no regulation specifies how to assign serial numbers – it only says that within a particular TSA the serial numbers of the timestamps issued must be unique. Because of this, unfortunately, the serial numbers of the timestamps might not tell us about the order in which they were issued. However, TSA policy specifies the method of assigning sequence numbers to a particular TSA.
- The **Time** item contains the time the stamp was issued. UTC time is used and local time zones are not used, i.e., the time ends with the letter ”Z” used for GMT.
- The **Accuracy** item expresses the time accuracy. The maximum deviation can be spec-

ified in seconds, milliseconds, and microseconds. If an item is not listed, it is assumed to have a zero value.

- The **Ordering** item specifies whether the time shown on the timestamps is so accurate that the issued timestamps can be sorted in the order they were issued. If this field is present in the timestamp and has the value TRUE, then the timestamps can be sorted by the time they were issued as specified in them.
- The optional **Nonce** item is filled with the content of the item of the same name from the timestamp request (if it was filled). This item is then used to match the issued timestamp with the corresponding timestamp request.
- The **TSA Name** optional field may contain the name of the TSA authority. Surprisingly, this item is optional because the TSA name can be taken from the subject of the certificate that verifies the timestamp electronic signature.
- The **Extensions** item is intended for additional information that may be added to the timestamp in the future.

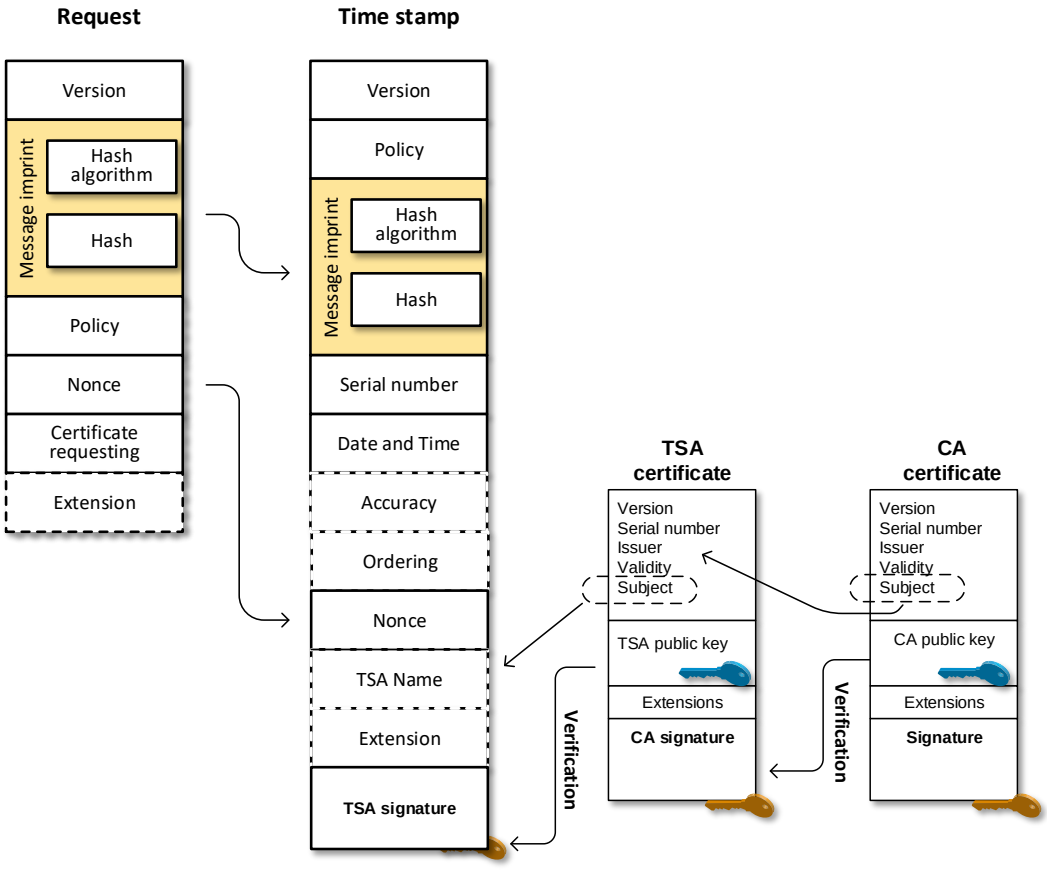



Figure 7.4 *Time stamp request and Time stamp*

### 7.2.5 Timestamp verification

A timestamp is a data structure digitally signed by the TSA. So, we are going to verify that signature with a TSA certificate. However, the TSA certificate was issued by a trusted certification authority (Figure 7.5). So, we need to create a Certificate chain for verification.

Let us imagine that we have already validated the TSA certificate in the certificate chain up to the trusted anchor. And this with the result that the TSA certificate is valid. Then we can verify the electronic signature of the timestamp using the certificate verified in this way. If the timestamp electronic signature is valid, we need to verify the document's imprint, i.e., we need to compute the hash from the original message and verify that it matches the hash given in the timestamp.

Now that we have the cryptographic gymnastics out of the way, let us find out if the TSA is trustworthy for us. It should convince us of its credibility with the policy indicated in the timestamp ...

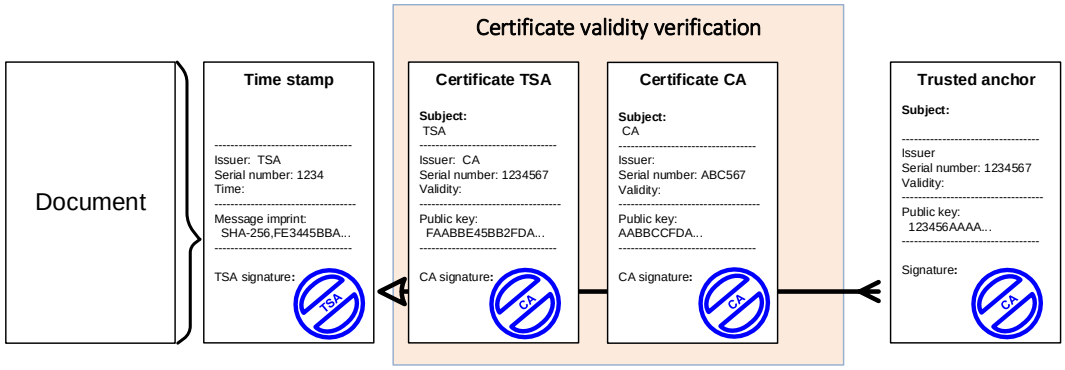



Figure 7.5 *Timestamp verification*

### 7.2.6 Timestamp validity

From a certification authority perspective, TSA is a special case of an end user. A CA issues a TSA certificate with a few differences from certificates for regular end users.

The TSA certificate differs from other end-user certificates not only in that it is filled with extensions so that it might not be used for anything other than timestamp verification, but in particular in that it has a different period for which it is valid. A TSA certificate usually has a validity period similar to that of the CA that issued it.

A timestamp is a digitally signed structure. The validity of this structure is thus limited by the validity of the certificate with which we verify its electronic signature, i.e., the validity of the TSA certificate. If this were not the case, then we would have difficulty verifying timestamps from electronic signatures.

Let us imagine that we are verifying an electronic signature provided with a timestamp from the electronic signature (Figure 7.5). Then we have to:

- Verify the timestamp of the electronic signature using the appropriate TSA certificate.
- Verify the electronic signature using the appropriate certificate of the end user who

created the signature.

If the validity period of the end-user certificate overlaps with the validity period of the TSA certificate, for example, by only 1 day, then the electronic signature of the document will have to be renewed after only one day. So, the effort is to make the validity period of the TSA certificate longer than the validity of the certificates of ordinary end users. We have already encountered a similar problem when renewing certificates of certification authorities, which must be renewed well in advance to avoid a situation where the end-user's certificate is still valid when the validity of the CA certificate that issued it has already expired.

Similarly, in the case of TSA, we will issue a new TSA certificate in advance so that:

- At the time when the validity periods of the old and new TSA certificates overlap, it is technically possible to restore all timestamps created by the old TSA with the new TSA.
- All timestamps of electronic document signatures created using the old TSA certificate are valid for at least the validity period of all user certificates intended for the verification of electronic signatures of documents.

**Conclusion:**

- The time stamp of a valid electronic signature is proof that the signature existed at the time specified in the time stamp.
- This proof is valid for the duration of the timestamp.
- The time stamp of the electronic signature therefore ”extends” the validity of the electronic signature.

### 7.2.7 What is not a timestamp?

The timestamp (in RFC 3161 format) is not sufficient by itself. It proves that a print from a document existed at a given time. From this it can be deduced that the document must have existed before the time indicated in the timestamp. However, the timestamp does not contain the identification of the holder of the document, so the timestamp cannot serve as proof that a document was owned by a specific person at a given time.

The basic problem of the timestamp is also its validity period. At first glance, it seems that the timestamp is permanently valid. However, this is not the case. A timestamp is a digitally signed structure that is signed by the TSA. This electronic signature is verified using a TSA certificate and it has a limited period of validity. Therefore, the validity period of the timestamp is indirectly determined by the validity period of the TSA certificate.

# 8 HTTP

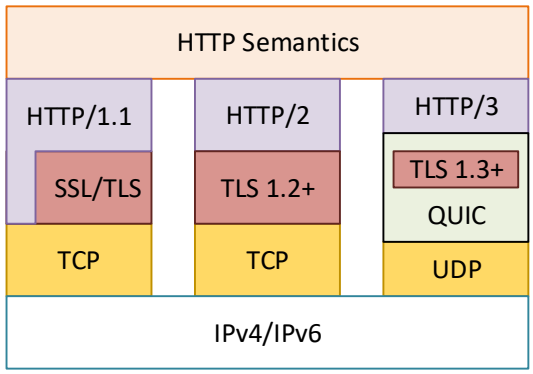



Figure 8.1 *HTTP model*

Hypertext Transfer Protocol (HTTP) was created around 1990 by Sir Timothy John Berners-Lee's team at CERN (European Organization for Nuclear Research). Version 0.9 was published in 1991. TCP was used as the transport protocol, which was also adopted in later versions HTTP/1.0, HTTP/1.1 and HTTP/2.0.

We currently have three versions of the HTTP protocol: HTTP/1.1, specified by the RFC 9112 [49] standard; HTTP/2, specified by RFC 9113 [50]; and HTTP/3, specified by RFC 9114 [51]. The HTTP/2 protocol is also referred to as *h2* , and HTTP/3 as *h3*.

Table 8.1 *HTTP Terminology*

| | |
|---|---|
| **User Agent** | Any of the various client programs that initiate a request. |
| **Origin Server** | A program that can originate authoritative responses for a given origin representation. In this text, Origin Server is often abbreviated to Server. |
| **Cache** | Local storage of the previous messages and the subsystem that manages the storage, retrieval and deletion of these messages. |
| **Intermediary** | HTTP allows the use of intermediary to satisfy requests through a chain of connections. There are three forms of intermediary: proxy, gateway, and tunnel. |
| **Representation** or **Entity representation** | Data (entity) representation on the Origin server or Intermediary. |

At first glance at the HTTP model (Figure 8.1) we can see:

- All 3 variants of the HTTP protocol use common HTTP semantics [52].
- Not using TLS is only allowed by HTTP/1.1.
- HTTP/3 does not run on top of TCP, but on top of QUIC/UDP.

HTTP is a stateless request/response protocol for exchanging ”messages” over ”connections”. The terms ”client” and ”server” refer to any implementation that sends or receives a given message. The client sends requests to the server in the form of a ”request” with the method and the identified origin representation.

## 8.1 Messages and frames

HTTP semantics works at the message level. While the HTTP/1.1 protocol inserts messages into the TCP connection, which can be secured by the TLS protocol (or its predecessor SSL), the HTTP/2 protocol defines the syntax of its own frames into which the sent messages are cut. This allows multiple messages to be transported concurrently without having to establish a new TCP connection.

The HTTP/3 protocol uses QUIC protocol packets (including their flow control), which also uses its own frames.

### 8.1.1 HTTP/1.1

HTTP/1.1 (Figure 8.2) either allows individual messages to be sent sequentially through the same TCP connection (Connection: Keep-Alive header field), or messages can be sent in parallel, but in this case a new TCP connection and an SSL/TLS connection are established for each message. HTTP/1.1 does not use streams.

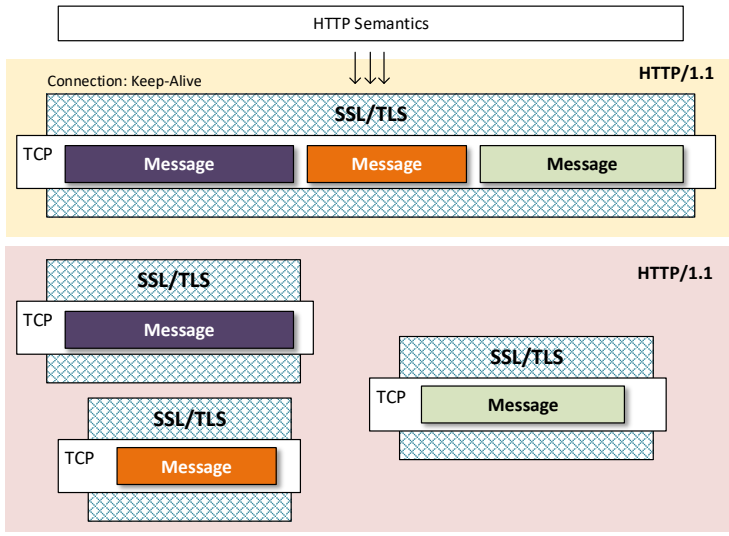



Figure 8.2 *HTTP/1.1*

### 8.1.2 HTTP/2

A web page usually consists of several objects. Some of them may be on the same server, others on 3rd party servers. Objects on the same server can be downloaded in parallel using HTTP/2 (Figure 8.3). In terms of the HTTP protocol, the object is downloaded using the HTTP ”request-response” message dialog. Individual dialogs are transmitted in separate streams.

So that the transmission of long messages does not slow down other transmissions, HTTP messages are divided into smaller units – HTTP/2 frames. Streams are transmitted in frames, with TCP mixing the frames of individual streams.

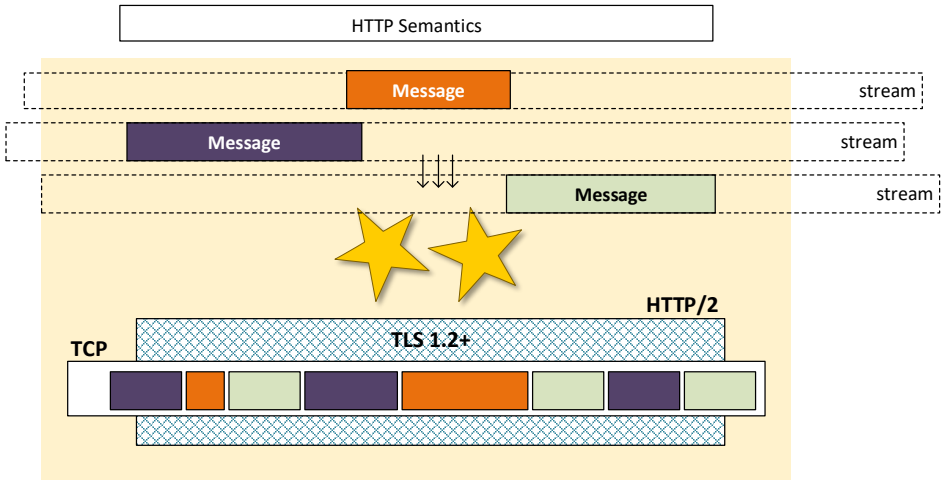



Figure 8.3 *HTTP/2*

When the connection is established, stream 0 (control stream) is created, in which both parties agree on the size of the compression tables and other common parameters of the connection (see SETTINGS frame in Table 8.1). Next streams transmit application messages. Streams initiated by the client have odd numbers, streams initiated by the server (e.g., PUSH method) have even numbers. The stream number is shared between the client and the server, meaning that if the client sends a request in stream 1, then the server also responds in stream 1.

The connection is always secured by TLS 1.2 or higher.

The beginning of client communication in the HTTP/2 protocol is special – as if the PRI method was used. It is not a method, but a so-called magic string, with which the client starts HTTP/2 communication (in the control stream). If an HTTP/1.1 client contacted an HTTP/2 server by mistake, it would encounter a non-existent method. This magic string has the form:

```
PRI * HTTP/2.0\r\n\r\nSM\r\n\r\n
```

The HTTP/2 protocol:

- Connection parameters are set in stream 0, usually within frames WINDOW_UPDATE and SETTING.
- The client continues in stream 1 with a request in the HEADERS frame, or sends additional HEADERS frames in other odd streams.
- The server responds in individual streams.

The HTTP/2 protocol:

- Connection parameters are set in stream 0, usually within SETTING and WINDOW_UPDATE frames.

Table 8.2 *HTTP/2 frames*

| | |
|---|---|
| DATA | Transfers application data – conveys arbitrary, variable-length sequences of octets associated with a stream. |
| HEADERS | Transfers both header fields and footer header fields. HPACK header compression is used [53]. |
| RST_STREAM | Terminates the stream immediately. |
| SETTINGS | `SETTINGS_HEADER_TABLE_SIZE` – informs the remote endpoint of the maximum size of the compression table used to decode field blocks, in units of octets.<br><br>`SETTINGS_ENABLE_PUSH` – enable/disable server PUSH.<br><br>`SETTINGS_MAX_CONCURRENT_STREAMS` – indicates the maximum number of concurrent streams.<br><br>`SETTINGS_INITIAL_WINDOW_SIZE` – indicates the initial size of the sender's window (i.e., how many bytes it is able to receive).<br><br>`SETTINGS_MAX_FRAME_SIZE` – indicates the size of the largest frame payload that the sender is willing to receive, in units of octets.<br><br>`SETTINGS_MAX_HEADER_LIST_SIZE` – informs about the maximum size of the area for receiving uncompressed header fields. |
| PUSH_PROMISE | Notify the peer endpoint in advance of streams the sender intends to initiate PUSH. |
| PING | A mechanism for measuring a minimal round-trip time from the sender, as well as determining whether an idle connection is still functional. PING frames can be sent from any endpoint. |
| GOAWAY | Initiate shutdown of a connection or to signal serious error conditions. GOAWAY allows an endpoint to gracefully stop accepting new streams while still finishing processing of previously established streams. |
| WINDOW_UPDATE | It controls the flow of data using a window size that determines how many bytes a page is able to accept. |
| CONTINUATION | If the content does not fit into the HEADER or PUSH_PROMISE frames, then the CONTINUATION frame is continued. |

- The client continues in stream 1 with a request in the HEADERS frame, or sends additional HEADERS frames in other odd streams.
- The server responds in individual streams.

HTTP/2 introduces several types of frames, some of which address stream management (Table 8.2).

### 8.1.3 HTTP/3

The disadvantage of HTTP/2 is that if one message is not transmitted (Figure 8.4), then the entire connection will fail. HTTP/3 solves this by not using the TCP protocol, but establishing the connection using QUIC protocols over the UDP protocol.

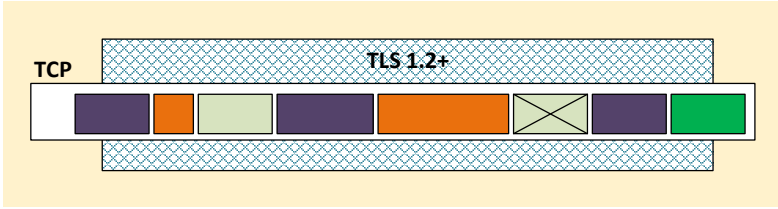



Figure 8.4 *HTTP/2 frame failure*

A QUIC stream provides reliable in-order delivery of bytes, but makes no guarantees about the order of delivery with regard to bytes on other streams. The stream data containing HTTP frames is carried by QUIC STREAM frames, but this framing is invisible to the HTTP framing layer.

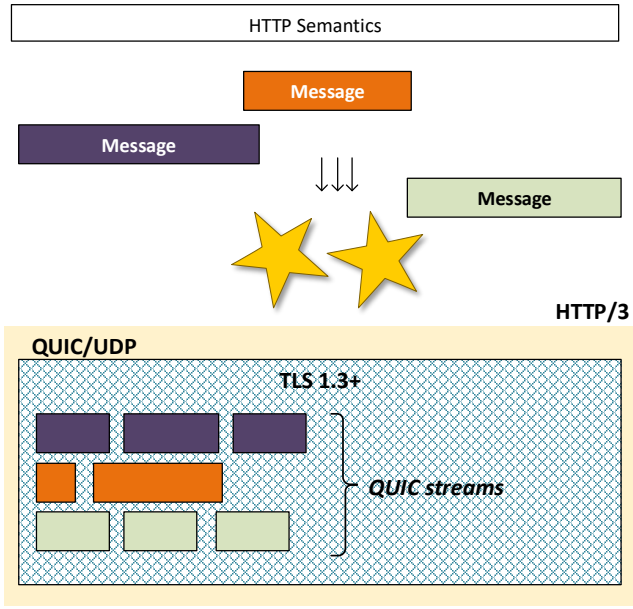



Figure 8.5 *HTTP/3*

HTTP/3 uses two types of QUIC protocol streams: Bidirectional and Unidirectional. Bidirectional streams (referred to as Request streams) are used for HTTP requests and responses. A bidirectional stream ensures that the response can be readily correlated with the request.

Unidirectional streams are:

- Control Stream – each party at the beginning of the connection must initiate a control stream (unidirectional) that transmits SETTINGS type frames, which are used to set the communication parameters.

- Server Push is an optional feature introduced in HTTP/2 that allows a server to initiate a response before a request has been made.

Table 8.3 *HTTP/3 frames*

| | |
|---|---|
| DATA | Transfers application data. |
| HEADERS | Transfers both header fields and footer header fields. The header fields are compressed by QPACK [54]. |
| CANCEL_PUSH | A request to cancel a PUSH server stream. |
| SETTINGS | Connection settings – can only be used in the control stream. Examples of settings:<br>`SETTINGS_QPACK_MAX_TABLE_CAPACITY (0x01)` – maximum size of dynamic compression table<br>`SETTINGS_MAX_FIELD_SECTION_SIZE (0x06)` – the maximum size of the message header<br>`SETTINGS_ENABLE_CONNECT_PROTOCOL (0x08)` – allows accepting other protocols (methods) in URIs than HTTPS using a new pseudo-header field `:protocol:` [54] |
| PUSH_PROMISE | Contains the PUSH server message header. |
| GOAWAY | Initializes connection termination or error. GOAWAY allows you to stop accepting new streams while still completing processing of previously established streams. |
| MAX_PUSH_ID | The client sets how many PUSHs the server can initiate. |

**Note:** When monitoring HTTP/3 with Wireshark, viewing HTTP/3 protocol frames is prevented by QPACK encoding. It is easier to display the header fields, for example, in the browser using the option: F12 - Developer Tools - Network.

## 8.2 Message format

The message format of the HTTP/1.1 protocol is based on the e-mail message format (officially Internet Message Format defined by RFC 5322 [55]). An HTTP/1.1 message begins with a header called Control data. Control data consist of the method (in case of request) or status code (in case of a response) followed by header fields. Header fields are made up of a case-insensitive field name terminated by a colon, followed by a content of field. The header fields are encoded exclusively in US-ASCII.

This concept was reworked in HTTP/2 and the rework was taken over in HTTP/3. In these protocols, the method (or status code) is converted into the format of header fields, sometimes referred to as pseudo-header fields. In pseudo-header fields, the header field name not only ends with a colon, but also begins with a colon.

HTTP semantics therefore introduces a so-called message abstraction that specifies ”message characteristics common to all HTTP variants”. Imprecisely, but intelligibly, to say: it describes messages textually, as we know it from the HTTP/1.1 protocol, but with certain modifications.

An HTTP message consists of:

- Control data:
  - In the case of the client, it contains the method and the name of the server.
  - In the case of the server, it contains a status code.
- A header fields lookup table that contains individual header fields. The lookup table is because the effort is not to always transmit the same header fields, but if possible each header field only once (within the connection). This mechanism is called header compression.
- Stream of content.
- The Trailer, which again carries the header fields provides a list of field names that the sender anticipates sending as trailer fields within that message. This allows a recipient to prepare for receipt of the indicated metadata before it starts processing the content.

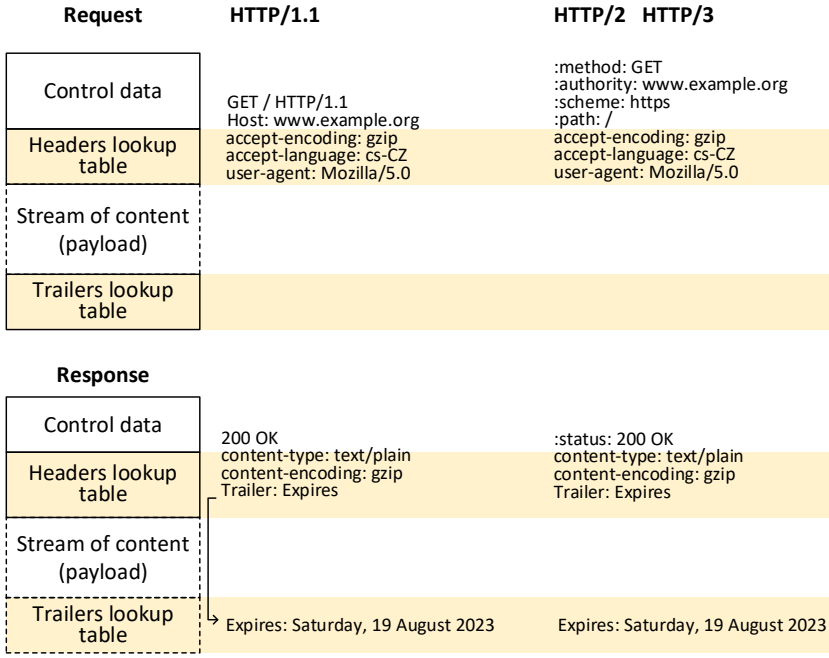



Figure 8.6 *HTTP request and response with examples in individual variants of the HTTP protocol*

## 8.3 An example of communication

**Example:** The user entered a request in the browser:

```
https://www.example.org
```

**The browser communicates via the HTTP/2 protocol:**

1. In stream 0, client sends a magic string, a SETTINGS frame, and a WINDOW_UPDATE frame.
2. In stream 1, client sends an HTTP/2 HEADERS frame (abbreviated):

```
:authority: www.example.org
:method: GET
:path: /
:scheme: https
Accept: text/html
Accept-Encoding: gzip, deflate, br
Accept-Language: cs,cs-CZ;q=0.9,en;q=0.8
Cache-Control: max-age=0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
```

The server responds:

1. In stream 0, server sends WINDOW_UPDATE and SETTINGS frames.
2. In stream 1, server sends HEADERS frame returns:

```
:status: 200 OK
Age: 14
Cache-Control: max-age=30
Content-Encoding: gzip
Content-Type: text/html; charset=utf-8
Date: Thu, 24 Aug 2023 13:17:01 GMT
Expires: Thu, 24 Aug 2023 13:17:17 GMT
Server: cloudflare
Strict-Transport-Security: max-age=3600
Vary: Accept-Encoding
```

3. In stream 1, the DATA frame containing the data requested by the client continues.

**The browser communicates via the HTTP/1.1 protocol:**

```
Get / HTTP/1.1
Host: www.bbc.com
Accept: text/html
Accept-Encoding: gzip, deflate, br
Accept-Language: cs,cs-CZ;q=0.9,en;q=0.8
Cache-Control: max-age=0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
```

Server:

```
Status 200 OK
Age: 14
Cache-Control: max-age=30
Content-Encoding: gzip
Content-Type: text/html; charset=utf-8
Date: Thu, 24 Aug 2023 13:17:01 GMT
Expires: Thu, 24 Aug 2023 13:17:17 GMT
Server: cloudflare
Strict-Transport-Security: max-age=3600
```

```
Vary: Accept-Encoding
```

## 8.4 Headers compression

Header fields can be both after the control data and in the trailer of the message. The header fields make up lookup table. The question is why the term lookup table is introduced and not just header and trailer. When analyzing HTTP/1.1 traffic, it is easy to see that header fields make up a significant portion of the data being transmitted. Additionally, the same header fields are often transmitted.

The HTTP/2 protocol introduced header compression, where both sides of the connection maintain a header field lookup table, with the fact that the header field can be replaced by a link to it. The tables are valid for the entire connection, they are separate for both directions, and their content is always decided by the sender. Tables also contain pseudo-header fields. The size of the tables is set in stream 0.

The sender can choose one of the following variants for each header field in their messages:

- Sends a link into the header field lookup table if the entire header field is present in the lookup table.
- Refers the header field name in the lookup table and appends its own content to it.
- Sends the entire header field.

In the second and third cases, the sender also decides whether the recipient should add the header field to the table or not, eventually prohibits its addition to the tables for any intermediaries.

## 8.5 HTTP URI

HTTP semantics uses Uniform Resource Identifiers (URI)[56]. It defines http and https URI schemes:

```
http-URI = "http://" authority path [ "?" query ]
https-URI = "https://" authority path [ "?" query ]
```

Finally, a # character can be added followed by the fragment identifier of the web page. But that is not part of the HTTP protocol specification. The URI entries can mean the following:

- `authority` that specifies the Origin server (DNS name or IP address + possibly port). It contains the identification of the server (DNS name or IP address) and may contain a user name, but may not contain a password.
- `path` and optional query identify the origin representation within that origin server's namespace.

URIs can only contain US-ASCII code characters. If it is necessary to enter another character, this character is introduced with the % character, followed by the hexadecimal representation of the character. In the hexadecimal representation of the character, we can write both uppercase and lowercase letters for the hexadecimal digits from A to F. The characters ";",

”/”, ”?”, ”:”, ”@”, ”=” and ”&” are reserved for special use, i.e. if they are to be used in a string, they must be converted to hexadecimal and introduced after a percentage.

Examples:

```
http://example.org:80/~libor/home.html
http://EXAMPLE.ORG/%7Elibor/home.html
https://example.org:/%7elibor/home.html
```

If the URI does not begin with a scheme name, it is a relative URI. A relative URI is always relative to some base – some absolute URI. The base can be the URI of the displayed document or, possibly, of the previous document, i.e., the URI of the document from which the link was made. In the event that no such URI exists, the default URI of the entire application may be used.

A relative URI can even contain the “.” and the “ .. ” characters to refer to the current and parent directory, respectively. When processing a relative URI, the URI is viewed as several separate parts: the origin server name, the path, the query, and the fragment. It is assumed that the fragment starts with #, the query with “?”, the path with a slash, and the server with two slashes.

So, for example, on the page `http://www.example.org/path/file.htm` there can be, for example, relative links:

- `#paragraph1`, which can be converted to an absolute URI:
  `http://www.example.org/path/file.htm#paragraph1`
- `file2.htm`, which can be converted to an absolute URI:
  `http://www.example.org/path/file2.htm`
- `../file3.htm`, which can be converted to an absolute URI:
  `http://www.example.org/file3.htm`

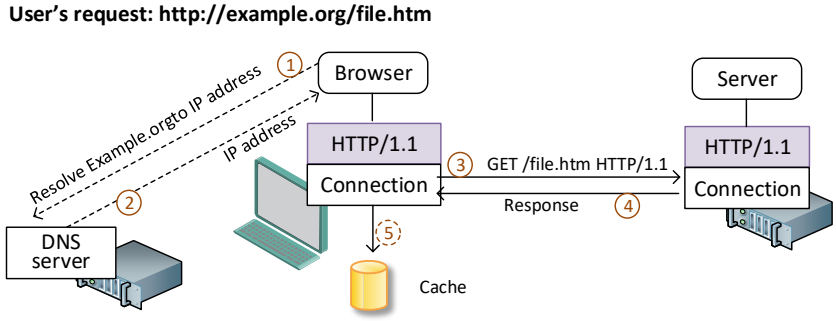



Figure 8.7 *HTTP without intermediaries*

## 8.6 Intermediaries

The principle of intermediaries will be clarified on the HTTP/1.1 protocol. First, we describe the client request process without any intermediaries (Figure 8.7):

1. The Origin server name is extracted from the user’s requested URI and sent to DNS

for resolution.

2. DNS returns the IPv4/IPv6 address of the Origin server.
3. The browser establishes a connection with the origin server and sends it a request (relative URI) using the GET method.
4. The origin server returns a response.
5. The response can also be stored in the local cache.

### 8.6.1 Proxy

In the case of a proxy, the client (browser) knows the DNS name of the server side of the proxy, and it is set in its configuration.

Proxy is a system consisting of two parts:

- The **server side**, which accepts the client's requests as if they were being received by the origin server. However, it immediately passes the requests to the client side.
- The **client side**, which receives requests from the server side, establishes a TCP connection with the origin server and forwards the requests to the origin server for processing on behalf of the client.

In the proxy between the server and client side, the Proxy's own logic is hidden from the user. The proxy understands the application protocol (in our case the HTTP protocol) and can perform several operations with the received request from the client:

1. It can overwrite the request (or response), i.e., change the data of the application protocol.
2. It can store responses in the Cache memory. If the proxy receives the same request in the future (e.g., from another client), it can return this request more quickly directly from cache without establishing a connection to the origin server. Thanks to the use of dynamic pages, and last but not least, the increase in the throughput of communication lines, the importance of the Cache memory on the Proxy begins to recede.
3. It can determine whether the client is authorized to make such a request, i.e., perform Proxy-authentication.
4. It can detect whether the client is accessing any unwanted server, for example, the employer can set up a list of servers on the proxy that his employees will not be able to access.
5. Proxies running on a firewall can check which network interface a user's proxy request is coming from, i.e., whether the user accesses from the network interface of the internal network or from the network interface to the Internet.
6. Proxy can scan data for viruses before to pass on (and caching) it.

Figure 8.8 schematically shows the operation of the proxy. In the beginning, the user enters the URI that he wants to view in the window of his browser, for example, the client enters a request in the browser window:

```
http://example.org/file.htm
```

However, the browser will handle this request through a proxy, i.e., in the configuration of his browser, the user has specified the DNS name of the server side of the proxy through which the request will be handled. The browser then only detects the application protocol from the URI, as the browser can use a different proxy for each application protocol.

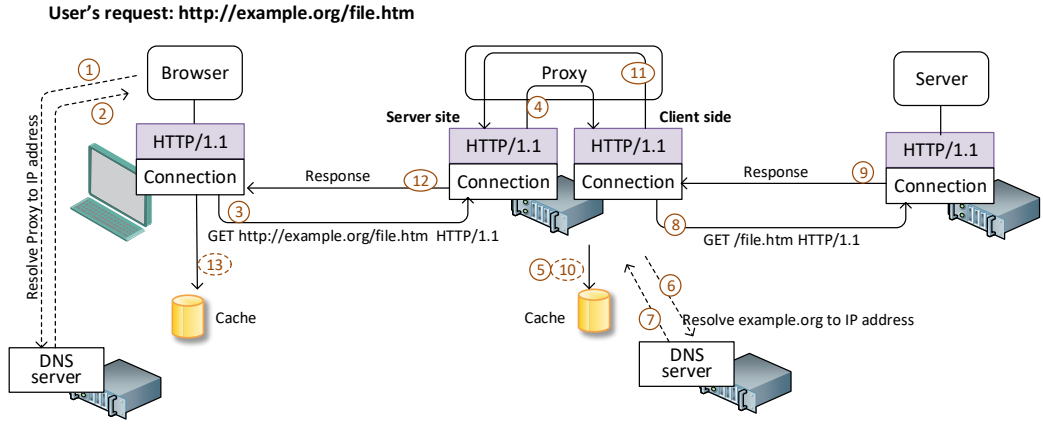



Figure 8.8 *Proxy*

In the first step, the client translates the Proxy name to an IP address (1 and 2). The origin server name may not even be resolved in the user's internal network. Now the client establishes a connection with the server side of the proxy on the port specified in the browser configuration (3). In the TCP connection created in this way, the browser enters the HTTP request (3) containing the absolute URI:

```
GET http://example.org/file.htm HTTP/1.1 Host: example.org
```

The request is received by the server side of the proxy, which is checked by the internal logic of the proxy (4).

The proxy will check its Cache memory if it does not have an answer to this request (5). If answer is found, it will then no longer contact the origin server and return a direct response.

If the answer is not found in the cache, then DNS resolves the name of the origin server in DNS (6 and 7). A connection to the origin server is established and a **relative URI** is entered into it. The origin server then returns a response (9), which is returned to the user's browser through the proxy (9 and 12). The proxy and browser can also put the response in their caches (10 and 13).

Today, a proxy is often used even in the case of communication secured by the TLS protocol (Figure 8.9), which only works if TLS does not include the authentication with the help of the client's certificate – more precisely, with a private key belonging to the public key of the client's certificate. Because the proxy does not have this private key.

The principle is simple because this type of proxy is usually used in the intranet, where the administrator also has control over the configuration of trusted certification authorities. A fake CA is launched on the intranet and automatically issues origin server certificates. Sub-

sequently, this fake CA is configured as trusted within the intranet.

The browser is configured so that all requests to servers outside the intranet are routed to the proxy server side (1), which poses as the origin server. To be able to do this, it issues a fake certificate on a fake certification authority (2 and 3). This certificate contains the DNS name of the origin server in the Subject Alternative Name certificate extension. Furthermore, it already runs similarly as in the case of a proxy without TLS.

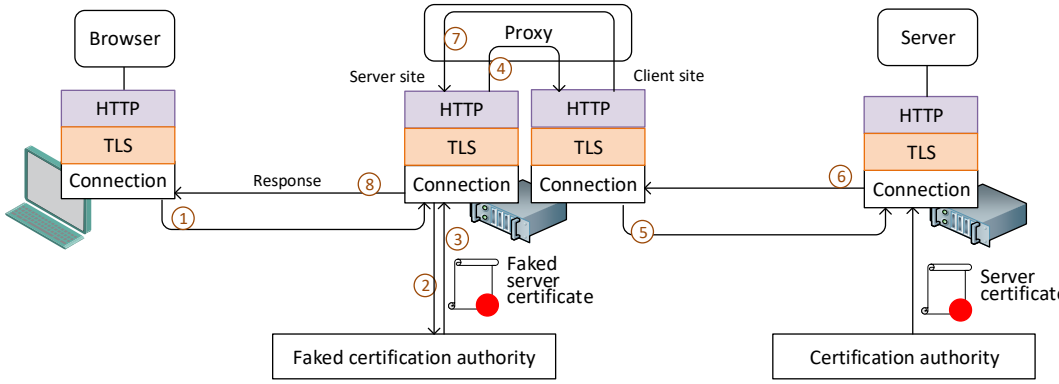



Figure 8.9 *TLS Inspection*

In this way, the proxy gets to the transmitted data. This mechanism is referred to as TLS inspection. TLS inspection can perform the same operations as a proxy, including virus checking and message caching.

Employees often feel outraged by the monitoring of their data, but in this day and age it might be necessary for security reasons. We can easily recognize TLS inspection if we view the server certificate in the browser. If it is issued by the company's CA, then it is a TLS inspection.

### 8.6.2 Gateway

A gateway is an intermediary that looks like an origin server, but converts requests to subsequent servers. The most common gateway implementation is application protocol conversion, for example, HTTP to FTP.

The most common case of application protocol conversion is a gateway, the server side of which receives a request from clients in the HTTP protocol and converts it to communication in the FTP protocol (Figure 8.10).

The interesting feature of this type of gateway is that if the user wishes to display the contents of the directory, the gateway will only obtain the contents of the directory as displayed by FTP, i.e., receives information about individual files in the directory via a data channel. However, this directory content must be returned to the client as a web page, i.e., in HTML language format. So, it must have images of icons for a file, icons for a directory, etc., which it then displays to the client. This means that if you view the contents of the directory through a proxy from one vendor and then through a proxy from another vendor, the graphics may

be different. It is not so surprising, because if the browser establishes a connection directly with the origin server (without a proxy), the contents of the directory obtained by the FTP protocol must similarly be reworked into a graphic form, so that the listing of the same directory displayed by different browsers (or different versions of the same browser) may have other graphic editing.

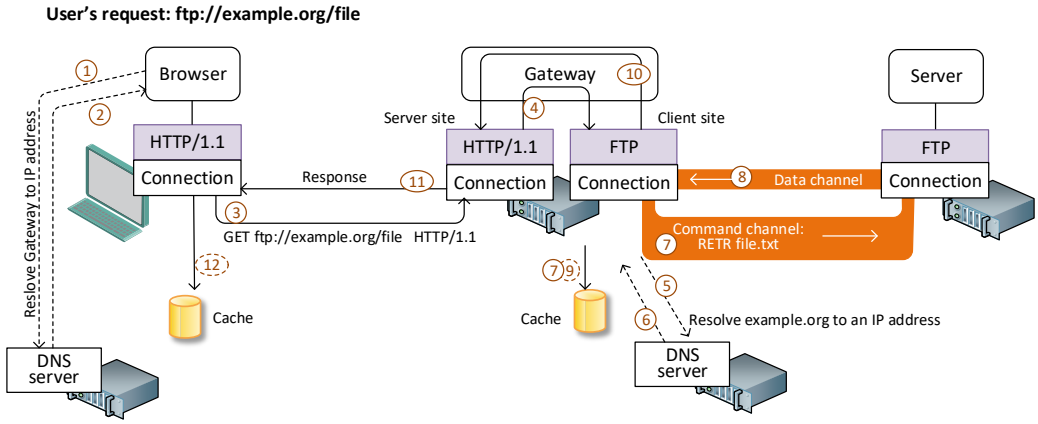



Figure 8.10 *Gateway converting HTTP to FTP*

### 8.6.3 Reverse proxy

While a proxy is most often used as part of a corporate firewall through which employees access the Internet, a reverse proxy is most often run as part of an application firewall through which customers access selected company servers. Figure 8.11 shows that the reverse proxy does not convert the application protocol, but converts the URI. In the URI path, the client indicates to which origin server the request should be forwarded (the use of a path in a URI is referred to as a Junction). However, today's portals usually no longer require users to enter a complicated path (Junction) – they solve it at the level of the portal, in which the reverse proxy is hidden.

Another use of a reverse proxy is load balancing between multiple instances of origin servers. To distribute the load, it is necessary to monitor the availability of origin servers on the application layer (server health check). Thus, the server application must be extended to respond to the load balancer queries about the server state, sometimes called "application ping".

In terms of authentication, we must realize that the client authenticates to the server side of the reverse proxy which passes the name of the authenticated client (or his certificate when authenticating with a certificate) through the client side to the origin server. Information about the authenticated client can be transmitted between the reverse proxy and the origin server, for example, in private HTTP headers. The reverse proxy is thus the core of systems known as **Access Management**, i.e., systems performing centralized authentication.

Authentication between the client side of the reverse proxy and the origin servers is usually performed using certificates with mutual authentication. However, the client side of the proxy usually has one certificate for all origin servers, i.e., the original user's authentication is no longer used here – only his identity is transferred (e.g., in private HTTP headers).

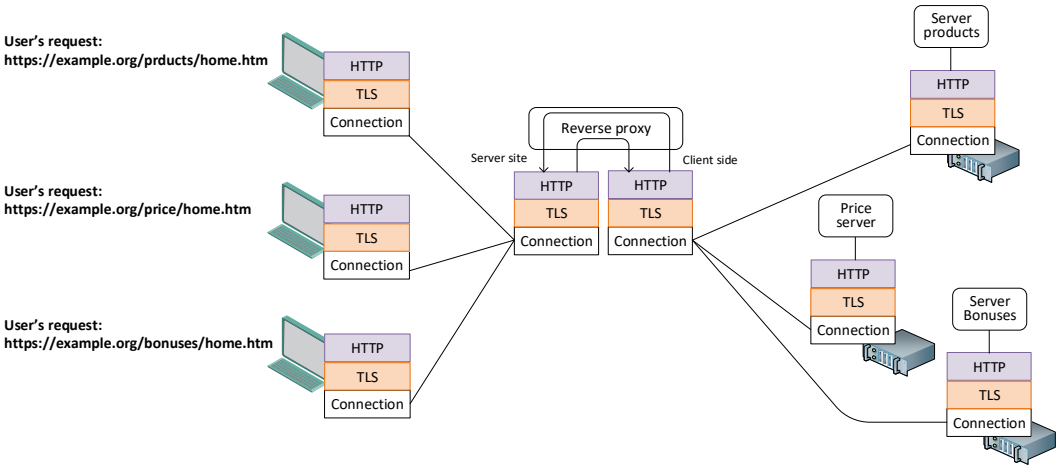



Figure 8.11 *Reverse proxy*

### 8.6.4 Tunnel

A tunnel is another type of intermediary. The client establishes a connection with the tunnel (Figure 8.12). In the connection created in this way, the client inserts the CONNECT method with a parameter that is the name and possibly the port of the origin server. The tunnel translates the name of the origin server into an IP address and, on behalf of the client, establishes a connection with the origin server on the port specified in the CONNECT method. Now the tunnel has two bidirectional connections established:

- Client - tunnel connection.
- Tunnel - origin server connection.

Let us imagine each of these connections as a pair of pipes – one pipe is for transferring data there and the other is for transferring data back (duplex connection). The function of the tunnel is simple – the tunnel will do nothing but ”mechanically weld the pipes together” without knowing what will be transferred in them. Everything that comes from the client is forwarded mechanically in a pipe to the origin server. Similarly, everything that comes from the server will be forwarded to the client.

The result is a direct connection between the client and the origin server. In such a connection, the client can also start to secure the connection with the TLS protocol.

The basic feature of a tunnel is that the tunnel does not ”understand” the data being transmitted. In the past, the tunnel was often used for connections secured by the TLS protocol, i.e., a tunnel was created using the HTTP/1.1 protocol and TLS communication started in the created tunnel. It had the advantage that TLS communication with mutual certificate authentication also passed through the tunnel. A major security risk was that malicious code could come through the tunnel to steal information from the intranet.

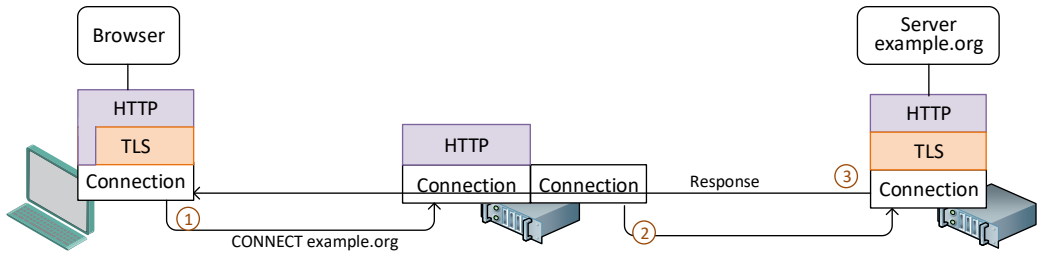



Figure 8.12 *Tunnel*

The tunnel has only the CONNECT method implemented. After connecting the origin server to the client, it no longer participates in HTTP communication.

### 8.6.5 More intermediaries

One or more intermediaries can be placed between the client and the origin server. Various combinations of proxy, gateway and tunnel are possible on the way from the client to the origin server.

### 8.6.6 Intermediaries and HTTP/2 or HTTP/3

In the previous paragraphs, we used the HTTP/1.1 protocol in the examples. This does not mean that the HTTP/2 and HTTP/3 protocols do not support intermediaries. This was mainly because in HTTP/1.1 the method can be easily written in text, and also because we have to realize that HTTP/2 and HTTP/3 use multiple streams. The CONNECT methods are used by a specific streams, which then establish connections with subsequent systems. Additionally, HTTP/3 has TLS built into it.

## 8.7 Methods

Every HTTP request starts with a method (Table 8.4).

## 8.8 Status codes

The status code specifies the success/failure of the request, and the subsequent note textually clarifies the result of the operation.

Result codes are three digits long. The first digit determines the type of the answer, as shown in table 8.5.

## 8.9 Representaon of Data and Metadata

The server returns a representaon of the requested data in response. Data representaon header fields are used to inform the browser how to process the content of the transferred data (e.g., how to display the data to the user).

Examples of header fields Representation of data and metadata are in Table 8.5. The exact specification can be found in RFC 9110 [52].

Table 8.4 *HTTP Methods*

| Method | Description |
|---|---|
| **GET** | A request to transfer the current representation of the origin server. The GET method is used to query the client for a specific representation stored on the server. The query is formulated as part of the URI – after the character "?". |
| **HEAD** | Same as GET, but does not return the response content (returns header only). The HEAD method only requests response header without content. It is mainly used for caches, when the HEAD method can be used to find out if the origin server has more recent data. |
| **POST** | Requests for the processing of the sent content. We use the POST method to send data (e.g., HTML form fields) to the server. |
| **PUSH** | Supports only in the HTTP/2 and HTTP/3. Server push is an interaction mode that allows a server to initialize a request-response exchange to a client. |
| **PUT** | Replaces all current representations of the origin representation with the content of the request. |
| **DELETE** | Removes all current representations of the origin representation. |
| **CONNECT** | Creates a tunnel. |
| **OPTIONS** | Describes the communication options for the origin representation. The OPTIONS method is used to find the communication properties of the server or the requested URI. If we are querying the all properties of the server, we use an asterisk instead of the URI. |
| **TRACE** | Performs a loopback test along the path to the origin server. The TRACE method is similar to the `tracert` command. However, this time we learn not about the number of routers between us and the origin server, but about the number of intermediaries. |

Table 8.5 *HTTP status code overview*

| Code | Type | Description |
|---|---|---|
| **1xx** | Informational | The request has been received, the process continues. Example: `100 Continue` means the initial part of the request has been received and not yet rejected. |
| **2xx** | Successful | The request was successfully received, understood, and accepted. Example: `200 OK` |
| **3xx** | Redirection | Further action needs to be taken to complete the request. Example: `302 Found` `Location: https://server.com/new?uid=bob` |
| **4xx** | Client Error | The request contains bad syntax or cannot be fulfilled for reason caused by client. Example: `401 Unauthorized` |
| **5xx** | Server Error | The request is valid, but cannot be fulfilled for some reason on server side. Example: `500 Internal Server Error` |

Table 8.6 *Header fields for the representation of data and metadata*

| | |
|---|---|
| `Content-Type: text/html; charset=utf-8` | The Media Type of the transmitted data is plain text in HTML format, UTF-8 encoding. |
| `Content-Type: application/javascript` | The data sent must be processed using a javascript application. |
| `Content-Encoding: gzip` | The content is compressed using the gzip method, i.e., it must be de-compressed using the specified method before processing. |
| `Content-Language: en` | The content is in English (en) – see RFC 5646 [57]. |
| `Content-Length: 44280` | The content is 44280 bytes in size. |
| `Content-Location: https://fir.org/source.htm` | Representation identifier. If someone GET requests this representation while the message is being generated, then the `200 (OK)` response will contain the same representation as the message content. |
| `Last-Modified: Thu, 24 Aug 2023 13:17:17 GMT` | The date and time when the source server believes that the selected representation was last modified. See Cache. |
| `ETag: "09ab0c4f11ed91:0"` | Entity tag – identification of the entity representation. See Cache. |

## 8.10 Message Context

The headers listed here provide additional information about the context of the request, including information about the user, user agent, and representation behind the request.

Examples of message context header fields are in Table 8.7. The exact specification can be found in RFC 9110 [52].

Table 8.7 *Message context header fields*

| | |
|---|---|
| `Referer: http://example.org/View.htm` | Specifies the URI from which the origin URI was obtained. See Cookies. |
| `User-Agent: Mozilla/5.0` | Information about the user agent software that created the request, often used by servers to identify the extent of the reported interoperability issues, to work around or tailor responses to avoid the specific user agent limitations, and for the analysis. |
| `Server: CERN/3.0 libwww/2.17` | Information about the software server. |
| `Allow: GET, HEAD, PUT` | The origin server supports GET, HEAD and PUT methods. |
| `Location: https://server.com/new?uid=bob` | For `201 (Created)` responses, the Location value refers to the primary representation created by the request. For 3xx (redirect) responses, the Location value refers to the preferred representation to automatically redirect the request. |
| `Retry-After: Thu, 24 Aug 2023 13:17:17 GMT` | Indicates how long the user agent should wait before making a subsequent request. When sent with a 503 (Service Unavailable) response, Retry-After indicates how long the service is expected to be unavailable to the client. When sent with any 3xx (redirect) response, Retry-After indicates the minimum amount of time the user agent is asked to wait before issuing a redirected request. A specific time or interval in seconds is entered. |
| `Retry-After: 120` | Same meaning as above, but specified as an interval in seconds instead of a timestamp. |

**Note:** The header name is ”Referer” instead of ”Referrer” due to typo in original HTTP specification. It since became widely used spelling of this word when used in HTTP context, but in same cases, the correct form of word is used instead (e.g. Referrer-Policy HTTP header).

## 8.11 Content negotiation

The origin server often has different ways of representing information, for example, in different formats, languages or encodings. Similarly, different user agents may have different abilities, characteristics, or preferences that may affect which representation of the available ones is best to provide. For this reason, HTTP provides mechanisms for content negotiation.

As a rule, a server proposes several ways of representation. A specific way of representation can be followed by the ";" character separating the parameters (e.g., charset). Parameter q (media quality) characterizes support priority. This is in the interval from 0.001 to 1. The value 1 is the highest priority, which is also the default value.

Examples of context negotiation header fields are in Table 8.8. The exact specification can be found in RFC 9110 [52].

Table 8.8 *Content negotiation header fields*

| | |
|---|---|
| `Aaccept: text/html, image/jpeg;q=0.8` | I accept plain HTML text and jpeg images with a priority of 0.8 |
| `Accept-Charset: iso-8859-1, unicode;q=0.8` | I accept iso-8859-1 character set with priority 1 and unicode set with priority 0.8. |
| `Accept-Encoding: compress;q=0.5, gzip;q=1.0` | I accept compress encoding with priority 0.5 and gzip encoding with priority 1 |
| `Accept-Language: cs, cs-CZ;q=0.9,en;q=0.8` | With priority 1 I accept language cs, with priority 0.9 - language CS-CZ and with priority 0.8 - language en |

## 8.12 Protocol Upgrade

This is a tool that provides a simple mechanism for upgrading from HTTP/1.1 to another protocol. Here is an example of a request to transition from HTTP/1.1 to websocket:

```
GET / HTTP/1.1
Host: www.example.com
Connection: upgrade
Upgrade: websocket
```

However, the HTTP/3 protocol can specify a higher layer protocol in the ALPN (Application Layer Protocol Negotiation) extension in the TLSv1.3 Client-Hello packet.

## 8.13 HTTP authentication

Authentication at the HTTP protocol level and at the TLS level should not be confused. This paragraph deals with authentication at the HTTP level.

Authentication of the client in the browser is usually in two stages:

1. The client accesses the server without authentication, the server returns an error and subsequently offers authentication options in the WWW-authenticate header field. It is always a pair: the authentication method and the realm (area) within which authorization is performed. The following example shows the offered authentication methods: Basic, Negotiate and Bearer:

```
HTTP/1.1 401 Unauthorized
WWW-authenticate: Basic realm=intranet.com,
```

```
Negotiate realm=win.intranet.com,
Bearer realm=auth.int.com
```

2. The client selects one of the offered methods and returns.
   Example:
   ```
   Authorization: Basic VXNlcjpwYXNzd29yZA==
   ```
   *Where* `Base64(User:password) = VXNlcjpwYXNzd29yZA==`

Some other authentication methods are as follows:

- The Negotiate method (SPNEGO protocol – RFC 4559 [58]): the client sends a Kerberos ticket in the Authorization header field. It is widely used in MS Windows networks.
- Bearer method: the client in the Authorization header field: sends a Json Web Token of the OAuth 2.0 protocol.

**Note:** The current version of the HTTP protocol prohibits the use of a password in a URI.

**Note:** HTTP also supports Proxy-authentication, i.e., authentication against intermediaries.

## 8.14 Cache

The HTTP cache is specified in RFC 9111 [59]. In the case of Cache, we must first realize that it can be implemented on the client, on the proxy, on the gateway and on the server. It cannot be realized on the tunnel, because the tunnel does not know what it is communicating.

Cache memory can be either shared or private. The shared cache is used to store information that is independent of the user to whom it is intended. If the information is user-dependent, it must not be stored in the shared Cache memory, but it can be stored in the private cache.

There can be multiple occurrences of the same representation in the cache. Each occurrence has a unique Entity Tag identifier, which is listed in the **ETag** header field.

Cache can speed up communication. However, the basic problem is how to ensure that the cache does not respond with the stale data. If the server does not want anything special from the cache, it usually fills in the Date, Last-Modified and Expires header fields in the response. In case of extraordinary requirement, it also fills in the Cache-Control header field. This field can also be used by the client in the query.

**Expires** header field specifies the date and time after which the information is considered expired. Until then, the information is considered fresh. The cache calculates the time for which it can consider the information fresh (freshness lifetime).

Cache maintains information about the time during which it has been stored in the Cache memory or traveled through the network. Age is given in seconds. If the information is provided from the Cache, i.e., not from the origin server, the Age header field is used, indicating the age of the information.

For example, a response containing the following header fields arrives on 12/23/2000 at 20:11:22:

```
Date: Sat, 23 Dec 2000 19:11:22 GMT
Expires: Sat, 23 Dec 2000 22:11:22 GMT
```

Then:

- *age = 3600*.
- *freshness lifetime = 7200*.

If this information were to be forwarded immediately (23/12/2000 at 20:11:22), it would be supplemented by the header field:

```
Age: 3600
```

The problem is with messages that do not contain the Expires header field. This is where implementation matters. It seems quite practical to consider using the Date and Last-Modified header fields. The difference between the values listed in the Date and Last-Modified headers indicates the time the information has been unchanged on the server. It may seem reasonable to keep this information in the Cache memory, for example, after 10% of this difference.

The Cache-Control header field can have a variety of parameters. The client can include the following parameters in the query in the Cache-Control header field:

- **no-cache** – the request must not be answered from the Cache memory.
- **no-store** – this is probably sensitive information, so the information must not be cached.
- **max-age=s** – the client does not want information older than s seconds.
- **min-fresh=s** – the client only wants information that will be considered fresh for at least s seconds.
- **max-stale=s** – the client is willing to accept even information that is out of date, but not by more than s seconds.

The server can include parameters in its response:

- **public** – the answer can also be stored in the shared Cache.
- **private** – the response can only be stored in the private Cache (the response contains private information – another user may have different information from the same URI).
- **no-cache** – the response must not be cached.
- **no-store** – the answer must not be stored on disk (it may contain, for example, sensitive information). The exception is saving to disk (outside the cache) explicitly by the user (e.g., with the right mouse button).
- **must-revalidate** – the server requires that all caches on the path must be refreshed with this request.
- **proxy-revalidate** – similar to must-revalidate, but the cache on the client does not have to be refreshed. In this way, e.g., information for client authentication on the proxy can be preserved in the client's cache.
- **max-age=s** – the server explicitly specifies the maximum time for which the information should be kept in the Cache (Age). If the server includes the parameter max-age=0 in its response, it will force all caches on the way to refresh this information.

- **s-maxage=s** – similar to max-age, but only applies to shared cache.

## 8.15 Conditional requirements

Conditional requests are HTTP requests that indicate a precondition to be tested before the request method is applied to the origin of representation.

Examples of Conditional requirements header fields are in Table 8.9. The exact specification can be found in RFC 9110 [52].

Table 8.9 *Conditional requirements header fields*

| | |
|---|---|
| `If-Modified-Since: Sat, 29 Oct 1994 19:43:31 GMT` | Conditions the GET or HEAD method that the modification date of the representation is newer than the date specified in the header field. Data transfer of the selected representation is prevented if that data has not changed.<br>For example, if there is a representation in the cache, but we are not sure if it is fresh, then we send a request to the server with a condition where is the date of the last representation in the cache. |
| `If-Unmodified-Since: Sat, 29 Oct 1994 19:43:31 GMT` | Conditions the PUT method that the last modified date of the selected representation is earlier than or equal to the date specified in the header field. |

## 8.16 Cookie - HTTP state mechanism

HTTP is a request-response protocol. This does not allow maintaining a session consisting of a series of request-responses between the client and the server. In old HTTP versions it was not possible to pass any information between two requests to the server.

RFC 6265 [60] solved this problem by establishing a simple session during which it is possible for information from a response to be passed to the next request. This information is called a cookie. In its response, the server sends status information to the client with the **Set-Cookie** header field. The client can then repeat this information in his next request.

The browser saves Cookies in a specialized cache for ”cookies and other website data”. Code running in the browser page can also access this data. This is a different cache from the message cache (i.e., for images and files) and different from the browsing history cache.

For example, the client can use cookies to shop in a virtual department store. The client walks through the virtual department store and selects goods for his shopping cart. But the information about what the client has in the shopping cart must be maintained somewhere. This information about the state of the client’s session with the store will be maintained using cookies (in cache cookies):

1. The client enters the virtual store www.examplestore.com via the initial page

https://www.examplestore.com/store, i.e., the client sends an HTTP request:

```
GET /store HTTP/1.1
...
```

2. The server assigns the client a random identifier Customer=”007” upon entry and sets it in the response using the Set-Cookies header field:

```
HTTP/1.1 200 OK
Set-Cookie: Customer="007"; Path="/store"
...
```

3. The client selects the goods (the browser finds out that it has cookies for this server, so it inserts them into the request):

```
POST /shop/jacket HTTP/1.1
Cookie: Customer="007"
...
```

4. The server returns the identification of the selected goods:

```
HTTP/1.1 200 OK
Set-Cookie: Goods="jacket_05"; Path="/store"
...
```

5. The client can choose further, but in the end he must choose the method of transporting the goods:

```
POST /store/transport HTTP/1.1
Cookie: Customer="007"; "jacket_05"
...
```

6. The server returns the identification of the selected transport:

```
HTTP/1.1 200 OK
Set-Cookie: Shipping="C.O.D."; Path="/store"
...
```

7. The client has arrived at the checkout and wishes to confirm the order (we are not dealing with payment now):

```
POST /store/transport HTTP/1.1
Cookie: Customer="007"; "jacket_05"; Shipping="C.O.D."
...
```

8. All that remains for the server is to confirm the order:

```
HTTP/1.1 200 OK
...
```

The ease of using cookies lies in the client’s freedom to browse the virtual store and jump to a completely different server in the middle of a purchase, which may also use cookies and may even have some of the same data in the URI (e.g., path).

The client must, therefore, record a cookie for each server. However, several different applications using cookies can run on the same server, so the client must separately record individual communications using cookies. The client must distinguish individual communications according to the authority in the URI (server) and according to the Path in the Set-Cookie header line.

## 8.17 Third party cookies

The aforementioned use of cookies is referred to as first-party cookies. But a web page consists of many links to individual page elements. These can be either:

- Directly on the web page you are viewing.
- On a third party website. For example images are downloaded from third parties, but unfortunately, not only that. Third parties can also store their cookies in the browser cache – so-called third-party cookies.

**Exercises:** Start DevTool in your browser (F12 – Network) and you will see how many elements the website consists of and what cookies it has cached in your browser.

If your website (e.g., www.example.org) downloads website elements from third parties, it adds the header field Referer to the request to download these elements:

```
Referer: https://www.example.org
```

Thus, the third part obtains information about which page is downloading its source. The third party will generate a random identifier for you, that is connected with your identity. It will continue to monitor you under this identity. They insert this identity into the cookies of your browser. It creates a counter for this identity. With each further download, the counter for this identity increases by 1. It thus maintains the number of accesses, which can still be divided among the individual servers from which the request came. The mechanism of tracking cookies can also be implemented by redirecting to a specialized tracking server that records the tracking of individual pages.

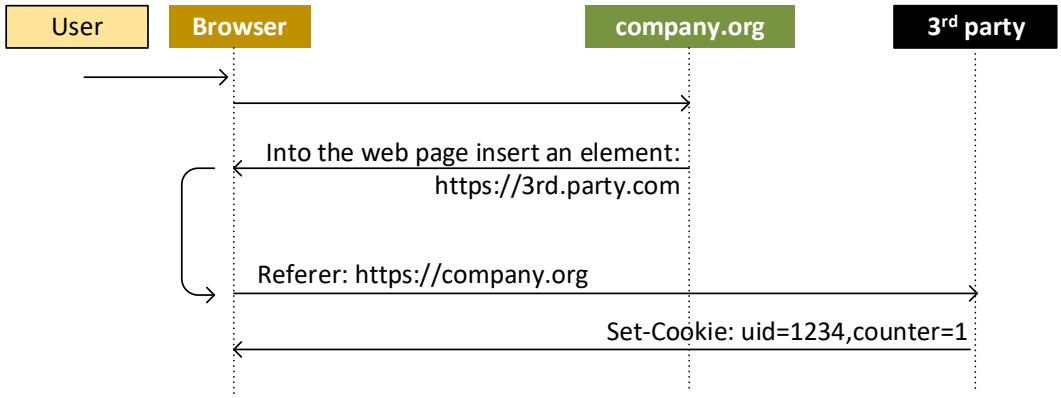



Figure 8.13 *3rd party cookies*

### 8.17.1 Tracking cookies

This mechanism is used, for example, by advertising agencies that provide web pages with advertising banners. Under the identity generated by them, they will find out which websites you visit (from the Referer: header field or from redirection) and can thus offer you targeted advertising. However, they are not able to directly connect the identity generated by them to your physical person.

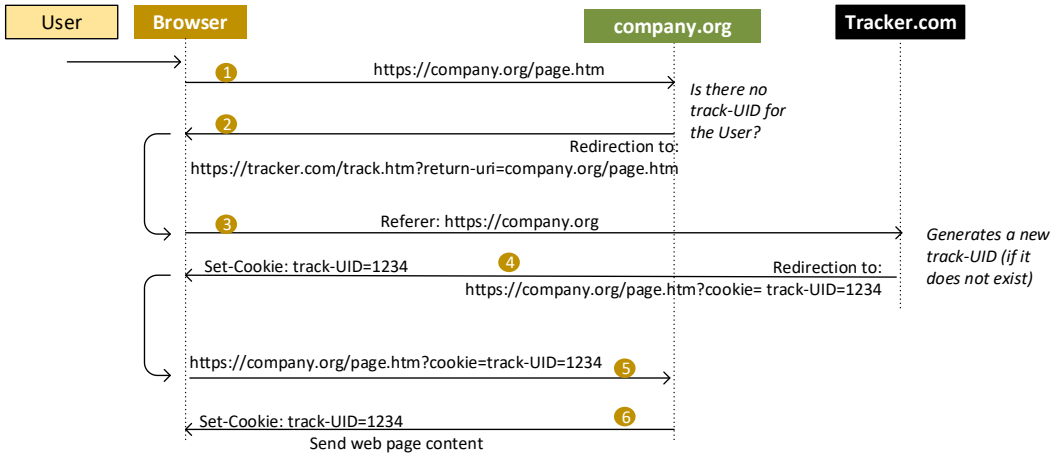



Figure 8.14 *Tracking Cookie*

Another mechanism is **tracking cookies**. The provider of your Example.org website decides to use a 3rd party server for website tracking – say the Tracker.com server. We know that cookies can only be seen by the URI that wrote them into your cache. Tracker.com will also use ”its” cookies. But the mechanism described below ensures that some of them will be identical to the cookies of your company.org server.

1. The user accesses https://example.org/page.htm.
2. The server knows that Tracker.com uses cookies such as ”track-UID”. It will find out if the client has sent such cookies, and if not, it will generate them in the following steps by Tracker.com. The Example.org server redirects the request through the browser to https://tracker.com/track.htm?return-uri=Example.org/index.htm.
3. This request tells Tracker.com to create a new value for the user’s identity. Tracker.com will generate, e.g., ”track-UID=1234”. It writes this value of the tracking cookie and the URI of the copmpany.org website into its database.
4. Tracker.com sets ”Set-Cookie: track-UID=1234” for its cache area and redirects to https://copmpany.org/page.htm?cookie=track-UID=1234.
5. The server Example.org will also set ”Set-Cookie: track-UID=1234” for its area in the browser cache, i.e. identical cookies are set in the cache area for URI Example.org and in the area for Tracer.com.
6. Finally, the server sends the content of the web page.

The thing is that Tracker.com uses multiple websites. Each site maintains its own tracking cookies in the user’s browser, and there are copies from all of them in the Tracker.com URI cache! So, from time to time, just send an insignificant message to Tracker.com and all the tracking cookies of that user will be dumped into the Tracking.com database. This allows Tracking.com to find out which websites the user has visited. It can evaluate this information (i.e., profile of the user) and sell it, for example, to advertising agencies, for which it is significantly more effective information than the above-mentioned primitive mechanism.

**Note:** Steps 3 and 4 using redirection are historical. Today ”Tracker.com” provides the website builder with JavaScript to do this more efficiently. In addition, javascript can also send Tracker.com other information of interest for profiling.

When it can be done so easily with JavaScript, so the next step is for the site provider to agree with two website tracking providers (Tracker1 and Tracker2). It will use JavaScript from both. If they also duplicate each other’s tracking cookies in the user’s cache, then they are cross-linked. This enables the subsequent trade of your personal data between website tracking providers.

The GDPR regulation forbids to use the profiling of natural persons as the processing of personal data. The EU and Council Directive on Privacy and Electronic Communications introduced the right to refuse the storage of profiling cookies.

This is still not enough. There are worldwide providers of so-called web analytics. If you want to see how your website is performing, then you insert the web analytics provider’s JavaScript into the website. This JavaScript not only monitors your pages, but also reads the contents of your cache and sends it to ”analysis”. Information from all pages containing this link is evaluated by analytical tools. The result is beautiful, clear charts that the website operator can receive. But it is processed through a huge number of web providers using artificial intelligence.

# 9 Kerberos

Kerberos [61] originated in 1988 at the Massachusetts Institute of Technology (MIT), when the Internet was the domain of academia. In those days, users logged into servers with passwords that were transmitted over the network in plain text form. Kerberos was supposed to solve both the security of password transmission over networks and the creation of common authentication for multiple applications, i.e., what we know today as Single Sign-On (SSO). Kerberos lived its life without much expansion, hindered by US export restrictions and the need to modify applications to support Kerberos authentication – the so-called ”Kerberized” applications. The breakthrough was brought by Windows 2000, which implemented Kerberos authentication in addition to the native NTLM authentication, i.e., all Microsoft applications were suddenly ”Kerberized”.

The principle is based on the idea that instead of a password, we transmit a message encrypted with this password over the network.

Kerberos introduces some interesting terminology:

- It uses the term principal for the user and the service.
- Principals for which Kerberos provides authentication, form a Realm.

In the world of Kerberos, everyone is a principal. Both users and services (servers) and even Realms. Each principal has its own password, so we must not forget to set up a password for the services, and if we want to set up trust (federation) between Realms, we must set passwords for the Realms as well.

Windows uses the terms **User Principal Name (UPN)** for user principals and **Service Principal Name (SPN)** for service principals. For UPNs, the notation we know from e-mail addresses is used, i.e., “principal @ Realm”. The same notation is used for SPNs, but we must remember that we need to find the designation for the service running on the server, i.e., the part must contain both the service name and the server name before the @. Examples of SPN:

```
http/www.example.org @ example.org
ldap/AD.example.org @ example.org
```

The basis of Kerberos is an independent third party Server for issuing keys – **Key Distribution Center (KDC)**, which consists of:

- A database of principals, their authentication methods, their passwords, and other information about them.
- **Authentication server (AS)**, which performs the user authentication. The result of the

user authentication is the issuance of a ticket granting ticket.

- **Ticket Granting Service (TGS)** – if the user subsequently wants to access a specific service (web, mail etc.), then the (TGS) will issue a session ticket for a required service. Then this ticket enables the user authentication into the requested service.

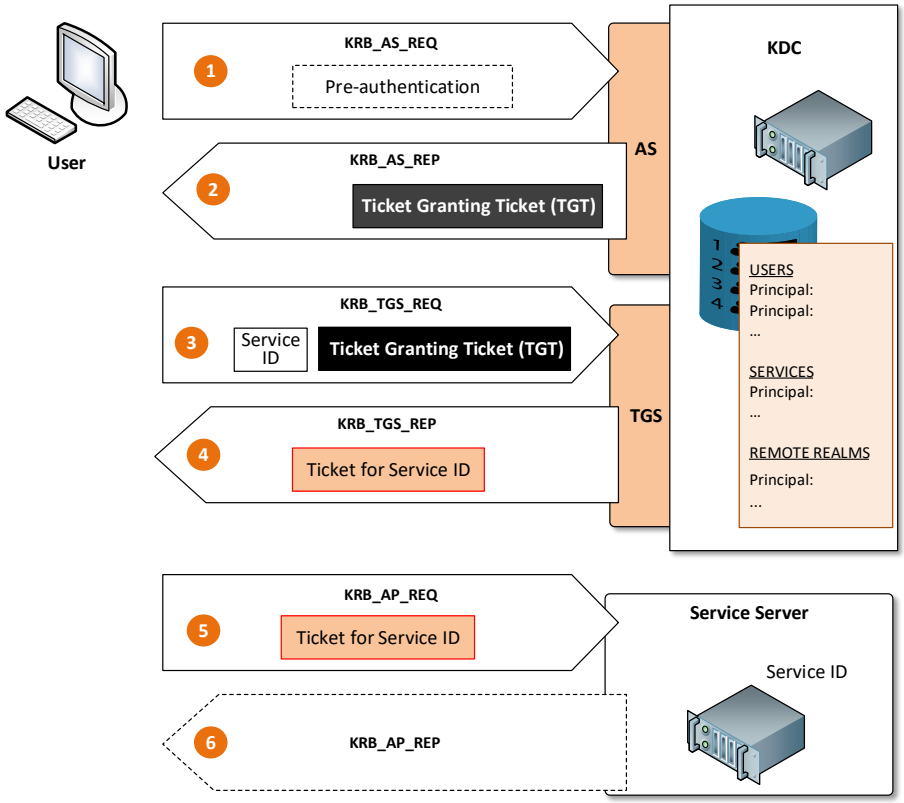



Figure 9.1 *Actors and their communication in the Kerberos protocol (overview)*

The important thing is that the user (human) only enters his authentication data (e.g., name and password) once, and the entire mechanism of issuing and applying session tickets is done for him by the software – it is true Single Sign-On (SSO). Practically: the user (person) comes to work in the morning:

1. Based on his password, he requests the Authentication Server (AS) for authentication.
2. The Authentication Server verifies him, then issues a Ticket Granting Ticket (TGT).
3. If the user wants to log in to a service, his computer automatically logs the user into all applications that the user (person) needs to access throughout the day using the implemented ticket issuing mechanism in the background.
4. User wishes to access Service ID: User's software on background sends a KRB_TGS_REQ message to TGS containing requested Service ID + Ticket Granting Ticket (TGT).
5. TGS will return the ID Service Access Ticket.
6. The User's software logs in to the ID Service using this Ticket.

7. The service may/may not confirm a successful login.

**Notice:** In this chapter, we use the term *timestamp* in a different sense than in previous chapters. In the Kerberos protocol, the term *timestamp* refers solely to recording the current date and time. This record is used in cryptographic operations, making synchronization across all communicating systems absolutely essential for the proper functioning of systems that rely on the Kerberos protocol.

## 9.1 Principle

Kerberos is based on symmetric cryptography (Figure 9.1). The user logs in using a name and password. The software creates a message on behalf of the user, which includes, among other things, the user's name (UPN), but does not contain the entered password – the software will use it in a different way.

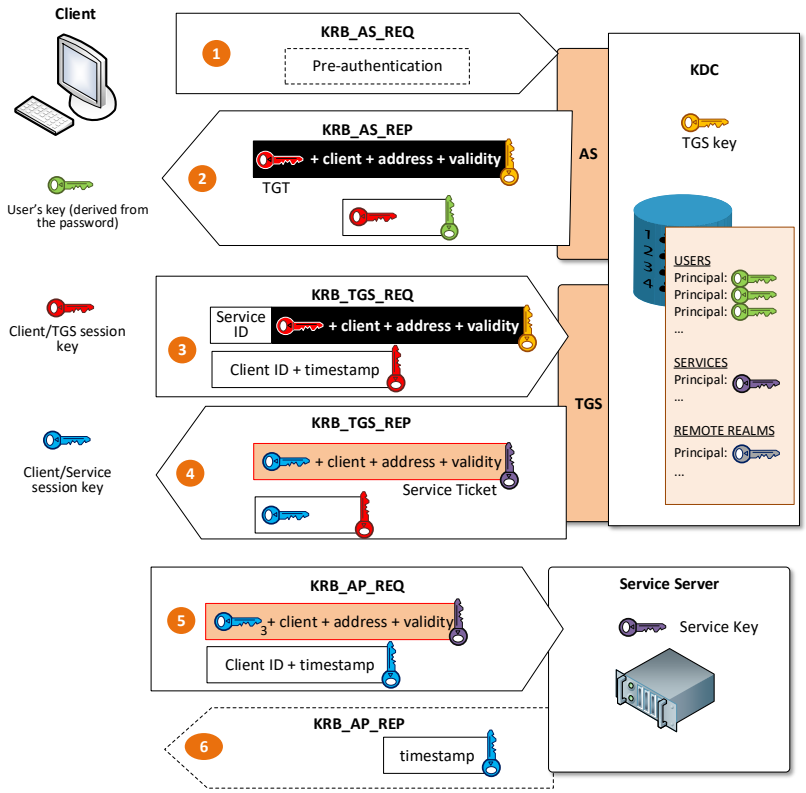



Figure 9.2 *Actors and their communication in the Kerberos protocol*

1. The authentication server (AS) receives this message (arrow 1 in Figure 9.1), looks up the user in the KDC where it finds the user's password.

   Now AS generates temporary user's Session Key for exchange between the client and TGS (Client/TGS session key).

2. The AS sends to the user's software message KRB_AS_REP, which contains:

   - Ticket Granting Ticket (TGT) which, among others, contains: UPN, Client/TGS session

key, client's network address and validly. TGT is encrypted by TGS key.

- Client/TGS session key is encrypted by the client's key derived from the user's password.

Client decrypt Client/TGS session key. From this moment on, the client (i.e., user's software) can request tickets for individual services.

Now the user wants to access a specific Service. Therefore, the client needs to issue a ticket from TGS that will allow him to access this service.

3. The user's Software creates a session ticket request. This request consists of two messages:
   - ID of the requested service + TGT.
   - Client ID and timestamp encrypted by Client/TGS session key.

   The TGS first decrypts the TGT with the TGS key, thereby obtaining the Client/TGS session key. With this key TGS decrypts the Client ID and timestamp, if the time is within the allowable interval, then the user is authenticated.
4. The TGS service returns to the authenticated client:
   - The ticket with which the client will be authorized to the requested service. The ticket among others contains the Client/Service session key, Client ID, client's network address and validity. Ticket is encrypted with the key of Service.
   - A session-key for authenticating the client to the Service (Client/Service session key) which is encrypted by Client/TGS session key.
5. Now the client can request access to the desired service by sending two messages:
   - Ticket for the requested Service.
   - Proof that this is the right client. This proof contains Client ID and timestamp encrypted by the Client/Service session key.

   Service decrypts Ticket, thereby obtaining Client/Service session key. This key is used to decrypt the Client ID and timestamp, if the time is within the allowable interval, then it will provide the service. As proof that it is not a fake service, it may send to the client:
6. Timestamp encrypted by the Client/Service session key.

## 9.2 Pre-authentication

It is assumed (see Figure 9.2) that authentication takes place in such a way that the client sends its User Principal Name (UPN) and the KDC immediately generates the client/TGS session key and TGT, which AS encrypts with the user's password. Only a user who knows his password can decrypt this message and obtain the Client/TGS session key. This method of authentication has a small flaw – the attacker can be sending the names of any users to the Authentication Server, and the AS will be returning answers that the attacker can further investigate, for example, if the user uses a weak password, he can try to crack it.

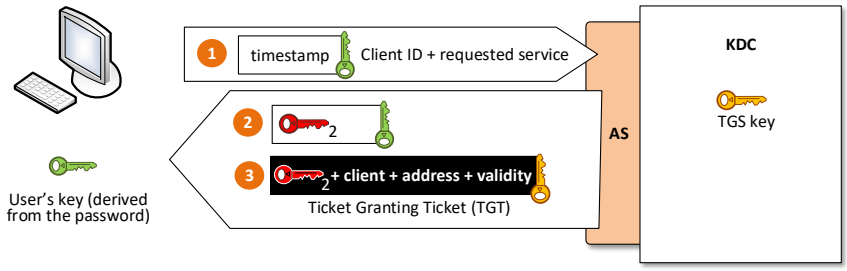



Figure 9.3 *Example of Kerberos Pre-authentication*

It is therefore necessary to prevent any unauthenticated user from sending authentication requests to the Authentication Server on which the Authentication Server automatically responds to them. The solution is the so-called pre-authentication, which contains proof that this is the right client. This proof may be given, for example, by timestamp encrypted by the user's password (Figure 9.3). The Authentication Server then tries to decrypt this structure with the user's password. If it is OK, then TGT is issued.

## 9.3 Authentication by smartcard (PKINIT)

Authentication by smart card is based on asymmetric cryptography (RFC 4556 [62]). The communication described below is another type of pre-authentication.

The smart card has a private key and also a public key, which is also included in the certificate.

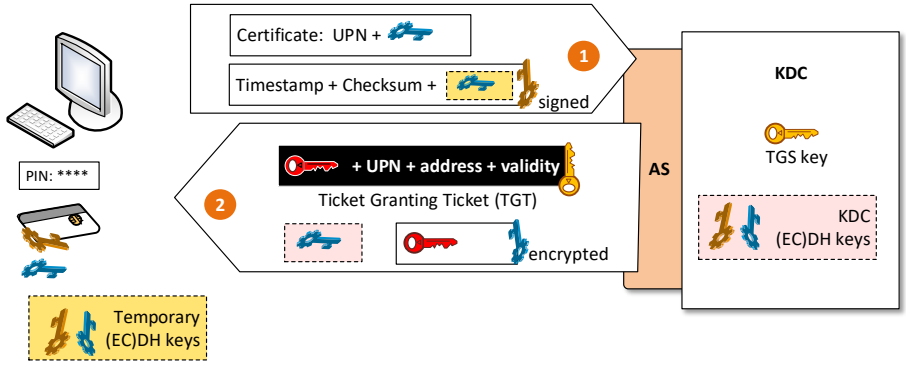



Figure 9.4 *Example of Smartcard logon*

In Kerberos asymmetric cryptography is exclusively used for the user authentication. As soon as the Client obtains TGT and the Client/TGS session key, everything else runs as previously described, i.e., based on symmetric cryptography.

In practice, two possibilities are used:

1. Using the RSA algorithm to sign and encrypt the client/TGS session key. In Figure 9.4, **dashed parts are omitted**:
   (a) The client sends a public key certificate and a private key digitally signed structure including the timestamp and message body checksum.

(b) The KDC verifies the signature and returns: the TGT and the client/TGS session key encrypted with the client's public key.

2. Using the (EC)DH algorithm, i.e., in Figure 9.4 **including the dashed parts**. The client generates a temporary (EC)DH key pair:

   (a) In addition, the client inserts an ephemeral (temporary) public (EC)DH key into the structure that is being sent.

   (b) In addition, the KDC returns its public (EC)DH key. The client/TGS session key is encrypted with the key derived by the (EC)DH key exchange.

Windows also uses smart card authentication to log the user into the system. The user inserts the card into the reader and enters the access PIN to the private key on the card, i.e., everything runs without passwords. The question is where to take the User Principal Name (UPN) when the user only enters a PIN. This is stored in the user's public key certificate in the Subject Alternative extension in the Other Name entry.

## 9.4 Password-based encryption

We used the term "password encrypted" in the previous text. This is just a strong simplification for easy understanding. Obviously, the password cannot be used directly as encryption key, because the encryption key must have a specific format. By "password encrypted" we mean that the relevant symmetric key will be derived from the password by a one-way function to match the correct format, so the derivation of the key from the password will not be reversible and the entropy of the password will be preserved. The operation that converts a password into a symmetric key is referred to as a "string-to-key" operation (see, example of the legendary PKCS#5 standard updated by RFC 8018 [63]).

Kerberos uses the so-called Encryption Type (also known as an enctype) to specify combination of a cipher algorithm with an integrity algorithm to provide both confidentiality and integrity to data. It is a set of cryptographic protocols that contains not only an encryption algorithm, but also a hash algorithm (or MAC), which are used for string-to-key operations, e.g., hmac-sha384-128-aes256 specified by RFC 8009 [64].

## 9.5 Proxy

The proxy (Figure 9.5) is used to solve the following problem: a client authenticates to the Front-end server which passes on the client's request to the Back-end (or the database) using a "technical user" common for all clients, i.e., no authentication information is transferred to the Back-end.

PROXY tickets are used when the client wants to issue a ticket, which he then passes to the Front-end server to access the back-end server on his behalf. In order to make it more difficult to use stolen tickets, tickets are generally bind to a network address. A PROXY ticket is therefore a session ticket that will be used from a different network address than the client's address, therefore the Back-end may require additional authentication of the Front-end server when using PROXY tickets.

The proxy requires the client to know the service name (SPN) on the Back-end server because it must specify it when requesting a session ticket.

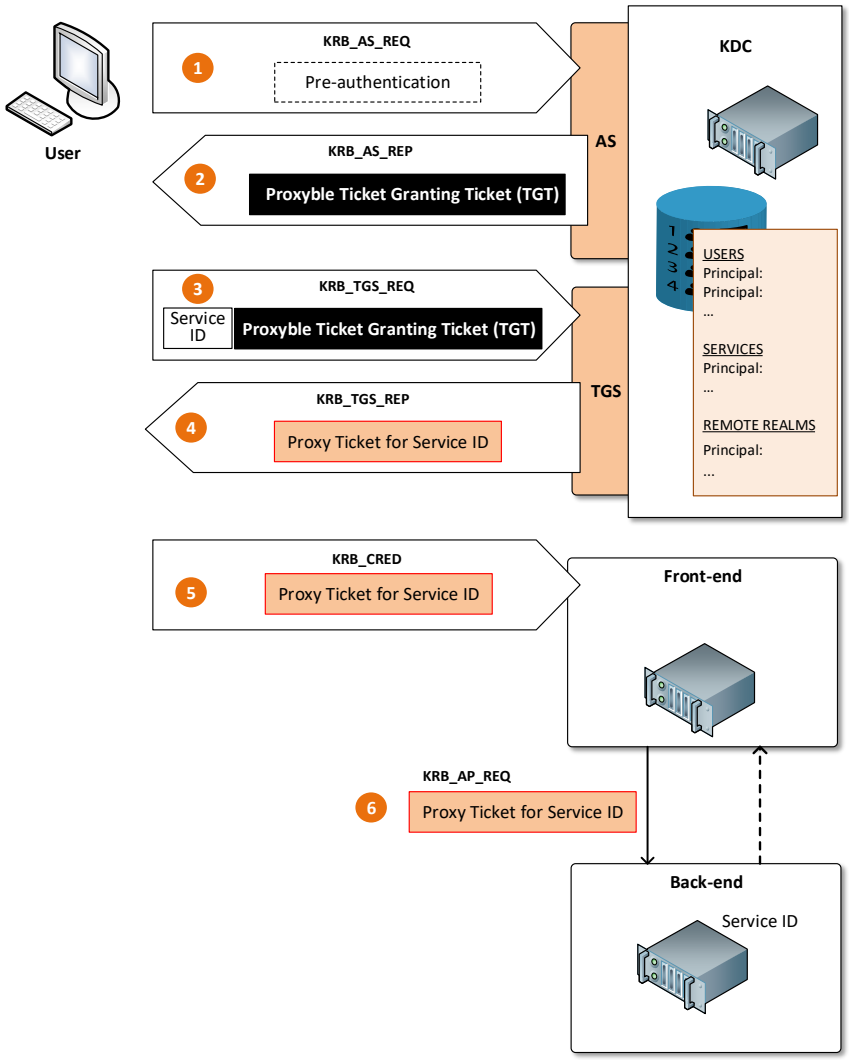



Figure 9.5 *Proxy*

## 9.6 Forwarding

Forwarding is somewhat similar to Proxy, but the client does not need to know the target server's SPN because the client delegates the authority to issue tickets on its behalf to the Front-end server.

In this case, the client must have been issued a Ticket Granting Ticket (TGT) with the FORWARDABLE flag by the Authentication Server. Such a TGT can then be forwarded by the client to the Front-end server in order to request TGS to issue a ticket for the Back-end server on behalf of the client. The ticket thus received for the Back-end server gets the FORWARDED flag.

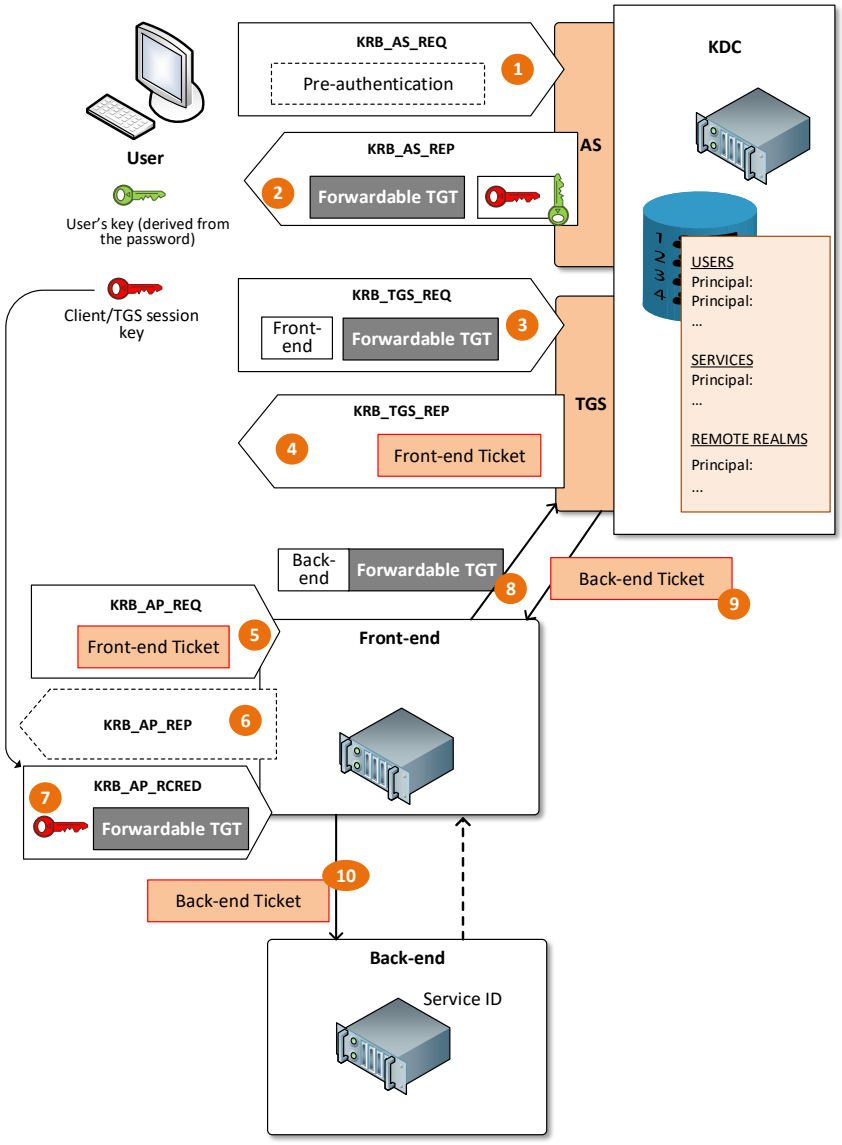



Figure 9.6 *Kerberos Forwarding*

The procedure is as follows:

1. The client authenticates and requests a TGT for its network address and for the Front-end server with the FORWARDABLE flag.
2. KDC issues the requested ticket.
3. Using the issued TGT, the client requests a session ticket to access the Front-end server.
4. KDC issues the requested ticket.
5. The client authenticates using the ticket issued to the Front-end and requests two-sided authentication.

6. The Front-end authenticates itself to the client. As a result, both the client and the Front-end are mutually authenticated.
7. The client sends its TGT and Client/TGS session key to the Front-end, i.e., delegates its identity to the Front-end.
8. The Front-end will use the client's TGT for issuing the ticket of back-end at the KDC.
9. The KDC issues a session ticket for the Back-end server with the FORWARDED flag.
10. The Front-end server authenticates itself by passing on the issued session ticket to the Back-end server.
11. If two-sided authentication is required, the back-end is also authenticating itself.

Passing the client's session key to the Front-end server is a sensitive operation, so sometimes a special TGT is issued for a specific Front-end, i.e., it will use a separate user key.

## 9.7 Trust Between Realms

Kerberos allows a principal in one Realm to authenticate to a server in another Realm using a cross-Realm trust relationship. Since the KDC can only issue tickets for its Realm's services, a trick is used – a foreign Realm registers as a special principal in our Realm. A password is also registered with this principal, from which the corresponding encryption key is derived:

1. The user's client software requests the KDC of its Realm (local KDC) to issue a ticket for a service that is registered in a foreign Realm. If the client is aware that this service is registered in a foreign Realm, it marks its request with the NAME_CANONICALIZE flag.
2. The local KDC creates a TGT that points to the KDC of the foreign Realm. This is a ticket for a foreign KDC. It will use the password given in the special principal of the foreign Realm to encrypt the key of this session.
3. The client forwards this newly issued ticket to the ticketing KDC of the foreign Realm.
4. The KDC of the foreign Realm creates a session ticket for the requested service.
5. The client applies the session ticket for the requested service.
6. Optionally, two-way authentication can occur.

Registering a special principal creates a trust relationship with a foreign Realm. This relationship is one-sided, if it is to be two-sided, then similarly foreign Realm must register a special principal of our Realm.

Trust relationships are transitive, i.e., if Realm A trusts Realm B and Realm B trusts Realm C, then Realm A trusts Realm C. The client can have a TGT issued in Realm A that points to Realm B. Subsequently, Realm B issues it a TGT that points to Realm C, whose KDC issues him a session ticket for the requested service.

If there are multiple Realms, creating direct trust relationships between Realms can result in registering a large number of special principals, which can be difficult to administer. In such a case, Kerberos recommends creating a tree structure of trust relationships between

Realms. Due to the transitivity of trust relationships, we can then have a trust relationship even with the most distant Realm (e.g., through the root of a tree). If we need to communicate frequently with the farthest Realm, then nothing prevents us from creating a direct trust relationship, i.e., a shortcut between the branches of the trust tree.

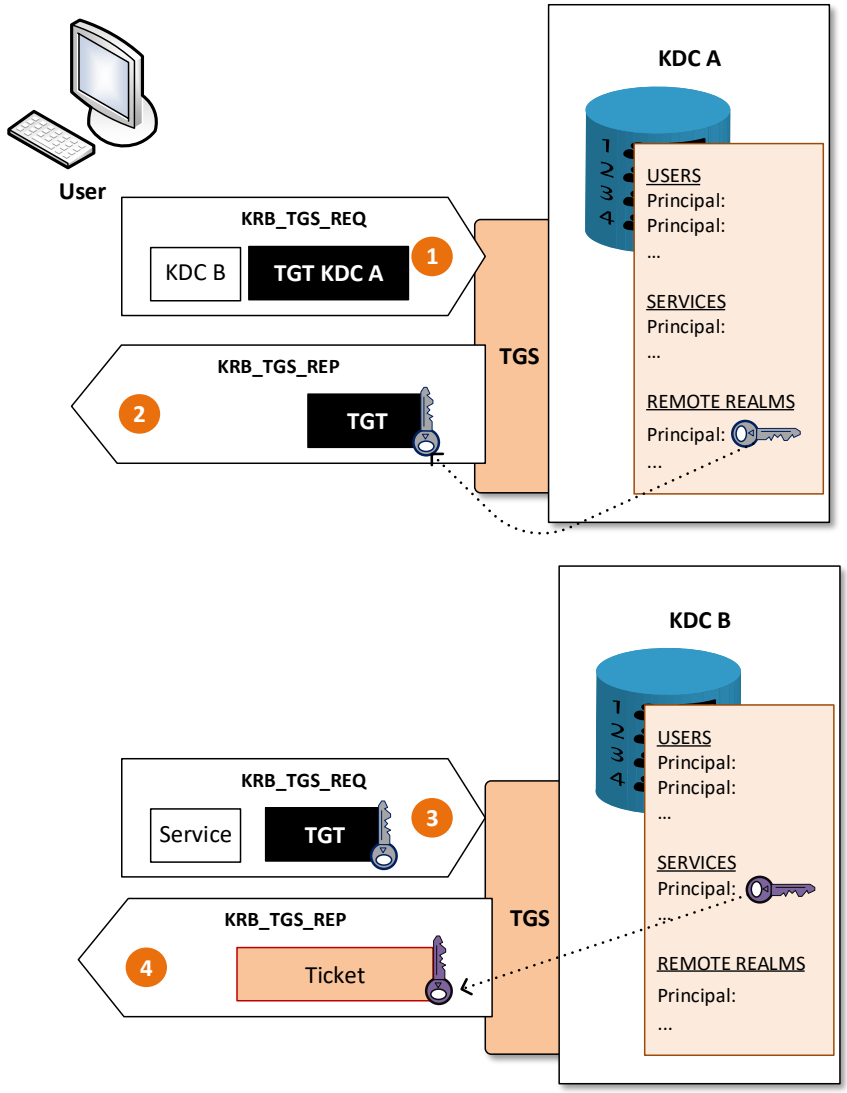



Figure 9.7 *Trust between Realms*

The problem is how to find out what services are available in foreign Realms. But this is not a problem with the Kerberos protocol. Microsoft, for example, solves this with the Global Catalog component, which, among other things, contains SPNs of interest to all Realms, e.g., in the company's internal network.

## 9.8 Timers

Kerberos is a general purpose protocol. The usual default Kerberos settings are listed in the Table 9.1.

Table 9.1 *Typical default Kerberos settings*

| | |
|---|---|
| **Maximum clock skew** | 5 minutes |
| **Maximum ticket validity period** | 10 hours |
| **The maximum period for which the ticket can be renewed** | 7 days |

Although 5 minutes for the maximum clock skew between the time specified in the request and the time of the KDC (or service) may seem long, Kerberos is quite difficult to operate without installing time synchronization services (e.g., NTP).

## 9.9 Ticket structure

The structure of a Kerberos ticket is shown in the Table 9.2.

Table 9.2 *Kerberos Ticket Fields Structure*

| Field | Name | Description |
|---|---|---|
| Version | Tkt-vno | 5 |
| Empire | Realm | Realm that issued a ticket |
| SPN | sname | Service Principal Name (SPN) |
| Encrypted (enc-part) | flags | Bitmap of flags (INITIAL, PRE-AUTHENT, …) |
| | crealm | Realm in which the user is registered |
| | cname | UPN |
| | transited | List of Realms that participated in user authentication |
| | authtime | The time the user authenticated |
| | starttime | The time from which the ticket is valid |
| | endtime | The time until which the ticket is valid |
| | renew-till | The time until which the ticket can be renewed |
| | caddr | Network address from which the ticket can be used (not required, then it can be used from any address) |
| | auth-data | Authorization data: contains ticket usage restrictions, but Microsoft inserts a Privilege Attribute Certificate (PAC) here. It contains information about the user's privileges and is used whenever they authenticate in a Microsoft Active Directory domain |

A very interesting item in the ticket structure is the ticket flags (Table 9.3).

Table 9.3 *Ticket flags overview*

| Flag | Description |
|---|---|
| **INITIAL** | The ticket was issued based on the user's initial authentication, i.e., it was issued by an AS - not a TGT. |
| **PRE-AUTHENT** | The initial authentication of the user also included pre-authentication. |
| **HW-AUTHENT** | Initial authentication was done using a hardware device. |
| **INVALID** | The ticket is not valid. It is used for tickets issued with a later validity date (e.g., for batch processing). Tickets issued at a later date must be validated by the KDC before use. |
| **RENEWABLE** | The ticket can be renewed before it expires. The renewed ticket contains the new session key. Tickets with this flag usually also contain a time after which the ticket can no longer be renewed. |
| **MAY-POSTDATE** | Must be set in the Ticketing Ticket (TGT) if a ticket with a later validity date is requested. |
| **POSTDATED** | Ticket issued with a later validity date (postdated). |
| **PROXIABLE** | A Ticket Issuing Ticket (TGT) can be used to issue a ticket with the PROXY flag. |
| **PROXY** | PROXY tickets. |
| **FORWARDABLE** | A ticket issuing ticket (TGT) can be used to issue a ticket with the FORWARDED flag. |
| **FORWARDED** | A FORWARDED ticket can be used similarly to a PROXY ticket, i.e., for web server access to the database on behalf of the client. |
| **TRANSITED-POLICY-CHECKED** | If the ticket was passed between Realms, then the KDCs of the individual Realms may have established rules (policies) for this transfer. If these policies were checked when issued, then the ticket is marked with the TRANSITED-POLICY-CHECKED flag, if not, then it is marked with the DISABLE-TRANSITED-CHECK flag. |
| **OK-AS-DELEGATE** | If the client delegates authority to the server to act on his behalf, he may have doubts as to whether he is acting correctly. Using the OK-AS-DELEGATE flag, the KDC can signal to the client that the server specified in the ticket has been deemed eligible for delegation by Realm policy. |
| **NAME_CANONICALIZE** | The name of the principal (server/service) that the KDC returns in its response (also in the ticket) should match the name of the principal that was specified in the request. However, especially if the target server/service is in a different Realm, the KDC may return a different name (alias) of the same principal. The NAME_CANONICALIZE flag in the request indicates that the client allows the return of a different principal name than requested. |

## 9.10 Examples of tickets

In Windows, the client can list the currently saved tickets in the local cache by `klist` command. For example:

```
C:\>klist

Cached Tickets: (2)

#0>     Client: bob @ example.com
Server: krbtgt/example.com @ example.com
KerbTicket Encryption Type: AES-256-CTS-HMAC-SHA1-96
Ticket Flags 0x40e10000 -> forwardable renewable initial
pre_authent name_canonicalize
Start Time: 9/12/2023 6:45:14 (local)
End Time:   9/12/2023 16:45:14 (local)
Renew Time: 9/19/2023 6:45:14 (local)
Session Key Type: AES-256-CTS-HMAC-SHA1-96
Cache Flags: 0x1 -> PRIMARY
Kdc Called: AD01.example.com

#1>     Client: bob @ example.com
Server: cifs/exampleshare @ example.com
KerbTicket Encryption Type: AES-256-CTS-HMAC-SHA1-96
Ticket Flags 0x40a50000 -> forwardable renewable pre_authent
ok_as_delegate name_canonicalize
Start Time: 9/12/2023 6:46:36 (local)
End Time:   9/12/2023 16:45:14 (local)
Renew Time: 9/19/2023 6:45:14 (local)
Session Key Type: AES-256-CTS-HMAC-SHA1-96
Cache Flags: 0
Kdc Called: AD01.example.com
```

In this statement we can see the following:

- The name of the Realm is `example.com`. The KDC name is `AD01.example.com`.
- UPN is Bob.
- Bob has:
  - Ticket number 0 is TGT (`krbtgt`).
  - Ticket number 1 is for the shared file system (`cifs`). This shared file system is called `exampleshare`.
- The time from which the ticket is valid.
- The time until which the ticket is valid.

- The time until which the ticket can be renewed.
- Algorithms „string-to-key“ is `AES-256-CTS-HMAC-SHA1-96`.
- KDC name is `AD01.example.com`.

## 9.11 SPNEGO protocol

The SPNEGO (*Simple And Protected Negotiate*) protocol is most often used for authentication to the web using the Kerberos protocol [65] (Figure 9.8).

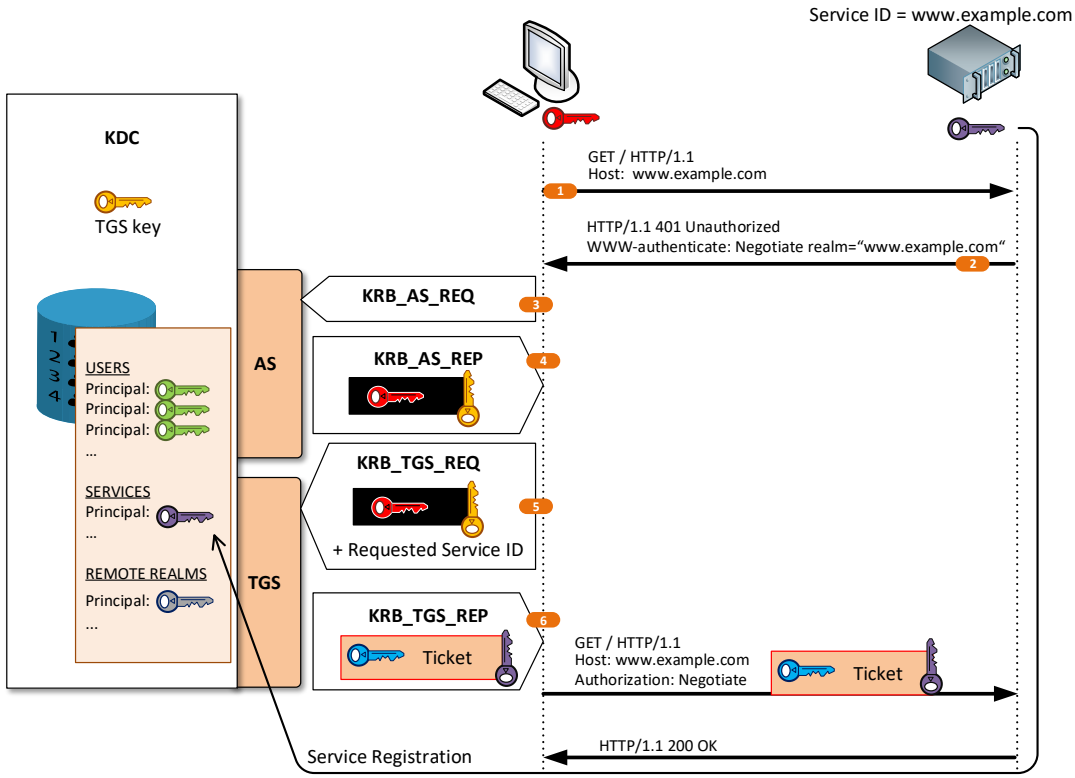



Figure 9.8 *SPNEGO*

In our case, the client accesses the server `www.example.com` from their browser. This server must be registered in advance so that its SPN (i.e. its Service ID) and password are stored in the KDC. It is important that the **`www.example.com` password must also be specified with the SPN** in KDC.

# 10 SAML, OAuth, Open ID Connect and Identity Management

SAML and Open ID Connect are client authentication protocols, OAauth is authorization protocol. They are primarily intended for authentication/authorization over the HTTP/HTTPS protocol, as they use a redirection mechanism.

## 10.1 SAML

Security Assertion Markup Language (SAML) version 1.0 was proposed by Organization for the Advancement of Structured Information Standards (OASIS) in 2002. In 2005, version 2.0 was released.

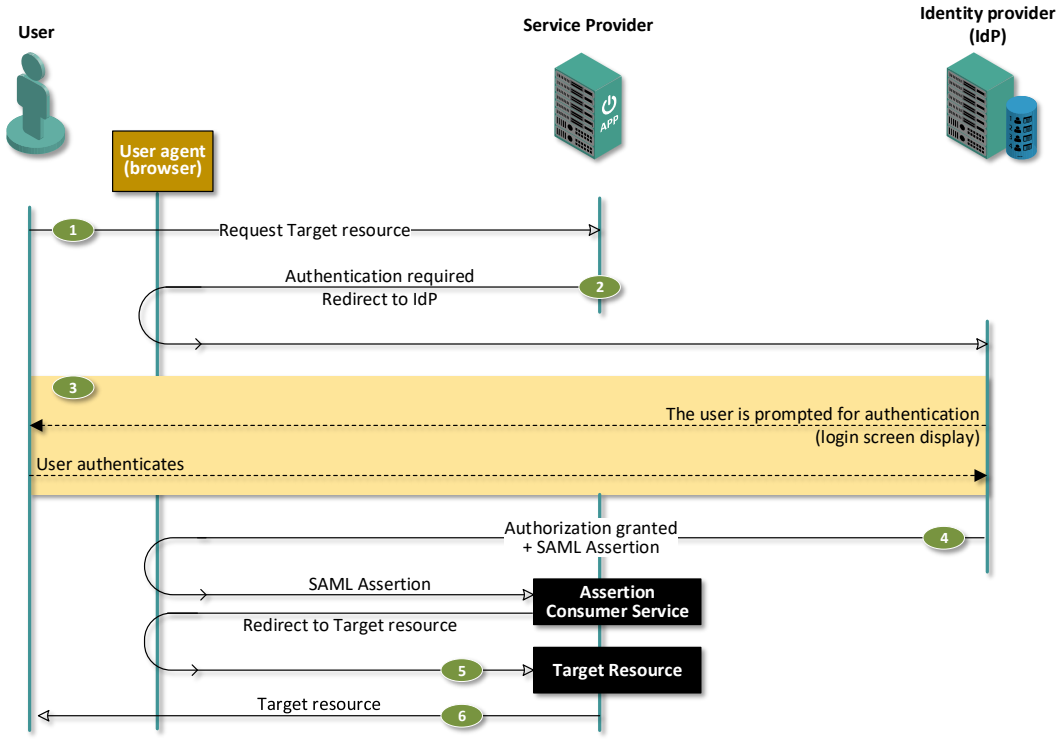



Figure 10.1 *An example of SAML usage*

The SAML protocol is built on the concept of Service Providers (Web Servers), that delegates user authentication to a specialized server known as an **Identity Provider (IdP)**. The IdP authenticates the user and confirms successful authentication by issuing a **SAML assertion**. A SAML Assertion is an XML structure typically signed by the IdP and may also be encrypted.

Importantly, the IdP can also maintain the roles of individual users in its database. These roles can be inserted into SAML assertions.

Figure 10.1 shows an example of SAML usage based on HTTP redirection. The user of the web browser requests the service provider to provide the target resource (1 in Figure 10.1). The service provider requires authentication, so it redirects the request to the Identity Provider (2 in Figure 10.1) with the help of redirection. The identity provider authenticates the user and, in the case of a positive result, issues a SAML Assertion of the completed authentication. Finally, the user equipped with the SAML Assertion will be redirected to the Service Provider.

Redirection from the IdP is first done to the Assertion Consumer Service, which processes the SAML Assertion and creates a security context. After that, the redirection is done to the target resource. Finally, the Target Resource is provided to the User.

Therefore, the identity provider carries out user authentication. The result of authentication is the issuance of a SAML Assertion, on the basis of which the Service Provider provides the required resource.

## 10.2 Identity Federation

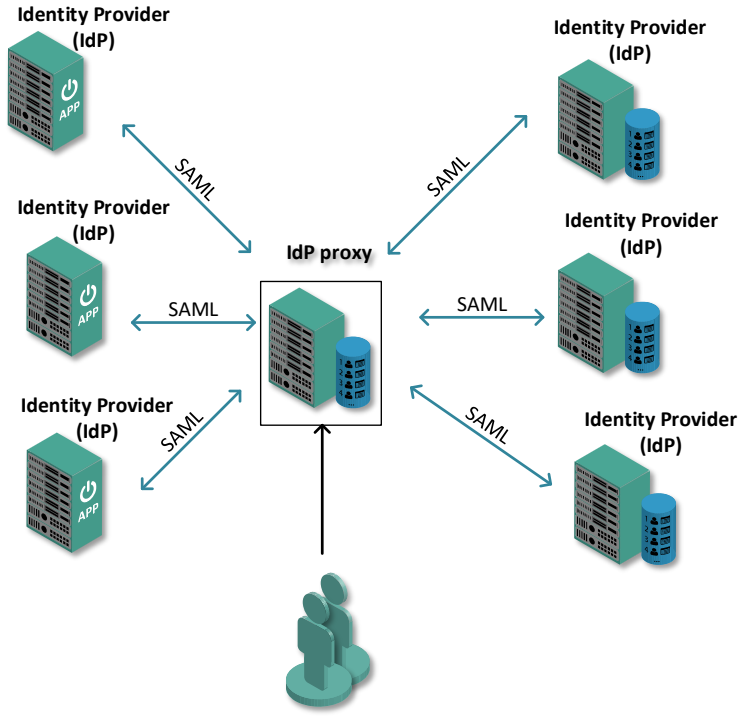



Figure 10.2 *Identity Federation*

An identity federation can be built using an **IdP proxy** (Figure 10.2).

A proxy IdP is an IdP that maintains the lists of:

- Service providers and for each Service provider it is indicated which IdPs they trust (whose SAML assertions they accept).
- IdPs that trust IdP proxies.

For all users there is IdP proxy single point of contact:

1. Users tell which service of which Service Provider they want.
2. The IdP proxy searches for an IdP that the requested Service Provider trusts.
3. Subsequently, the IdP proxy issues a SAML assertion to the user for the searched IdP.
4. IdP proxy sends SAML assertion (proxy assertion) to the searched IdP. Searched IdP acts as Service provider for proxy assertion.
5. The searched IdP issues a SAML assertion to the requested Service provider.
6. The Service provider provides the requested service.

Again, this is only one of the possible cases.

## 10.3 JWT

The SAML specification is very general and flexible. However, this generality comes at the cost of complexity — resulting in verbose and difficult-to-process Assertions. For this reason, authentication and authorization information is increasingly represented not in the form of SAML, but using a more lightweight format: **JavaScript Object Notation (JSON)**.

This shift led to the development of the **JSON Web Token (JWT)** standard. JWT is defined in RFC 7519 [66] and provides a compact, URL-safe means of representing claims to be transferred between two parties. The structure and purpose of a JWT are conceptually similar to those of a SAML Assertion or a Kerberos ticket.

A JWT used for authorization is typically issued by an authority known as an **Authorization Server** — a role similar to that of the Identity Provider in the SAML architecture.

The JWT structure issued by the Authorization Server binds the client identity with authorization to the requested resource of the Resource server (like to Service Provider in SAML). Optionally, the JWT can be signed, i.e., embedded in the **JWS (JSON Web Signature)** structure [67]. The signature here means not only an electronic signature based on asymmetric cryptography, but also a MAC. RFC 9068 [68] recommends for OAuth using algorithms for electronic signature and not for MAC.

## 10.4 JWS

JWS consists of header, body and signature. All three parts are Base64URL encoded, concatenated and separated by a "." character. The header contains the information needed to verify the signature and the body contains the signed data (i.e., JWT).

For the sake of completeness, it should be added that JWS can also be encrypted, i.e., inserted into the JWE (JSON Web Encryption) envelope.

### 10.4.1 JWS Header

The header contains information about the electronic signature algorithm and possibly a thumbprint of the Authorization Server certificate.

**Example:**

```
    {
  "typ": "at+jwt"
  "alg": "HS256",
  "x5t": "SHA- thumbprint from the Authorization Server certificate",
 }
```

Content type `"at+jwt"` was registered for the authentication JWT. The HS256 algorithm is an HMAC with a SHA-256 hash. Algorithms are registered by IANA (Internet Assigned Numbers Authority) under JOSE (*„JSON Object Signing and Encryption*“).

### 10.4.2 JWS body

The body (JWS Payload) contains JSON claims – just like in email header lines, but in JSON syntax. Some climes are mentioned in Table 10.1:

Table 10.1 *Basic JWT Claims*

| **Claim** | Meaning |
|---|---|
| **iss** | JWT Issuer |
| **sub** | Subject for whom the JWT was issued |
| **aud** | Audience (the resource, that the JWT gives access to) |
| **exp** | Expiration Time |
| **iat** | Issued at |
| **client_id** | Client ID |
| **jti** | JWT ID (case-sensitive unique token identifier) |
| **roles** | The list of the user’s roles |
| **groups** | Groups the user is a member of |
| **entitlements** | The list of the client’s permissions |
| **scope** | The scope of the service or the resource for which the JWT will be used |

**Example:**

```
{
 "iss": "https://auth.server.com",
```

```
  "aud": "https://www.example.org",
  "sub": "pepa26",
  "jti": 1234567890,
  "exp": 1694502441,
 }
```

## 10.5 OAuth 2.1

OAuth (*short for open authorization*) is a widely used authorization framework that allows internet users to grant third-party websites or applications limited access to their resources hosted on another service — without sharing their login credentials (such as passwords).

OAuth introduces the following terms:

- **Resource owner** – an entity capable of granting access to a protected resource. When the resource owner is a person, it is referred to as a user.
- **Resource server** – the server hosting the protected resources, capable of accepting and responding to protected resource requests using access tokens.
- **Client** – an application making protected resource requests on behalf of the resource owner and with its authorization.
- **Authorization server** – the server issuing access tokens to the client after successfully authenticating the resource owner and obtaining authorization.

While in Figure 10.3 it is assumed that the client is a web browser, in the case of OAuth, the client is usually an executable code that is either downloaded as JavaScript to the browser or runs on the server and the browser only renders information for the user. We have a total of three client profiles:

- **Web application** – a web application is a client running on a web server. Resource owners access the client through an HTML user interface rendered in a user agent (e.g., web browser) on the device used by the resource owner. Both the client's credentials and any access tokens issued to the client are stored on the web server and are not exposed or accessible to the resource owner.
- **Browser-based application** – a browser-based application is a client in which the client code is downloaded from a web server and executed within a user agent on the device used by the resource owner. Protocol data and credentials are easily accessible (and often visible) to the resource owner. If such applications wish to use client credentials, it is recommended to utilize the Back-end for Front-end pattern. Since such applications reside within the user agent, they can make seamless use of the user agent capabilities when requesting authorization.
- **Native application** – a native application is a client installed and executed on the device used by the resource owner. Protocol data and credentials are accessible to the resource owner. It is assumed that any client authentication credentials included in the application can be extracted. Dynamically issued access tokens and refresh tokens

can receive an acceptable level of protection.

The OAuth standard distinguishes between two types of clients:

- **Confidential clients** who are registered on the Authorization Server and log in to the Authorizaion Server using a name and password, a shared secret or a private key belonging to the login certificate.
- **Public clients**, i.e., clients without login details.

In the case of a confidential client, the client must be registered on the Authorization server. Usually, registered data contains:

- Unique client ID.
- One or more URI Redirection endpoints.
- Authentication data (for example, the client's login certificate).

If certificate is registered, it can also be self-signed, i.e., it is not necessary to use PKI (certification authority). It should be noted that in the case of a client running on a web server, the registration is done by the client's developer/administrator.

So, we have two authentications:

1. In the case of a confidential client, authentication of the client to the Authentication Server.
2. Authentication of the user (resource owner).

The OAuth standard does not specify OAuth protocol negotiation, as we know it, for example, in standards for network protocols. It only defines the following Web API endpoints (i.e., URLs and requests to communicate with them):

1. Two on the Authorization Server side:
    - **Authorization Endpoint** is used to authenticate the owner of the resource and obtain permission to access the resource. The authentication method is outside the OAuth specification, which allows the implementer to use any authentication method. If the user is authorized, the result is the issuance of an Authorization code that is directed to the Redirection Endpoint URI of the client. The authorization code is essentially a one-time password for issuing an Access Token (JWT). The authorization code should not be valid for more than 10 minutes.
    - **Token Endpoint** is used to issue Access Tokens. Optionally, a Refresh Token (one-time password to renew the Access Token without the need for further authentication) can be issued. Using the Access Token, the resource (service) is then provided by the Service Provider.
2. One on the Client side:
    - **Client Redirection Endpoint** receives the Authorization Code and issues the Access Token.

Figure 10.3 shows OAuth handshake. However, operations can be different in different implementations. The picture comes closest to the Browser-based application profile, in the

case that the user authenticates in the browser that also accesses the Resource Server:

1. The User requests a resource from the Resource Server.
2. Authentication is required to provide the resource.
3. The client identity and the Client Redirection Endpoint URI are transferred to the Authorization server.
4. The authorization server authenticates the client. Out of the OAuth protocol specification depends on the implementation of the Authorization Server. It can also take place through another channel (e.g., mobile application).
5. In the case of a positive authorization result, the Authorization Code is returned to the Client Redirection Endpoint. An authorization server can set an HTTP cookie. In such a case, it can maintain a session between the client and the Authorization Server, e.g., for logging in to other Resource Servers, without the user having to authenticate, i.e., cookies are used to implement SSO relation.

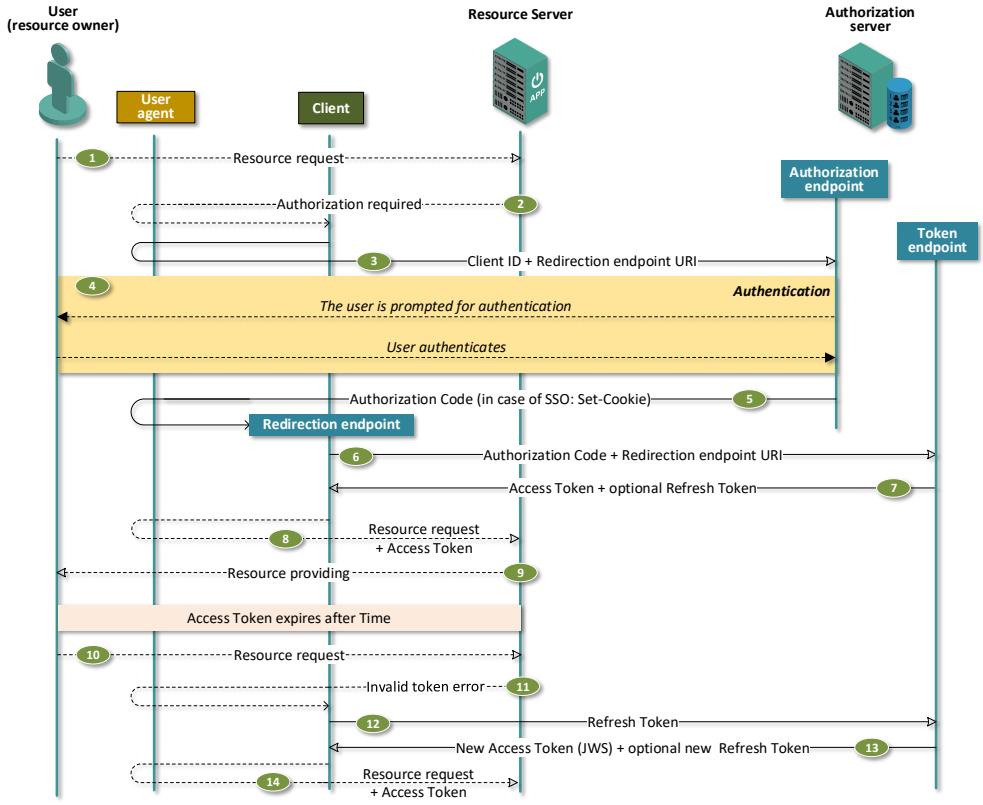



Figure 10.3 *OAuth handshake*

6. The client sends the Authorization Code and Redirection URI to the Token Endpoint of the Authorization Server.
7. The authorization server returns the issued Access Token (JWS) to the Client Redirection Endpoint URI.

8. Now the client can repeat the user's request supplemented with the issued Access Token.
9. Resource Server provides requested resource.

After access Token expired:

1. The user requests a service resource.
2. However, the service provider (resource) will reject the request because the Access Token has expired.
3. The client will use the Refresh Token to renew the Access Token.
4. The Token Endpoint will return a new Access token and possibly a new Refresh Token.
5. A service (resource) can be requested with a newly issued Access Token.

## 10.6 Open ID Connect

Open ID Connect (OIDC) is a superstructure on top of OAuth, which allows the Resource Server not to solve the authentication process, but to rely on authentication providers, so-called Open ID providers. In the OIDC protocol, the Resource Server is called the relying party that runs the OAuth client.

In addition, OIDC introduces the **UserInfo Endpoint**, on which the relying party can obtain information about the authenticated user.

The interesting thing is that the Token Endpoint does not only return an Access Token and possibly a Refresh token, but also returns a so-called **ID Token**, which is a confirmation that the User has been authenticated. It is also in JWT format and can be signed by an authorization server, i.e., embedded in JWS. The ID token does not replace the Access Token, but for the Relying Party it serves, for example, as evidence in the event of a complaint, as it contains information about the authentication that took place. The IT token is not intended for granting access to the resource – this is what the Access Token is for.

Open ID Connect handshake is as follows:

A. Prerequisites:
   - The Relying party has registered the client with the Open ID provider, for example, it has obtained a login shared secret or registered a login certificate.
   - The user has created a Client ID account with the Open ID provider.

B. The Process:
   1. The User wants a resource from the Relying party (from the Resource Server from the point of view of the OAuth) and for this purpose wants for authentication to use the Client ID account with the OIDC provider (OIDC server).
   2. The Relying party directs the request to the Authorization Endpoint of the OIDC server.
   3. The OIDC server not only authenticates the client, but first asks it for consent

to process personal data and to specify the User's personal information that it can provide to the Relying party.

4. The result of the client authentication is the Authorization Code.
5. Based on the Authorization code, the client requests the issuance of an Access Token.

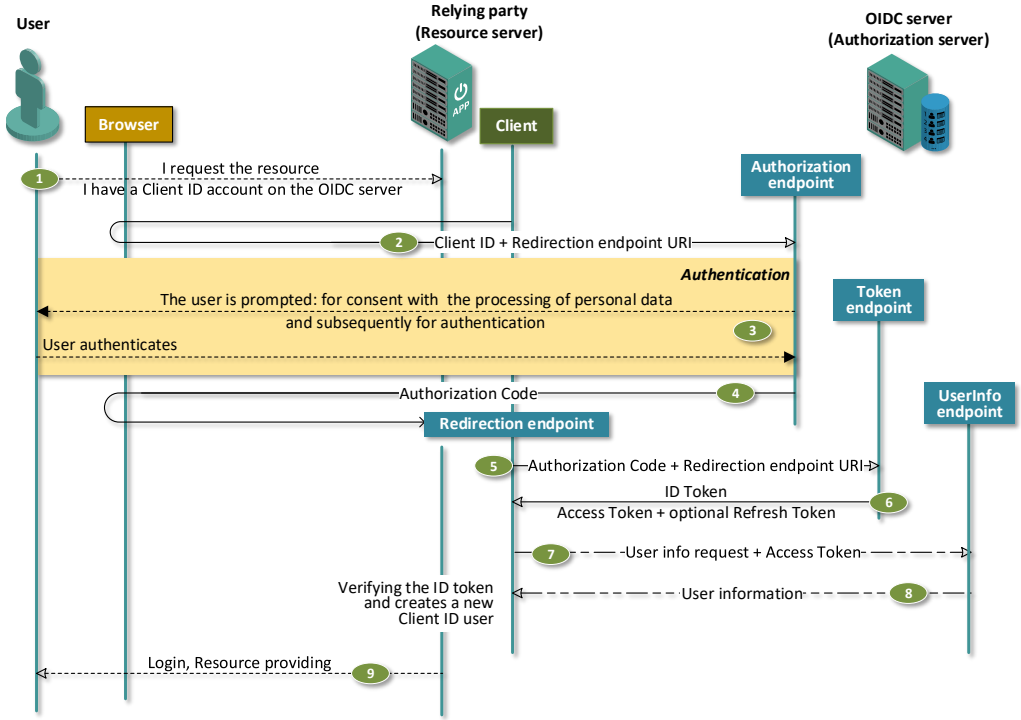



Figure 10.4 *Open ID Connect*

6. The OIDC server issues an ID token, Access token and optionally Refresh token.
7. Now (optionally) the relying party can request information about the user based on the Access Token. Usually, this happens when the user logs in to the Service Provider for the first time.
8. If the client has consented, the Relying party will receive information about the user (the user could only consent with some personal data).
9. The Relying party can create a new user, which is identified by the name of the OIDC provider and the client ID registered on the OIDC server. Subsequently, the Service Provider will provide the resource.

## 10.7 Identity management

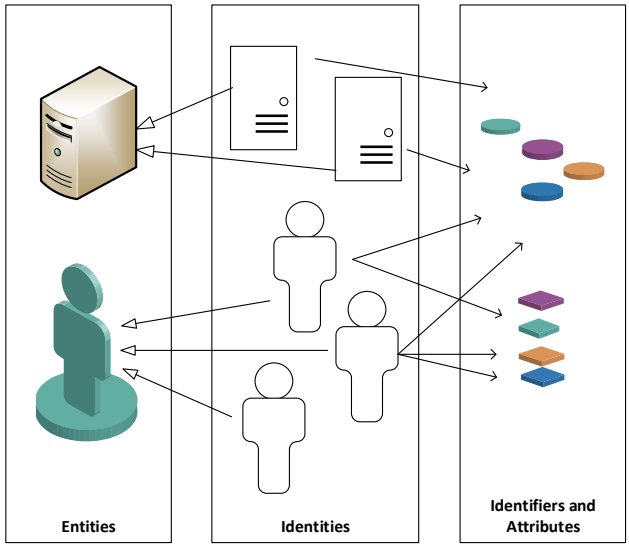



Figure 10.5 *Entities and identities*

An **entity** (person, computer, building, network, etc.) can have multiple **identities** (administrator, user, external user, etc.) from an IT perspective.

Each identity is assigned one or more identifiers (e.g., email address) and attributes (e.g., department name).

### 10.7.1 RBAC

The Role-Based Access Control (RBAC) model (Figure 10.6) assumes that an entity has one or more identities within an organization (Figure 10.5). Each of its identities has specific attributes (contact details, passwords, public key certificates, etc.). But it is important that specific authorizations for access to assets are not directly tied to identity, but to roles, i.e., identities are assigned roles, and only roles are assigned permissions to assets.

The roles should correspond to the positions in the company (project manager, accountant, etc.). Ideally, all identities with the same role had the same permissions within the company. However, this is unrealistic in practice, which is why roles are created only for groups of positions. However, within the organization there may also be roles common to all employees (e.g., email role or physical access to specific place role, etc.)

When the role is changed, the permissions are automatically changed. This is of absolutely fundamental importance when changing jobs position and especially when terminating an employment relationship. In such a case, the automated removal of roles will prevent abuse of the permissions that should have been removed and the removal was forgotten.

If we have an asset (e.g., a file in the server's operating system), then if access is according to the RBAC model, specific roles are assigned to individual users (in the case of an operating system, roles are implemented by group membership). Rules are then defined in the operating system which groups have access to specific files (assets) in the operating system.

Each role contains:

- Attributes (e.g., role description).
- Rules that represent a set of permissions.
- Explicitly enabled/disabled any other permissions.

Basic rules of the RBAC model:

- For each role, resources must be defined and access to them must be provided for a typical user of that role.
- Each user must have at least one defined role and permissions for that role.
- If user have multiple roles, they have to mutually exclusive. No user can have assigned roles that contradict each other.

The RBAC model is not the only one. An alternative is, for example, Attribute-based access control (ABAC).

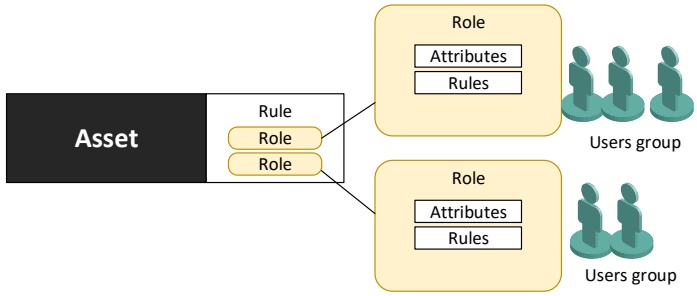



Figure 10.6 *RBAC*

### 10.7.2 Identity Management (information system)

Identity Management (IdM) is an information system that manages the life cycle of identities in a company/organization.

If an employee starts work, then he will be registered as an entity in the Human Resource System (HR System), as a student in the Student registration system, etc. These systems are superior to the Identity management system (Figure 10.7). These systems establish entities and some of their roles and some of their attributes.

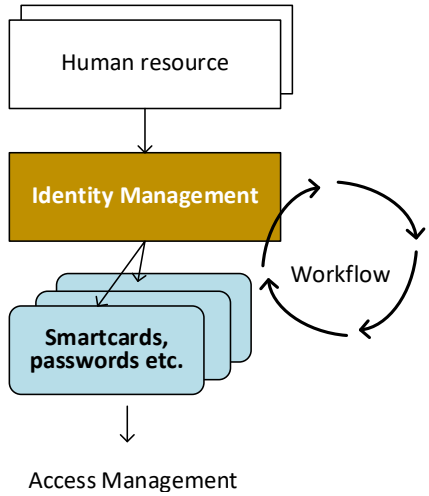



Figure 10.7 *Employee registration*

But it also changes the properties of entities (change of job position) or deletes entities (termination of employment). Some roles will be created directly when the employee joins (e.g., role: e-mail, access to buildings, etc.).

New roles or changes in existing roles are transferred from the HR system to Identity management. The classic type of transfer is the transfer using a text file in the LDIF format.

Within Identity management, it is possible to request the assignment of additional roles and attributes. Approval is done using a workflow that involves, for example, a direct supervisor employee or in addition, a security manager, etc. for selected roles.

Allocation of authentication means (e.g., smart cards, passwords, etc.) is possible already at the level of the HR system, but also at the level of Identity management with the help of roles. The choice depends on the specific implementation.

Once employees, their roles and authentication means are in place, then they can access individual systems according to their authorizations, as Identity management has in the meantime transferred roles and authorizations to access control systems, which may be a Kerberos Distribution Center (KDC), an IdP, an Authorization Server or even individual systems, into which users log in directly, e.g., building management systems, etc.

The basic lesson is that identities and roles should only be transferred from Identity Management to subordinate systems. As soon as the administrators of subordinate systems are allowed to add/change users and their roles, it is necessary to propagate these changes back to the Identity System, which is non-trivial, because conflicts may arise that must be solved by authorized persons, which is not only complicated, but also costly. Without it, however, the Identity management system within the company/organization will begin to fail over time. Therefore, it is easier to limit the rights of administrators.

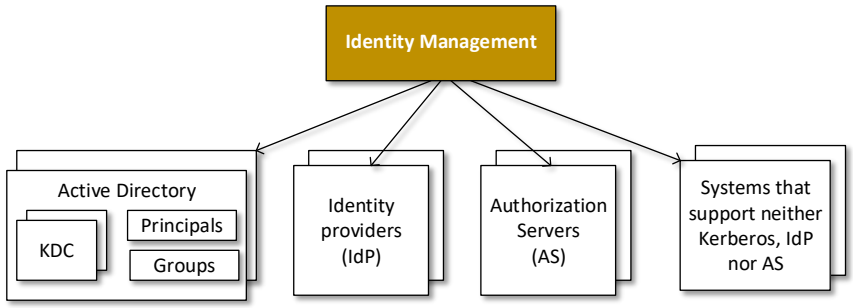



Figure 10.8 *Identity Management is superior to individual access control systems*

# 11 QUIC

The QUIC was experimentally developed by Google, later modified and released in May 2021 as RFC 9000 [69]. It is a protocol that tries to replace the TCP protocol. Some refer to it as TCP/2, which is certainly not the official name.

Table 11.1 *QUIC terminology*

| **Term** | Meaning |
|---|---|
| **Endpoint** | An entity that can participate in a QUIC connection by generating, receiving, and processing QUIC packets. There are only two types of endpoints in QUIC: client and server |
| **Client** | The endpoint that initiates a QUIC connection |
| **Server** | The endpoint that accepts a QUIC connection |
| **Peer** | Client or Server |
| **Connection ID** | An identifier that is used to identify a QUIC connection at an endpoint. Each endpoint selects one or more connection IDs for its peer to include in packets sent towards the endpoint. This value is opaque to the peer |
| **Stream** | Unidirectional or bidirectional channel of ordered bytes within a QUIC connection. A QUIC connection can carry multiple simultaneous streams |
| **Key exchange** | Key exchange in the sense of the ECDH algorithm |

The TCP, proposed in 1974 [70], was at the birth of the Internet. It is still used with modifications [71]. Over time, next protocols were proposed to expand or replace TCP. It is worth mentioning, for example, the T/TCP protocol [72]. Only the SCTP [73] and QUIC [69] achieve success. While SCTP has its source in telecommunications [74] (the SS7 protocols family), QUIC was developed for web technology. While the goal of SCTP is to transfer multimedia streams (”telephone calls”) in parallel for a long time, QUIC is oriented towards the maximum speed of web page objects downloaded in parallel. Both protocols emphasize parallel transmission in multiple streams, which is a fundamental difference from TCP.

QUIC uses UDP as the transport protocol. So, from the operating system point of view, it is actually an application protocol, but from the point of the network architecture view, it is a transport protocol.

With the help of connection migration, the **client can change its IP address during the connection**. For example, in the case of a mobile phone, it can switch from Wi-Fi to mobile

network without breaking the connection at the QUIC protocol level.

Another difference compared to TCP is that the QUIC protocol packets may carry application data from the first connection establishment packets (Figure 11.1). In case of session resumption, two packets will also be saved (Figure 11.2). This is how it is described simply in presentations on the Internet. If, for example, you view the network communication with the program Wireshark, then you will see a significantly larger number of UDP datagrams. There are several reasons, let us name just a few:

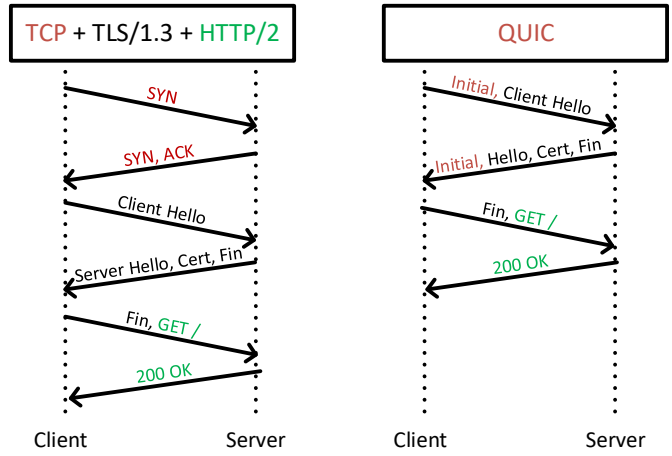



Figure 11.1 *Initial handshake of TLS and QUIC protocols*

- The client can send application protocol data (e.g., HTTP/3) already during connection handshake in so-called **0-RTT packets**. This weakly secured data is referred to as **"early data"** which is weakly secured especially against replay attacks; therefore, this data should not contain information that could trigger any action caused by this type of attack.
- QUIC assumes a minimum IP packet size of at least 1280 bytes (with correspondents to the minimum IPv6 size and is also supported by most IPv4 networks). If the peers do not agree on a larger MTU, then the size of IP datagrams will not exceed 1280 bytes.
- UDP datagrams containing Initial packets have UDP payloads of at least 1200 bytes. If necessary, PADDING frames are added. In this way, QUIC tests whether the network path supports an MTU of at least 1200, since QUIC does not support a MTU less than 1200.
- For example, the HTTP/3 application protocol first establishes the connection parameters between the client and the server. Only then the streams are created that carry application data. So, the first frame of the HTTP/3 protocol will not contain the method (e.g., GET method).

In the case of the QUIC protocol, we do not especially appreciate the reduction of transmitted packets, but in particular:

- Elimination of shortcomings of the HTTP/2 protocol.
- The fact that it is a general multi-streaming transport protocol that can be used by other application protocols.

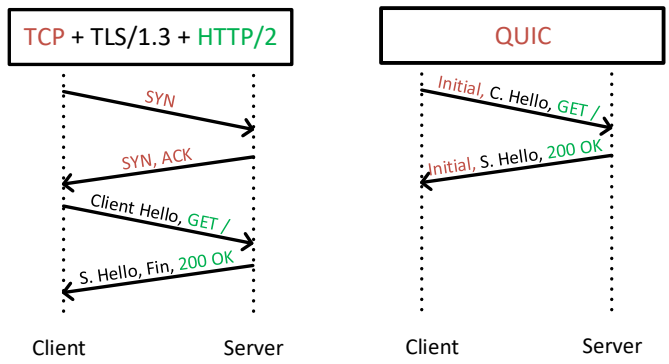



Figure 11.2 *Session resumption of TLS and QUIC protocols*

## 11.1 Connection, packet, frame and stream

QUIC is a connection-oriented protocol which uses client-server architecture. The QUIC client establishes a connection (similar to the TCP protocol). Unlike the TCP protocol, it allows communication in multiple streams.

A connection is established between the client and the server. The client and server identify this connection using the **Connection ID** value, which is different for the client side and for the server side. At the beginning of communication, the client generates a random Connection ID for the server side and puts it in the **Destination Connection ID** field. The server then repeats it in the **Source Connection ID** field. The Connection ID can also be of zero length if the peer can be unequivocally identified using the IP address and UDP port, which is practical in the case of the client, but not in the case of the server.

The use of connection IDs allows connections to migrate to a new network path.

QUIC uses the **TLSv1.3 protocol** to secure its packets, which is directly embed into QUIC. QUIC performs packet-level security, which means that QUIC always secures the entire QUIC packet, not individual frames. Any acknowledgement of received packets by the peer is also carried out at the packet level (not frames level).

The application protocols (e.g., HTTP/3) and TLS handshake packets QUIC ”are cut” into QUIC frames. One or more QUIC frames are transmitted in QUIC packet. QUIC packets are then encapsulated into UDP datagrams.

STREAM-type frames are used to transfer application communications. QUIC can carry multiple communication (streams) in parallel. QUIC uses a separate **QUIC stream** for each of these communications. Specific stream is for TLS handshake, which uses CRYPTO frames instead of STREAM frames.

A stream is only an abstraction since communication takes place using QUIC frames. One of the frame types is a STREAM frame that carries application data. When we imagine the line in the content of STREAM frames belonging to a particular stream, then we get this particular stream.

While QUIC frames are transmitted without any order, QUIC packets are numbered (from 1) and QUIC ensures their delivery. Furthermore, QUIC ensures that data is delivered within the stream in the order in which it was sent.

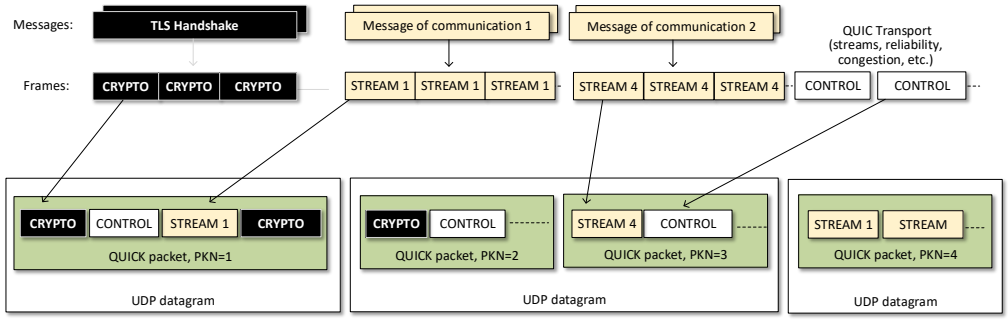



Figure 11.3 *QUIC Frames and Packets*

## 11.2 QUIC packets

The QUIC protocol uses two types of packets:

- Long header packets that are used before protocol TLSv1.3 establish 1-RTT keys (in Initial, 0-RTT, Handshake and Retry packets).
- Short header packets that are used once 1-RTT keys are established.

**Note:** In computer networks, RTT (Round Trip Time) is generally the total time it takes for the request to travel over the network and for the response to travel back. However sometimes it means number of communication rounds between two sides (in this case it is generally called $x$-RTT, where $x$ means number of full communication rounds). For example 1-RTT means there was one communication round (client sent message to server and server responded).

The QUIC uses RTT in the following way:

- 0-RTT for packets carrying information that do not need to be acknowledged.
- 1-RTT for packets carrying information that need to be acknowledged (e.g., by ACK frames), so it is a round trip.

### 11.2.1 Security of QUIC packets

Important part of the QUIC protocol is security. QUIC defines its own security measures for QUIC packets. It uses parts of TLSv1.3 protocol, specifically only the HP (Handshake Protocol) and AP (Alert Protocol) protocols. Conversely, the RLP (Record Layer Protocol) is not used, i.e., the negotiation of cryptographic handshake is used, but QUIC secures its packets itself. The mechanism is specified by RFC 9001 [75], see Figure 11.4.

QUIC implements two phases of security:

- At the beginning of the communication, the cryptographic material is derived from the random Destination Connection ID generated by the client and uses the constant value

of salt defined in the standard, i.e., it is easy for a knowledgeable attacker to access such secured data. This cryptographic material is called 0-RTT keys. Application data secured by this weak 0-RTT keys is called ”early data”.

- After the agreement on the algorithms and keys by the HP protocol, these agreed so-called 1-RTT keys are used.

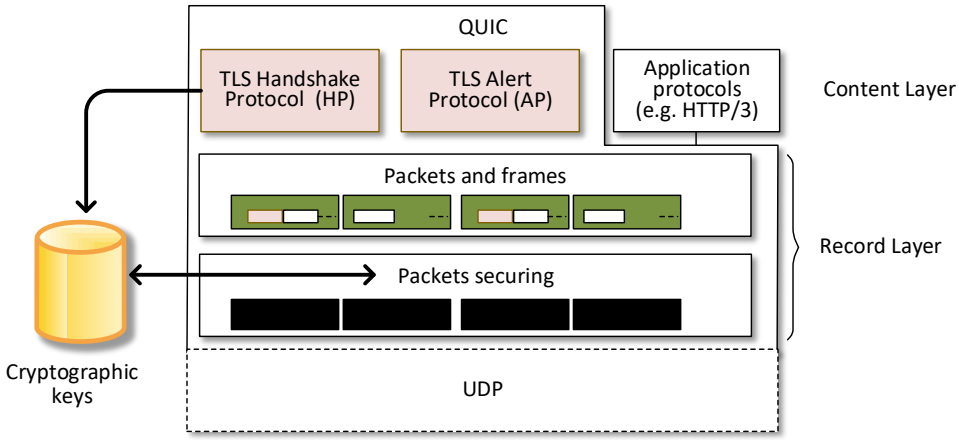



Figure 11.4 *Security of QUIC packets*

During the connection, 1-RTT keys can be renewed. There is something remarkable about it. When a packet is lost, QUIC requests resending of the packet. However, if the keys were renewed in the meantime, then the repeated packet must be secured with the ”old” keys.

### 11.2.2 Long header packets

Packets with a long header (Figure 11.5) generally have the following content:

- The most significant bit of the first byte is set to 1 for long header packets. This bit indicates a long header packet.
- Next bit is fixed and is always set to 1.
- Long packet Type field indicates the packet type (Initial, 0-RTT, Handshake, or Retry packet type).
- Reserved Bits (R) contain two bits reserved across multiple packet types. The value included before protection must be set to 0.
- Length (L) contains the length of the remaining portion of the packet in bytes, encoded as a variable-length integer.
- The Version is set to 1 in this version of QUIC.
- Destination Connection ID Length contains the length of the following field.
- Destination Connection ID contains the Connection ID of the peer.

- Source Connection ID Length contains the length of the following field.
- Source Connection ID contains the Connection ID of the sender.
- Packet type-dependent content is specific for each long header packet type.
- The Payload contains frames in any order (before packet securing).

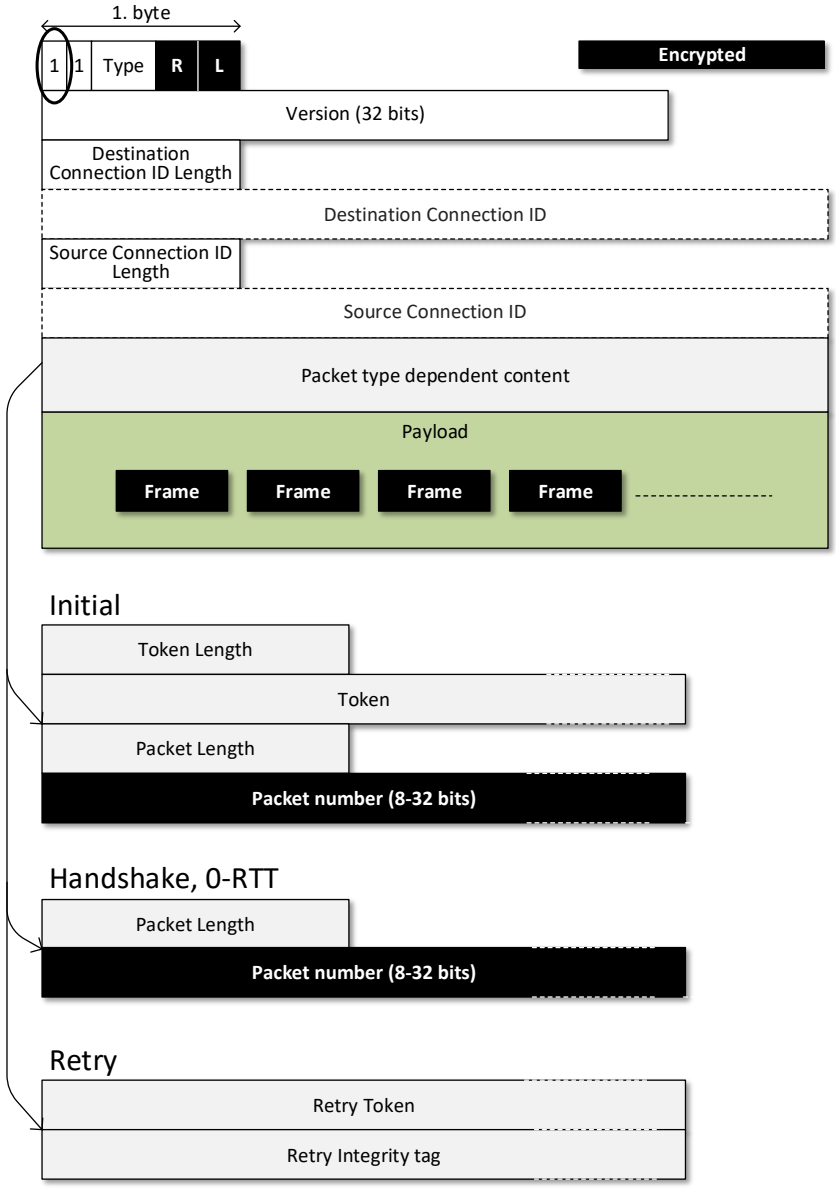



Figure 11.5 *Long header packet*

We use the following terminology for long header packets:

An **Initial packets** carries the first CRYPTO frames sent by both the client and server to perform a key exchange. It also carries ACK frames that acknowledge packets received from the

peer. To meet protocol requirements, Initial packets commonly include PADDING or PING frames to ensure the packet reaches the minimum required length — the MTU for Initial packets must be at least **1200 bytes**.

A **0-RTT packet** carries early data from the client to the server as part of the first flight of packets, i.e., before the handshake is completed. This allows data to be sent with minimal latency, but with limited cryptographic assurance.

**Handshake packets** facilitate the agreement on cryptographic parameters between client and server. In addition to CRYPTO frames, they also contain ACK frames that acknowledge received packets

QUIC uses a **variable-length encoding** for non-negative integer values. QUIC's variable-length integer encoding uses the two most significant bits of the first byte of the integer encoding length in bytes. The integer value is encoded on the remaining bits, in network byte order:

| Two most significant bits | Length | Usable bits | Range |
|---|---|---|---|
| 00 | 1 | 6 | 0-63 |
| 01 | 2 | 14 | 0-16383 |
| 10 | 4 | 30 | 0-1073741823 |
| 11 | 8 | 62 | 0-4611686018427387903 |

### 11.2.3 Short header packets

Short header packets are used only after the cryptographic material has been negotiated and thus are secured by the negotiated cryptographic material. Short header packets are **1-RTT packets** (Figure 11.6).

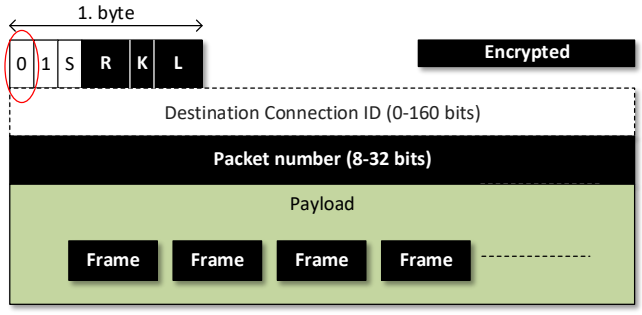



Figure 11.6 *Short header packet*

1-RTT packets contain:

- The most significant bit of **bit 0** is set to 0 to **indicate a Short header packet**.
- Next bit is fixed and is always set to 1.
- **The Spin (S)** bit is used for passive RTT measurement (if the received packet increases the highest number of the acknowledged packet, then the content of the spin bit from the received packet is copied into the response). Importantly, this bit is not encrypted,

so network devices on the network path can see it.

- Reserved bits (R) contain two bits reserved across multiple packet types. The value included before protection must be set to 0.
- The **Key Phase bit** (K, in Wireshark KP) indicate Key Phase, allowing a recipient of a packet to identify the packet protection keys used to protect the packet. An endpoint that notices a changed Key Phase bit updates keys and decrypts the packet containing the changed value (this mechanism replaces the key update mechanism of TLS).
- The **Length (L)** of the Packet number in bytes (value 0 to 3). The value is one less than the length of the Packet Number field in bytes (length 1 to 4).
- The **Destination Connection ID** contains the Connection ID of the peer.
- Packet number. This number is acknowledged in ACK frames.

## 11.3 QUIC Frames

QUIC frames starting with the frame type followed by content that depends on the frame type.

### 11.3.1 CRYPTO and STREAM frame types

A stream is an abstraction created by concatenating the contents of individual STREAM frames of the same stream. A stream starts by sending its first frame.

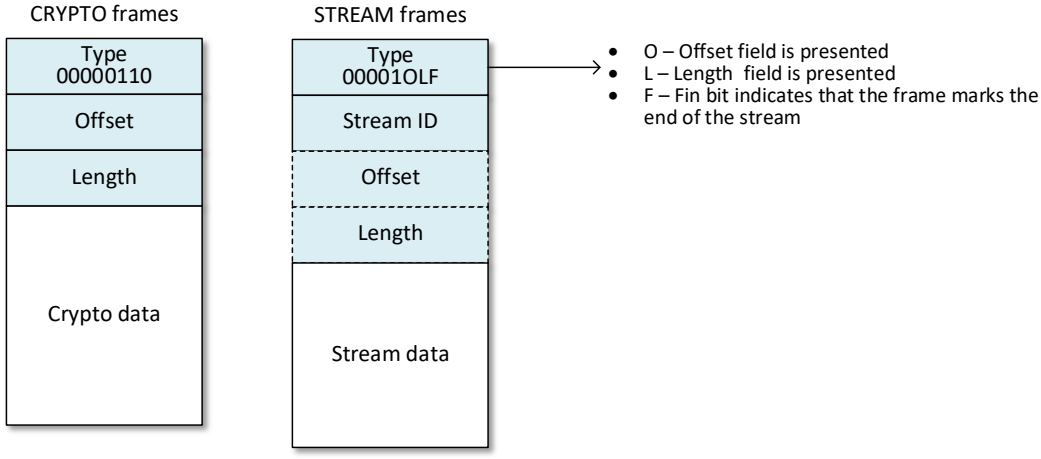



Figure 11.7 *CRYPTO and STREAM frames*

Streams carrying application data use STREAM type frames, while streams carrying cryptographic handshakes (TLSv1.3) use CRYPTO type frames.

- STREAM frames contain **Stream ID**, which is variable-length integer indicating the ID of the stream. The last two bits of Stream ID indicate the type of the stream:

Table 11.2 *QUIC frames*

| Frame | Meaning |
|---|---|
| PADDING | PADDING frames can be used to increase the size of a packet. |
| PING | Endpoints can use PING frames to verify that their peers are still alive or to check reachability to the peer. In practice, however, it is also used as a short padding. |
| ACK | Receiver sends ACK frames to inform the senders of the packets they have received and processed. The packet number is acknowledged - not the received bytes, as in the case of TCP. |
| RESET_STREAM | An endpoint is used to abruptly terminate the sending part of a stream. |
| STOP_SENDING | An endpoint informs that incoming data is being discarded on receipt per application request. |
| CRYPTO | A CRYPTO frame is used to transmit cryptographic handshake messages. |
| NEW_TOKEN | A server sends this frame to provide the client with a token to send in the header of an Initial packet for a future connection. |
| STREAM | Implicitly creates a stream and carries stream data. |
| MAX_DATA | This frame is used in flow control to inform the peer of the maximum amount of data that can be sent on the connection as a whole. |
| MAX_STREAM_DATA | This frame is used in flow control to inform a peer of the maximum amount of data that can be sent on a stream. |
| MAX_STREAMS | This frame informs the peer of the cumulative number of streams of a given type it is permitted to open |
| DATA_BLOCKED | A sender sends this frame when it wishes to send data but is unable to do so due to the connection-level flow control. |
| STREAM_DATA_BLOCKED | A sender sends this frame when it wishes to send data but is unable to do so due to the stream-level flow control. |
| STREAMS_BLOCKED | A sender sends this frame when it wishes to open a stream but is unable to do so due to the maximum stream limit set by its peer. |
| NEW_CONNECTION_ID | An endpoint sends this frame to provide its peer with alternative connection IDs that can be used to break link ability when migrating connections. |
| RETIRE_CONNECTION_ID | An endpoint sends this frame to indicate that it will no longer use a connection ID that was issued by its peer. |
| PATH_CHALLENGE | Endpoints can use these frames to check reachability to the peer and for path validation during connection migration. |
| PATH_RESPONSE | This frame is sent in response to a PATH_CHALLENGE frame. |
| CONNECTION_CLOSE | An endpoint sends this frame to notify its peer that the connection is being closed. |
| HANDSHAKE_DONE | The server uses this frame to signal confirmation of the handshake to the client. |

| Last two bits of Stream ID | Stream Type |
|---|---|
| 0x00 | Client-initiated, Bidirectional |
| 0x01 | Server-initiated, Bidirectional |
| 0x02 | Client-initiated, Unidirectional |
| 0x03 | Server-initiated, Unidirectional |

- The **offset** is a variable-length integer that specifies byte offset for the data in this STREAM frame. This field is present when the O bit is set to 1. When the Offset field is absent, the offset is 0.
- **Length** is a variable-length integer specifying the length of the Stream Data field in this STREAM frame. This field is present when the L bit is set to 1. When the L bit is set to 0, the Stream Data field consumes all the remaining bytes in the packet.

### 11.3.2 PADDING and PING type frames

One-byte length PADDING frame can be used to increase the size of a packet. They are used, for example, in INITIAL packets to increase the packet to the minimum required size.

One-byte length PING frame can be used to keep a connection alive when an application wishes to prevent the connection from timing out.


Figure 11.8 *PADDING and PING type frames*

### 11.3.3 ACK type frames

With the help of ACK frames receivers inform senders of packets they have received and processed them. If the frame type is 11 (binary), the ACK frames also contain the cumulative count of QUIC packets with associated ECN marks received on the connection up to this point (see Explicit Congestion Notification).

Meaning of individual frame fields:

- **Largest ACK** contains the largest packet number that the peer has received before generating the ACK frame.
- **ACK Delay** represents the time elapsed between receiving the packet with the largest packet number and sending the acknowledgment.
- **ACK Range Count** specifies the number of ACK Range fields in the frame.
- **First ACK** Range indicates the number of contiguous packets preceding the Largest ACK that are being acknowledged.
- Optional **ACK Range** denotes the range of packet gaps, representing contiguous unacknowledged packets.

- **ECN Counts** consist of three integers:
  - Total number of IP datagrams received with the ECT(0) code point.
  - Total number of IP datagrams received with the ECT(1) code point.
  - Total number of IP datagrams received with the ECN-CE code point.

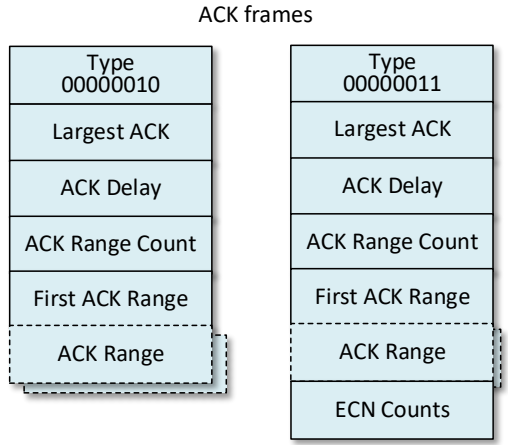



Figure 11.9 *ACK type frames*

## 11.4 Explicit Congestion Notification

IP networks signal congestion by dropping IP datagrams. Similarly, as in TCP, Explicit Congestion Notification (ECN) specified in RFC-3168 can be used. ECN allows end-to-end notification of congestion without dropping packets. The problem is that congestion is caused by the IP layer, but the solution to congestion is handled by the transport layer (TCP or QUIC).

The Explicit Congestion Notification uses the two least significant bits of the Traffic Class field in the IPv4 or IPv6 header to encode four different code points:

- `00` (binary) – Not ECN-Capable Transport, referred as **Not-ECT**.
- `01` (binary) – ECN Capable Transport 1, referred as **ECT(1)**.
- `10` (binary) – ECN Capable Transport 0, referred as **ECT(0)**.
- `11` (binary) – Congestion Experienced, referred as **ECN-CE**.

When both peers support ECN, they mark their IP datagrams with ECT(0) or ECT(1). If routers on the network path need to signal congestion, they change ECT(0) or ECT(1) to ECN-CE. An ECN-supporting QUIC receiver signals congestion to the sender in ACK frames using the ECN Counts field.

## 11.5 Address validation

Address validation is used to verify that the client does not send a packet to a server from a spoofed source address. It could potentially be an amplification attack, especially if a server generates more or larger packets in response to that packet, the attacker can use the server to send more data than it would be able to send on its own.

QUIC uses two address validation mechanisms:

- Address Validation Using Retry Packets.
- Address Validation for Future Connections.

### 11.5.1 Address Validation Using Retry Packets

This is a mechanism similar to the **Photuris mechanism** (Section section 3.3 Photuris) – only the term Token is used instead of the term Cookie. For example, when establishing the first connection, it may (but may not) verify that the client's IP address is not spoofed. The idea is that the server sends a packet to the client containing a string called a token and waits to see if the client responds correctly.

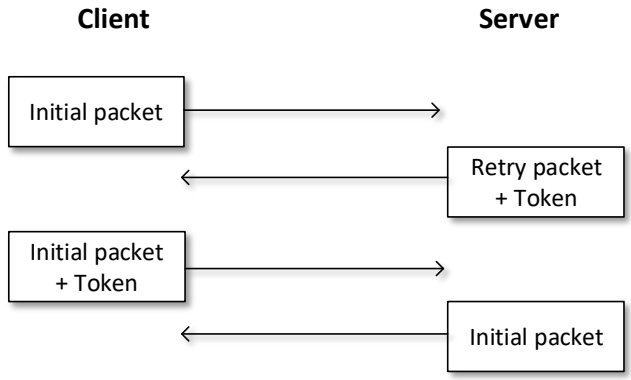



Figure 11.10 *Address Validation Using Retry Packets*

Specifically (Figure 11.10), the client initiates a connection with an Initial packet and expects an Initial packet from the server in return. However, the server responds with a Retry packet, containing a random Token. The client must then resend its Initial packet, but this time including the Token. Only after receiving the Initial packet with the Token does the server respond with the Initial packet.

### 11.5.2 Address Validation for Next Connections

For future connections, the server can provide the client with a new token, which it sends within a NEW_TOKEN frame. The client then uses this token when establishing a new connection.

# 12 TLS

Transport Layer Security (TLS) is a protocol that is inserted between transport protocols (such as TCP) and application protocols to secure client-server communications. The TLS protocol provides:

- Authentication of communicating parties. The server is always authenticated, and while client authentication is optional, it does not mean that the client cannot authenticate at the higher layer protocol level.
- Confidentiality and integrity of transferred dat a with the help of AEAD type ciphers (see subsection 2.6.3 AEAD).

To understand TLS protocol versions, we need to mention the history. The predecessor of TLS was the Secure Sockets Layer (SSL) protocol developed by the now defunct Netscape Communication. Back then, it was established that a protocol version would consist of a major and minor version. The last version of the SSL protocol was version 3. The TLS protocol kept version 3 as a major version and embeds its versions in minor versions. The minor version then distinguishes the individual versions of TLS protocols, i.e., individual versions of the TLS protocol follow only after ”3.”.

So, do not be fooled by the fact that there is little minor version changes between TLSv1.1 and TLSv1.3. The differences are fundamental. If you were to read a textbook describing, for example, TLSv1.0 and this text-book describing TLSv1.3, you would feel that they are two different protocols.

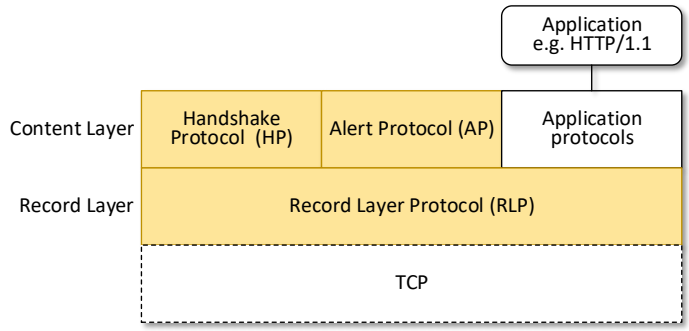



Figure 12.1 *TLS Architecture (using TCP protocol)*

TLS assumes that the transport protocol provides a reliable connection that detects trans-

mission errors and delivers packets in the order they were sent. A modification of the TLS protocol, called DTLS, is intended for datagram-oriented communication.

## 12.1 TLSv1.3 Architecture

The architecture of TLSv1.3 is different when the transport protocol is TCP and when the QUIC protocol is used:

- In the case of the **TCP protocol** (Figure 12.1), application data is inserted into packets called Records. Records and their security are specified by a protocol called the Record Layer (RL), which is part of the TLS protocol.
- In the case of the **QUIC protocol** (Figure 12.2), Record Layer is not used. QUIC (chapter 11 QUIC) secures its packets itself.

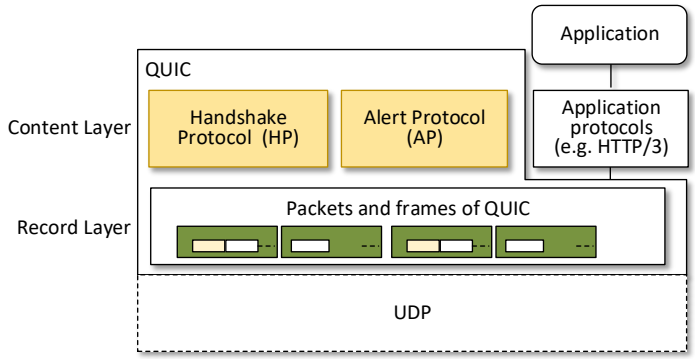



Figure 12.2 *TLS Architecture (using QUIC protocol)*

## 12.2 Perfect Forward Secrecy

If we use encryption keys for a long time, then an attacker could record the communication and try to break the cipher later. It might take him a long time, but he knows he will get to a lot of interesting data. **Perfect Forward Secrecy** (PFS) means that if an attacker broke some key, it would not help him with breaking subsequent keys. For each communication new set of ephemeral (temporary) keys are generated, so keys are not derived from the previous ones.

TLS allows temporary ECDH keys to be generated at the start of each connection as well as the ECDH public keys to be exchanged between the parties. Thus, for each connection, it is possible to generate a new shared secret from which new keys are derived that are independent of the previous connection.

TLS binds connections to the same TLS server into a single "**session**". If the cryptographic keys are derived only from the previous connection, then the connection does not satisfy PFS. If ECDH keys are additionally generated, then the connection complies with PFS.

**Note:** The word session is in quotation marks because TLSv1.3 does not explicitly introduce

the concept of a session, but uses it for the term “chain of contiguous connections”.

## 12.3 Handshake Protocol

The Handshake Protocol (HP) is the core of the TLS protocol. The HP protocol negotiates the security parameters of the connection. HP protocol messages are used for this negotiation, which are passed to TLS Record Layer (or QUIC frames). The initial negotiation goes through three stages:

- **Key Exchange**: establish shared keying material and select the cryptographic parameters. Everything after this phase is encrypted.
- **Server Parameters**: establish other handshake parameters.
- **Authentication**: authenticate the server (and, optionally, the client) and provide key confirmation and handshake integrity.

Application data can then be transferred.

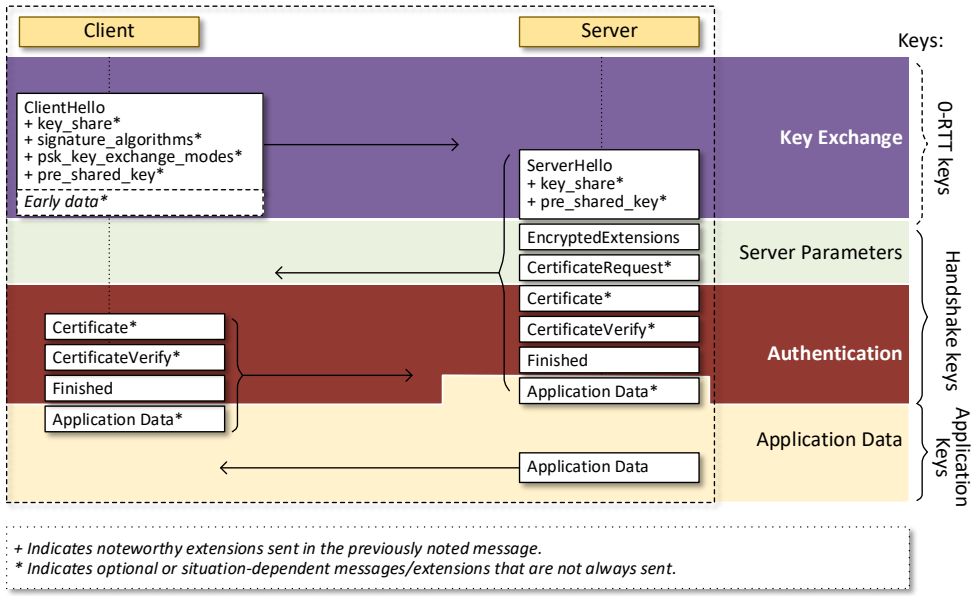



Figure 12.3 *HPv1.3*

### 12.3.1 Key Exchange phase

The Key Exchange phase is initiated by the client with the `ClientHello` message. In this message, the client offers the server a list of parameters supported by the client, and from them the server selects specific parameters and sends them in the `ServerHello` message. The key exchange phase is either not secured (e.g., the HTTP/2 protocol), or it is secured by

keys derived from the transmitted data (e.g., the HTTP/3 protocol), i.e., if it is secured, it is weakly secured.

`ClientHello` and `ServerHello` messages mainly include:

- `ClientHello.Random` and `ServerHello.Random` random numbers, which increase entropy when generating cryptographic material.
- ECDH public keys (or links to them).

After exchanging the ECDH public keys, both parties are able to calculate the shared secret, from which they derive symmetric keys with which they secure the following communication.

The 0-RTT data is interesting. This is application data that can be sent as part of `ClientHello` and `ServerHello` messages. They are called early data. They can only be transmitted if Pre-shared Secret Keys (PSK) are already established. This data is secured by keys derived from the previous connection (referred to as 0-RTT keys) and do not meet Perfect Forward Secrecy (PFS).

Used cryptographic material can be devided to 3 groups:

- **Early traffic key**: can be used to secure 0-RTT data.
- **Handshake keys**: secure the rest of handshake after `ClientHello` and `ServerHello` messages are sent.
- **Application Keys**: secure application data, and which are derived from Master Secret.

### 12.3.2 Server Parameters phase

This phase is already secured with the keys generated from the Key Exchange phase. The server sends additional selected connection parameters with the `EncryptedExtensions` message. Optionally, it can signal client to authenticate with a `CertificateRequest` message.

### 12.3.3 Authentication phase

Server authentication is mandatory; client authentication is optional. Authentication takes place using an electronic signature. The supported algorithms are: ECDSA, EdDSA, RSA-PSS and even RSASSA-PKCS1-v1_5, which is limited to the use of the SHA-2 family. As a rule, public key certificates are used, but it is also possible to use the keys themselves. Server authentication and optional client authentication proceed similarly:

1. By the `Certificate` message sends a certificate that authenticates the party. A certification path may be also sent.
2. The `CertificateVerify` message sends an electronic signature of the entire previous communication using private key that belong to certificate. The party authenticates the opposite party by verifying its signature.
3. The `Finished` message ends this particular HP protocol dialog (If the authentication is successful).

Then the Application Data exchange can start.

### 12.3.4 Application data

Application data is secured based on sets of cryptographic Protocol Suite agreed in previous handshake (application keys, from now on referred to as 1-RTT keys).

The Protocol Suite includes: AEAD protocol, key length, block cipher mode and hash algorithm. TLSv1.3, unlike TLSv1.2 and earlier, only supports a few cryptographic suites (Table 12.1).

Table 12.1 *Protocol suites supported by TLSv1.3*

| Protocol suite | Standard |
|---|---|
| **TLS_AES_128_GCM** | |
| **TLS_AES_256_GCM** | RFC 5116 [76] |
| **TLS_AES_128_CCM** | |
| **TLS_CHACHA20_POLY1305** | RFC 8439 [77] |
| **TLS_AES_128_CCM_8** | RFC 6655 [78] |

### 12.3.5 Cryptographic keys

Cryptographic keys are used for authentication and communication security. This paragraph deals only with keys for communication security.

TLSv1.3 supports three basic types of key exchange (the key exchange type is set by the client using the PSK Exchange Modes extension):

- ECDH key exchange.
- PSK-only mode.
- a combination of the previous two modes.

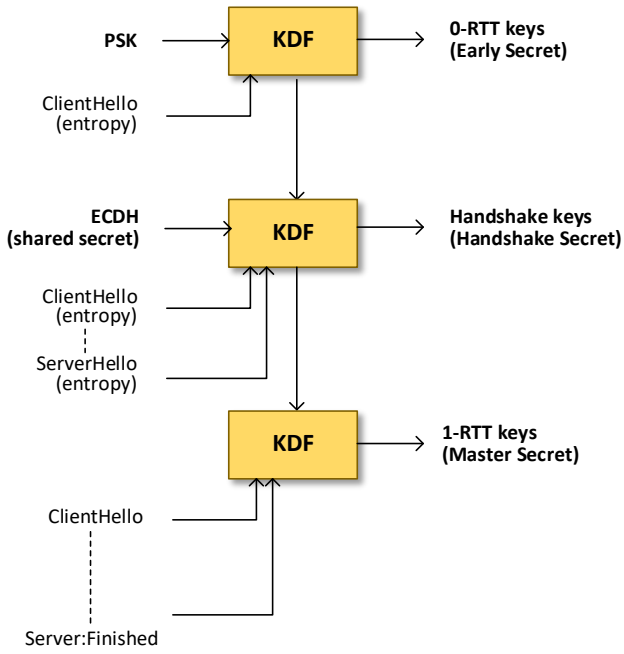



Figure 12.4 *Establishment of cryptographic keys*

**Pre-shared Secret Key (PSK)** is a shared secret that was previously shared between the two parties using some secure channel before it needs to be used. You can establish a PSK during one TLS handshake and then use it to establish a new connection in another handshake; this is called session resumption with a PSK. If they are not established, then there

must be an ECDH exchange at least at the beginning.

Assume that PSKs are established. The connection is established in 3 steps:

1. The **KDF (Key Derivation Function)** is called, which includes PSK (out of band or from a previous connection) and `ClientHello` with entropy in `ClientHello.Random`. The output is 0-RTT keys (Early Secret), which secure early data.
2. The input to the KDF is the output of the first step, the ECDH shared secret, and the content of the communication from the `ClientHello` message to the `ServerHello` message. The output is Handshake Keys, which form the so-called Handshake Secret.
3. The input to the KDF is the **Handshake Secret**, all communication from `ClientHello` to the `Finished` message. Cryptographic material for securing further communication (1-RTT keys) is subsequently derived from the Master Secret.

**Note:** If we set the environment variable $SSLKEYLOGFILE to the name of a directory for some browsers, then they will record the Master Secret of individual TLS connections in this directory. This will allow Wireshark to decrypt and display the contents of secure packets. It is also necessary to tell Wireshark the name of the specified directory (Edit – Preferences – Protocols – TLS – (Pre)-Master-Secret log filename).

### 12.3.6 Post Handshake authentication

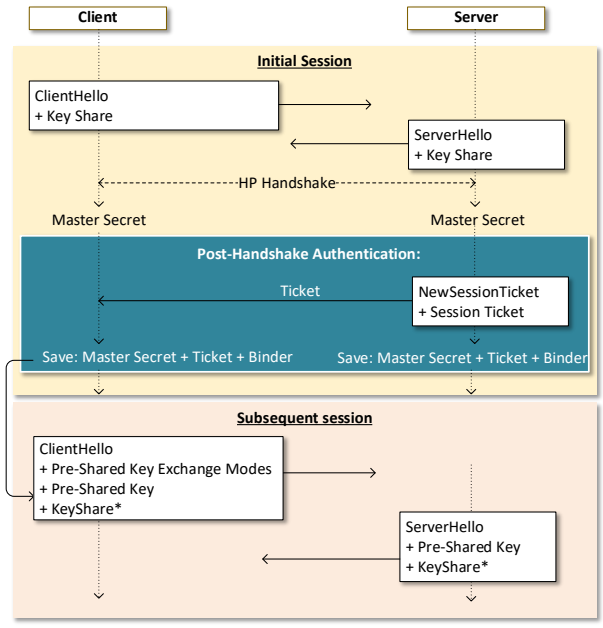



Figure 12.5 *Binding the previous connection to the subsequent connection*

Figure 12.3 illustrates the initial negotiation of cryptographic protocols and cryptographic material. However, the HP protocol can be activated at any time later. The so-called "Post-Handshake Authentication" is important. At any time after the server has received the client `Finished` message, it may send message `NewSessionTicket`.

This message creates a unique association between the ticket value and a secret PSK derived from the resumption master secret. The client MAY use this PSK for future handshakes by including the ticket value in the `pre_shared_key` extension in its `ClientHello`.

Figure 12.5 shows the establishment of the initial connection and the subsequent connection, but the mentioned mechanism is applied in general (not only after the initial connection).

As part of the initial connection, the client exchanges public ECDH keys with the server in the `key_share` extension, from which the Master Secret (i.e., 1-RTT keys) is ultimately derived.

As soon as the Master Secret is established, the server sends the `NewSessionTicket` message, which contains the Ticket. A Ticket is a structure that contains: identifier `ticket_nonce`, ticket value and the ticket's validity period. The identifier is usually a link to the database that maintains the Master Secret of each connection.

The client also keeps the Master Secret for the connection and adds the received Ticket to it. When establishing the subsequent connection, the parties no longer have to (but can) exchange public ECDH keys in the `key_share` extension. All they need to do is to exchange information about which Master Secret from previous connections will be used. Such a Master Secret constitutes a PSK.

The so-called PSK Binder is used for this, which binds the Master Secret and Ticket. The PSK Binder is derived using KDF from previous `ClientHello` messages. Using the PSK Binder, the client confirms the validity of the given Master Secret. Now we already have triplet: Master Secret – Ticket – PSK Binder.

When establishing a subsequent connection, the client sends a list of current triplets into the `pre_shared_key` extension. The server chooses one triplet and the master secret from the selected triplet is used as PSK on both sides of the connection.

### 12.3.7 ClientHello a ServerHello messages

The principle is that the Client usually offers a list of supported features and the Server chooses the one it considers the most secure. If the server is unable to select, then the connection is terminated by the controller with an Alert protocol message.

The format of these messages is based on the TLSv1.2 protocol to maintain backward compatibility. Their syntax is following:

- The **version** entry contains the TLSv1.2 version, i.e., the value 3.3 (TLSv1.0 version had the value 3.1).
- Next follows **random values** `ClientHello.Random` or ServerHello.Random.
- The **session ID** has a length of zero in HTTP/3, but HTTP/2 uses this field. Only the client can set this field, the server can only repeat it.
- **Protocol suites:** the client offers a list of suites – the server selects a suite.
- **Supported compression algorithms:** if this field is used, then in TLSv1.3 the list of supported compression algorithms must be "null".
- **Extensions** are the fundaments of TLSv1.3 (12.2). The entry contains a list of individual extensions.

Table 12.2 *TLSv1.3 Extension*

| Frame | Description |
|---|---|
| server_name | DNS name of TLS (HTTPS) server. |
| status_request | The client requests the TLS server to send it the status of the server's certificate (e.g., OCSP response). In this extension, the OCSP responder identification is present. |
| supported_groups | Supported parameters of elliptic curve algorithms. |
| padding | Padding of message. |
| signature_algorithm | Signature algorithm in `CertificateVerify` messages. |
| signature_algorithms_cert | Certificate signing algorithm. |
| application_layer_protocol_negotiation | The ALPN extension makes it possible to agree on an application layer protocol. The client sends a list of supported application protocols (e.g., h2 and h3) and the server selects from them. |
| supported_versions | The client sends a list of supported versions and the server:<br>• If he wants to communicate with a lower version, then he does not specify this extension, but sets the TLS Version 1.2 Items.<br>• If he wants to communicate TLSv1.3, then he confirms this version with this extension. If this extension is not used in `ClientHello`, then communication is done using the TLSv1.2 protocol. |
| cookie | Two primary purposes:<br>• Verification if it is not an attack from a spoofed IP address (more important for unconnected transmission, e.g., DTLS - see chapter 13 DTLS).<br>• Allowing the server to offload state to the client, thus allowing it to send a `HelloRetryRequest` without storing any state. The server can do this by storing the hash of the `ClientHello` in the `HelloRetryRequest` cookie (protected with some suitable integrity protection algorithm). |
| certificate_authorities | This extension contains a list of supported CA certificates and their certificate subjects. |
| oid_filters | With this extension, the server tells what extensions the client's certificate can have. The server sends a list of extension OIDs. |
| post_handshake_auth | With this extension, the client declares that it allows authentication. During the Post Handshake authentication, the server must not send a `CertificateRequest` to clients that do not offer this extension. |
| key_share | Contains the ECDH public key of the endpoint. |

| Frame | Description |
|---|---|
| psk_key_exchange_modes | Sets the key exchange type:<br>• ECDH key exchange.<br>• PSK-only mode.<br>• A combination of the previous two options. |
| pre_shared_key | The client will offer a list of triplets: Master Secret – Ticket – PSK Binder. The server confirms one triplet. |
| early_data | If the client inserts a 0-RTT packet after the `ClientHello` message, then it must signal this in this extension. |
| quic_transport_parameters | Only in case of QUIC protocol. QUIC specifies a number of connection parameters. For details, see chap. 18 of the RFC 9000 standard [69]. |
| GREASE | GREASE (*Generate Random Extensions And Sustain Extensibility*) is a non-existent extension. It is randomly generated and ignored by the other party. Poorly written software (e.g., poorly written malicious code) may crash when receiving this extension. |

### 12.3.8 Finished message

With this message, both the client and the server terminate the HP protocol handshake.

The content of this message is the MAC from the entire Handshake. This message confirms the association of the party with the established cryptographic material.

## 12.4 AP

Using the **AP (*Alert Protocol*)**, a warning or an error can be signaled. TLSv1.3 has two types of signaling:

- **Warning**: the client shares information with the server that they want to terminate the connection. In this case, the connection is terminated in the normal way. For example, `closenotify`, where the party signals that it will not send any more messages over this connection.
- **Abort**: an error has occurred, then the connection is terminated immediately. A number of different errors can be reported, for example, `unexpectedmessage` (message in unexpected format), `handshakefailure` (handshake could not be completed), `certificate_revoked` (client tried to use already revoked certificate), etc.

## 12.5 RLP

**RLP (*Record Layer Protocol*)** is a packet transfer protocol when QUIC is not used. RLP uses the types of packets (records) listed in Table 12.3.

Table 12.3 *RLP packets*

| RLP packet type | Description |
|---|---|
| `invalid` | Invalid packet |
| `change_cipher_spec` | Packet of Change Cipher Specification (CCS) – only for compatibility reasons |
| `alert` | Packet of Alert Protocol |
| `handshake` | Packet of Handshake Protocol |
| `application_data` | Packet of application data |

### 12.5.1 CCS

Packet of **CCS (*Change Cipher Specification*)** has a single message of value 1. This message tells the peer that this message is already followed by data secured by the newly negotiated keys.

### 12.5.2 RLP packet

To maintain backward compatibility, an RLP TLSv1.3 protocol packet has the format of an RLP TLSv1.2 packet. Application data is first cut into fragments not exceeding $2^{14}$ bytes.

Now there are two possibilities:

- The unsecured fragment.
- The secured fragment.

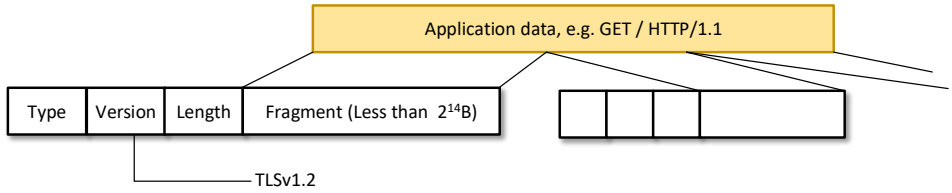



Figure 12.6 *The insecure fragment in the RLP protocol*

#### 12.5.2.1 The insecure fragment in the RLP protocol

In the case of an insecure fragment, the RLP packet is created by adding before each fragment payload (Figure 12.6):

- Record (packet) type (Table 12.2).
- The version is TLSv1.2, i.e., value 3.3, except for the first packet, which carries a value of 3.1 due to the backward compatibility of TLSv1.2 with TLSv1.0.
- Fragment length.

**Example** of a display of insecure fragments of the initial Handshake TLS by the Wireshark program:

*The first packet of the connection:*

```
TLSv1.3 Record Layer: Handshake Protocol: Client Hello
  Content Type: Handshake (22)
  Version: TLS 1.0 (0x0301)
  Length: 512
  Handshake Protocol: Client Hello
```

*Next packet:*

```
TLSv1.3 Record Layer: Handshake Protocol: Server Hello
  Content Type: Handshake (22)
  Version: TLS 1.2 (0x0303)
  Length: 122
  Handshake Protocol: Server Hello
```

#### 12.5.2.2 The secured fragment in the RLP protocol

In the case of a secure fragment, the fragment is secured by an agreed protocol set (Figure 12.7). The type (Table 12.6) and any padding are added to the end of the fragment so that the fragment can be encrypted with a block cipher.

The result is inserted into the RLP TLSv1.2 packet, where it is supplemented from the front:

- Type is always 23 (Application data).
- The version is TLSv1.2, i.e., value 3.3, except for the first packet, which carries the value 3.1 due to TLSv1.2 backward compatibilities with TLSv1.0.
- Length that must not exceed $2^{14}$ + 256 bytes.

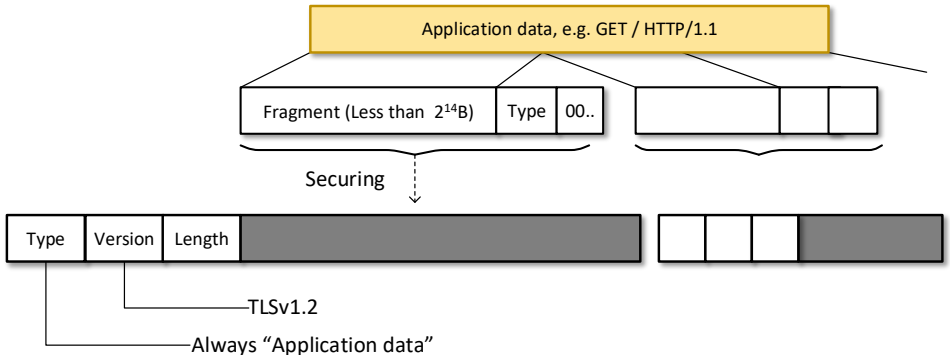



Figure 12.7 *The secured fragment in the RLP protocol*

**Example:** Lets look at the same RLP fragment header display of the same secure packet in the Wireshark with and without packet decryption:

1. Packet decryption is not set in Wireshark, i.e., the $SSLKEYLOGFILE environment variable is not set. As a result, the HP protocol packet appears as an Application Data packet:

```
TLSv1.3 Record Layer: Application Data Protocol: Hypertext
Transfer Protocol
  Opaque Type: Application Data (23)
  Version: TLS 1.2 (0x0303)
  Length: 2985
  Encrypted Application Data: 0268069f8d589a15ad0fa7…
  [Application Data Protocol: Hypertext Transfer Protocol]
```

2. Packet decryption is set in Wireshark, i.e., the $SSLKEYLOGFILE environment variable is set. As a result, an HP protocol packet appears as an HP packet:

```
TLSv1.3 Record Layer: Handshake Protocol: Certificate
  Opaque Type: Application Data (23)
  Version: TLS 1.2 (0x0303)
  Length: 2985
  [Content Type: Handshake (22)]
  Handshake Protocol: Certificate
```

# 13 DTLS

The TLS protocol is a very well-proven protocol. However, it has one drawback – it is not intended for securing datagram communication. This problem is addressed by the DTLS (*Datagram Transport Layer Security*) protocol [79], which modifies the TLS protocol (chapter 12 TLS) for datagram services. In particular, it must solve following issues:

- The packet can be lost, i.e., even in the HP protocol, the possible repetition of the `HelloRetryRequest` message, to which no response has been received, must be taken into account.

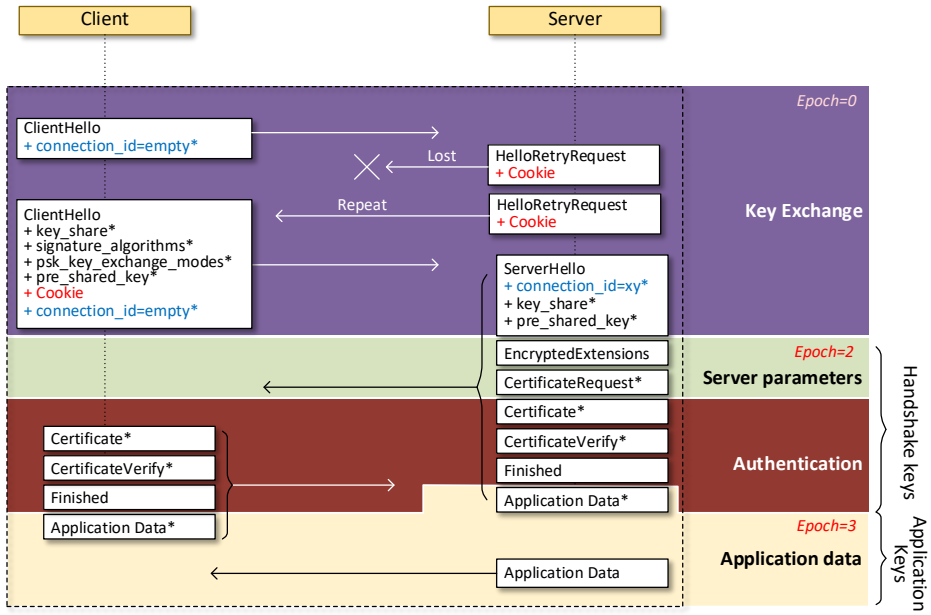



Figure 13.1 *Initial dialogue by the HP protocol in DTLSv1.3 (epoch 1 is skipped if the client does not offer early data)*

- TCP and QUIC transfers packets in the order in which they were sent, which UDP cannot provide. DTLS therefore numbers packets and especially introduces the term **Epoch**. The epoch changes after the encryption keys are changed. With the change of an epoch, the numbering is reset and starts from beginning.

- In IoT, the device can go silent for a long time, and if NAT (Network Address Translation) is inserted between the client and the server, then the association between the client and the server can be lost, therefore DTLSv1.3 in RLP packets can explicitly state the session identification **Connection ID (CID)**.

## 13.1 Handshake protocol (HP)

Unlike TLS, the initial HP dialog must start with the server issuing a Cooke using a `Hello RetryRequest` message. Again, this is in the manner introduced by the Photuris protocol (see Section section 3.3 Photuris). Since this is a datagram transfer, we must assume that the `HelloRetryRequest` message can be lost, so the server must keep a time limit after which it may retry the message if it does not receive the second `ClientHello` message.

**Note:** The initial dialog with Cookie is also possible in the case of TLSv1.3.

## 13.2 DTLS Record layer protocol

DTLS HP protocol messages and application data are cut into fragments that are transmitted in RLP protocol frames, analogously to TLSv1.3. `ClientHello`, `ServerHello` and `Hello RetryRequest` messages are not secured, further communication is secured.

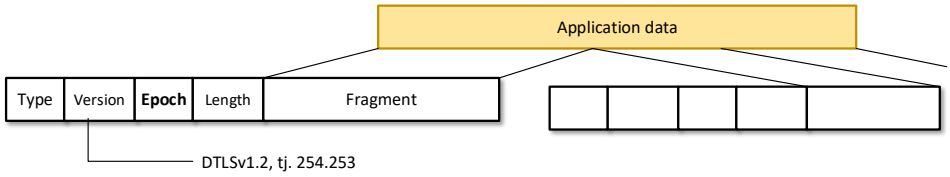



Figure 13.2 *Unsecured fragments of the DTLS RLP protocol*

We are talking about DTLS version 1.3, but RLP packets use version value of 254.253 to avoid confusion with TLS.

### 13.2.1 Unsecured frames

The unsecured fragments of the DTLS RLP protocol (Figure 13.2) are similar to the unsecured fragments of the TLSv1.3 protocol. The only difference is that the epoch is inserted into the header, which is 0 (unsecured messages).

### 13.2.2 Secured frames

The secure message format is fundamentally different from both TLSv1.3 and DTLSv1.2. Values Type and Padding are appended to the fragment. So far, it is similar to the TLSv1.3 protocol. However, after encryption of packet, only a Header entry is prepended to the secure fragment. The Header contains (Figure 13.3):

- 1 B Flags. The first 3 bits of Flags are constant. Bit C signals the presence of Connection ID (CID) entry. The S bit specifies the length of the Sequence Number (8 or 16 bits) and the two E bits are the last Epoch bits.
- Connection ID (CID).
- Sequence number of the fragment.
- Fragment length (similarly as in TLS).

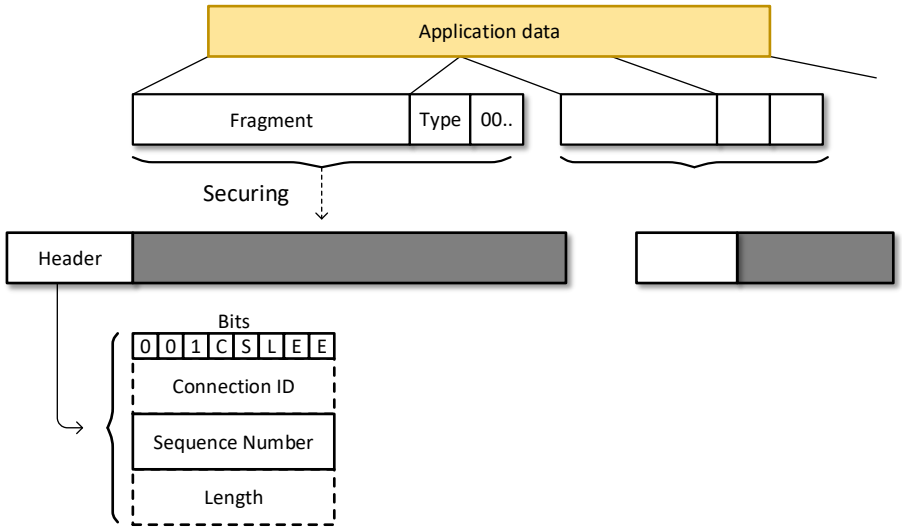



Figure 13.3 *Secured fragments of the DTLS RLP protocol*

# 14 SCTP

Stream Control Transmission Protocol (SCTP) [80] is reliable transport protocol from the TCP/IP protocol family. It differs from UDP [81] and TCP [70] protocols in many ways. SCTP [80] offers the following services to its users:

- **Acknowledged error-free, non-duplicated transfer** of user data.
- **Data fragmentation** to conform to discovered Path Maximum Transmission Unit (PMTU) size.
- **Sequenced delivery** of user messages within multiple streams, with an option for order-of-arrival delivery of individual user messages.
- **Optional bundling** of multiple user messages into a single SCTP packet.
- Network-level **fault tolerance** through supporting of multi-homing at either or both ends of an association.

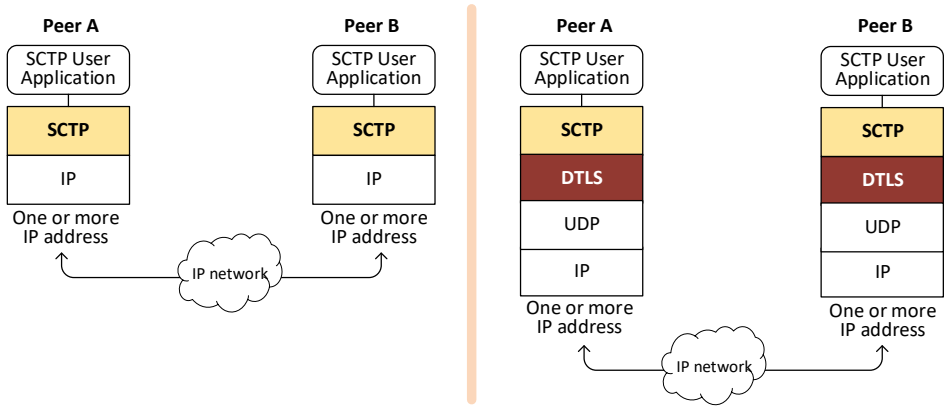



Figure 14.1 *Peer-to-peer architecture*

While the QUIC protocol (chapter 11 QUIC) was designed by web application developers to optimize the parallel download of web page objects, SCTP has its origins in telecommunications, i.e., for parallel transmission of multimedia sessions (”calls”).

While the UDP and TCP protocols create a client/server connection, **SCTP creates a connection between two peers (SCTP endpoints)** of communication, i.e., a Peer-to-Peer connection. The fundamental difference is that a peer can have multiple IP addresses (multi-

homing) both IPv4 or IPv6 (including their combination).

SCTP uses ports similar to TCP or UDP. In the past, SCTP communication was not secured directly, but via VPN. Today it is secured with DTLS, but in that case it still has to be encapsulated in the UDP datagram (Figure 14.1).

## 14.1 SCTP association

Client begins communication by a TCP packet with the SYN flag set. The server responds with a TCP packet with the SYN and ACK flags set. Finally, the client acknowledges with the packet with the ACK flag set. This is commonly called a ”three-way handshake”. Once all three steps are completed, communication can begin. In contrast, SCTP uses a ”four-way handshake”. The SCTP node initiates communication with an INIT packet. The peer responds with an INIT-ACK that includes a cookie. Node responds by COOKIE-ECHO with Cookie. Peer verifies the Cookie. In case of successful verification, the peer responds with COOKIE-ACK and the association is established.

A cookie (see Section section 3.3 Photuris) is employed during the initialization to provide protection against SYN (Synchronize) attack. SCTP association uses a four-way handshake, the last two steps of which are allowed to carry application data.

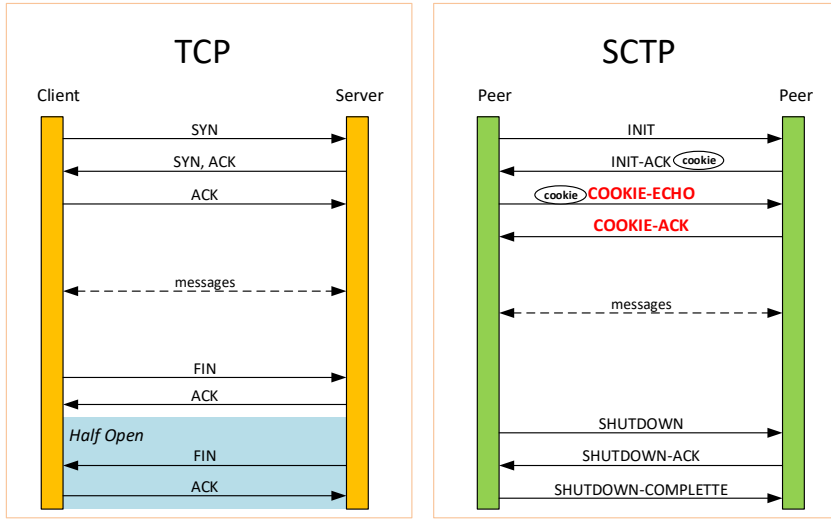



Figure 14.2 *Connection and SCTP Association*

SCTP provides for graceful close, i.e., shutdown of an active association on behalf of the peer. SCTP also allows ungraceful close, i.e., abort either on request from the peer (ABORT) or as a result of an error detected on the SCTP layer. There is no half-closed connection, as in the case of TCP.

## 14.2 Packets and Chunks

SCTP transmits data in packets containing Chunks. There are two types of chunks, control chunks and DATA chunks. Control chunks are used to establish an association (chunks INI and INIT ACK), close an association (SHUTDOWN and SHUTDOWN ACK), and probe the reachability of a particular destination transport address defined in the present association (HEARTBEAT and HEARTBEAT ACK), etc. Data chunks are used to transmit user data.

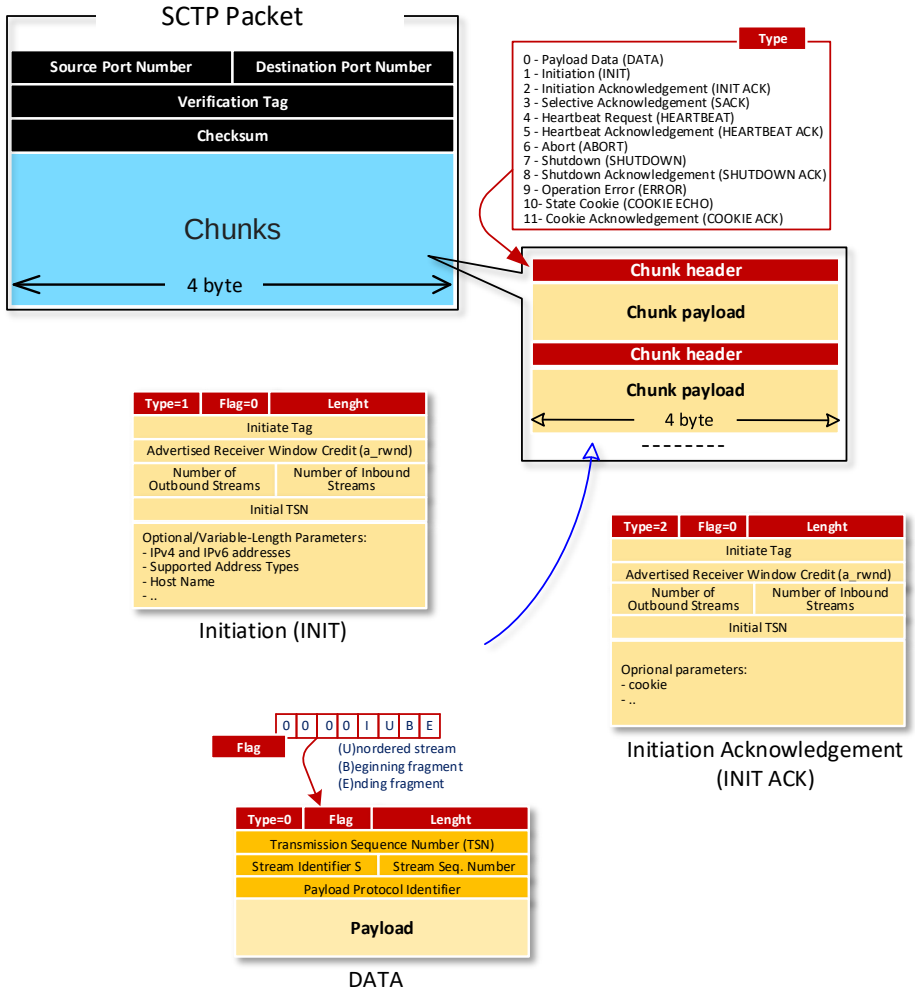



Figure 14.3 *SCTP Packets and some Chunks*

An SCTP packet (Figure 14.3) begins with a common header containing:

- **Source and destination** SCTP ports.
- **Verification Tag**, which indicates that a specific SCTP packet belongs to a specific association. That is, for example, it is not a stray packet from a previous association or a spoofed packet.

- **Checksum** of the SCTP packet.

The verification tag is established at association startup time. In the first packets of the association, the verification tag is set to 0, with the fact that both parties generated random numbers, which they inserted into the Initialization Tag item of the INIT chunk or INIT-ACK chunk, respectively. In the following SCTP packets, these random numbers are already used in the Verification Tag item.

After the common header, chunks follow. User messages, i.e., higher layer protocol data are transmitted in DATA chunks.

## 14.3 Streams

Stream is a unidirectional logical channel established from one to another associated SCTP endpoint, within which all user messages are delivered in sequence, except for those submitted to the unordered delivery service.

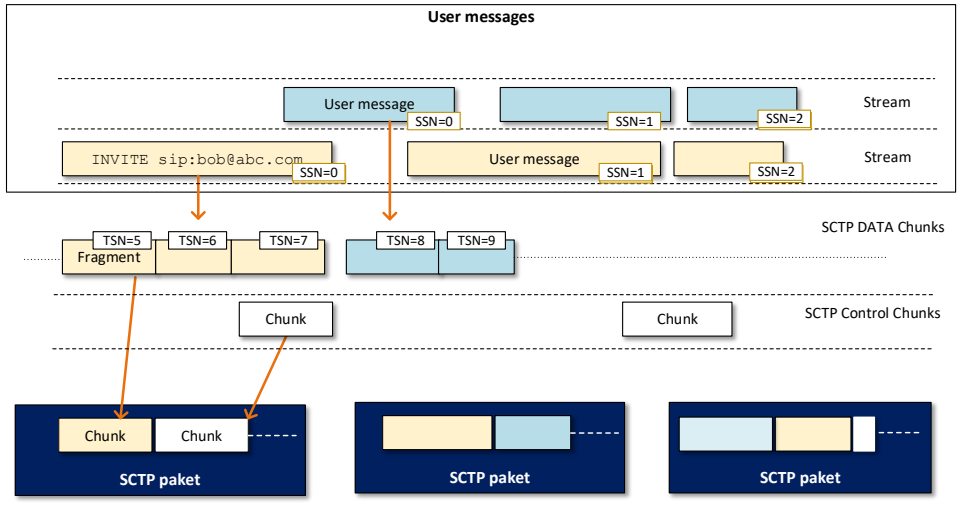



Figure 14.4 *SCTP Chunks*

User messages may (or may not) be fragmented. Individual fragments are inserted into chunks. Chunks DATA are embedded in SCTP packets together with control chunks.

We distinguish between ordered and unordered streams. Numbered streams deliver chunks in the order in which they were sent.

Within the stream, individual user messages are numbered using the Stream Sequence Number (SSN). SSN ensures sequenced delivery of the user messages within a given stream.

The individual chunks DATA are numbered using the Transmission Sequence Number (TSN) in the order in which they are sent, i.e., independently of the streams. Numbering starts from the random number ”Initialization TSN” negotiated within the association time. One TSN is attached to each chunk containing the user data to permit the receiving SCTP endpoint to acknowledge its receipt and detect duplicate deliveries.

Transmitted user messages may be fragmented, and individual fragments are embedded in chunks. The first chunk of the application message is marked with the flag ”B” and the last fragment with the flag ”E”.

A SACK (Selective Acknowledgment) chunk is used to acknowledge the received DATA chunk. SACK chunks are sent to the SCTP endpoint to acknowledge the received DATA chunks. Furthermore, SACK chunk contains information about undeliverable chunks and duplicate chunks.

## 14.4 Multi-homing

Multi-homing is designed to establish robust communication associations between two endpoints, each of which might be reachable by more than one transport address.

Within SCTP one interface is established as the primary and the rest become secondary. If the primary interface fails for whatever reason, a secondary one is selected and utilized. When the primary interface becomes available again, the communications can be transferred back without the application being aware that there was an issue. While establishing the connections, the primary and secondary interfaces are checked and monitored using a HEARBEAT/HEARBEAT-ACK acknowledgement process that validates addresses and maintains a Round Trip Time (RTT) calculation for each address. The RTT can indicate that the primary interface is slower than a secondary one and allows communications to migrate to the secondary interface.

# 15 SIP

Session Initiation Protocol (SIP) [82] is an application-layer control protocol that can establish, modify, and terminate multimedia sessions such as telephone calls and conferences. SIP can also invite participants to already existing sessions, such as multimedia conferences. Media can be added to (and removed from) an existing session.

SIP is not integrated communications system. SIP is rather a component that can be used with other networks protocols to build a complete multimedia architecture. Typically, these architectures will include protocols such as the **Real-time Transport Protocol (RTP)** (chapter 18 RTP/RTCP) for transporting real-time multimedia data (Medium), and the **Session Description Protocol (SDP)** (chapter 17 SDP) for describing multimedia sessions.

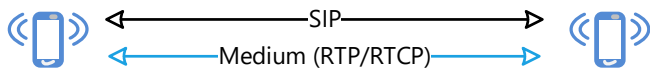



Figure 15.1 *SIP is usually used with other networks protocols*

Although SIP is a client-server protocol, we use the term '**SIP agent**' because most SIP entities act as both a client (typically the **caller**) and a server (the recipient or **callee**) during a session. The end user (**subscriber**) often uses a SIP agent in the form of a SIP phone (today rather a mobile phone), which provides traditional telephone services such as dialing, rejecting a call, receiving a call, suspending a call, transferring a call, etc. In addition, SIP enables other services such as instant messaging, etc.

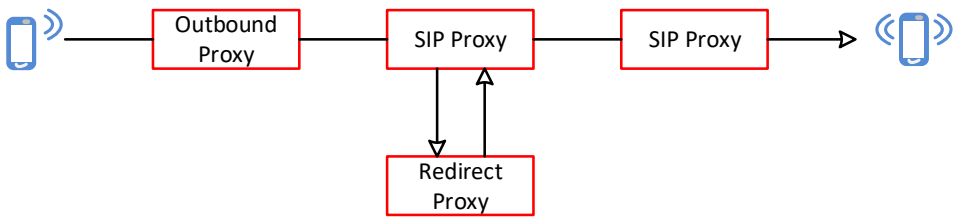



Figure 15.2 *Example of SIP network topology*

Communication of only two subscribers is possible, but it is a bit impractical, especially for public communication service providers. Thus, in the real world, usually not only two participants communicate with each other. SIP therefore introduces several intermediate entities

(Figure 15.2), with the help of which it is possible to build the required network topology.

Table 15.1 *SIP network entities*

| SIP Entity | Description |
|---|---|
| **User Agent (UA)** | A logical entity that can act as both a user agent client and user agent server. |
| **User Agent Client (UAC)** | A user agent client is a logical entity that creates a new request. The role of UAC lasts only for the duration of that transaction. In other words, if a piece of software initiates a request, it acts as a UAC for the duration of that transaction. If it receives a request later, it assumes the role of a user agent server for the processing of that transaction. |
| **User Agent Server (UAS)** | A user agent server is a logical entity that generates a response to a SIP request. The response accepts, rejects, or redirects the request. This role lasts only for the duration of that transaction. In other words, if a piece of software responds to a request, it acts as a UAS for the duration of that transaction. If it generates a request later, it assumes the role of a user agent client for the processing of that transaction. |
| **Server** | A server is a network element that receives requests in order to service them and sends back responses to those requests. Examples of servers are proxies, user agent servers, redirect servers, and registrars. |
| **Proxy, Proxy Server** | The proxy server is an intermediary entity that acts as both a server and a client for the purpose of making requests on behalf of other clients. A proxy server primarily plays the role of routing, meaning that its job is to ensure that a request is sent to another entity closer to the targeted user. Proxies are also useful for enforcing policy, for example, making sure a user is allowed to make a call. A proxy interprets, and, if necessary, rewrites specific parts of a request message before forwarding it. |
| **Outbound Proxy** | A proxy that receives requests from a client, even though it may not be the server resolved by the Request-URI. Typically, a UA is manually configured with an outbound proxy, or can learn about one through auto-configuration protocols (e.g., DHCP). |
| **Back-to-Back User Agent (B2BUA)** | A logical entity that receives a request and processes it as a user agent server (UAS). In order to determine how the request should be answered, it acts as a user agent client (UAC) and generates requests. Unlike a proxy server, it maintains dialog state and must participate in all requests sent on the dialogs it has established. |
| **Registrar** | A register is a SIP endpoint that accepts REGISTER requests and places the information it receives in those requests into a location service for the domain it handles. The location service links one or more IP addresses to the SIP URI of the registering agent. More than one user agent can register at the same URI, with the result that all registered user agents receive the calls to the URI. |
| **Redirect Server** | A user agent server that generates 3xx (Redirection) responses to requests it receives, directing the client to contact an alternate set of URIs. The redirect server allows proxy servers to direct SIP session invitations to external domains. |
| **Gateway** | Gateways can be used to interface a SIP network to other networks, such as the public switched telephone network, which use different protocols or technologies. |

The SIP protocol not only replaces classic telephone communication, but calls can also transmit video, text, etc. That is why we do not generally talk about ”call” but about ”**multimedia session**”. In addition, the SIP protocol opens up the possibilities of other services, such as **conferences** (of course, multimedia), and then, for example, services that provide information about the status of participants (e.g., Online, Do Not Disturb, etc.). This service can provide not only information about whether other participants are available, but other information about other participants can be provided, such as contact information as well as business information about the services offered.

A specific communication supported by the SIP protocol is ”Push to talk”, which enables half-duplex communication within a group of participants. This method of communication is advantageous, for example, within rescue teams.

Table 15.2 *SIP methods*

| SIP Methods | Description |
|---|---|
| **ACK** | Confirms that an entity has received a response to an INVITE request |
| **BYE** | Terminates a session and confirms that an entity has received a final response to an INVITE request |
| **CANCEL** | Cancels a pending request |
| **INFO** | Carries application level information between endpoints |
| **INVITE** | Initiates a SIP dialog with the intent to establish a call. It is sent by a user agent client to a user agent server |
| **MESSAGE** | Instant Messaging |
| **NOTIFY** | Informs a subscriber of notifications of a new event |
| **OPTIONS** | Query the capabilities of an endpoint |
| **PRACK** | PRovisional ACKnowledgement – improves network reliability by adding an acknowledgement system to the provisional Responses (1xx). PRACK is sent in response to provisional response (1xx) |
| **PUBLISH** | Publishing event to a notification server. PUBLISH is similar to REGISTER in that it allows a user to create, modify, and remove the state in another entity, which manages this state on behalf of the user |
| **REFER** | Asks recipient to issue SIP request for the purpose of call transfer |
| **REGISTER** | Registers the SIP URI listed in the To header field with a location server and associates it with the network address given in a Contact header field |
| **SUBSCRIBE** | Initiates a subscription for notification of events from a notifier |
| **UPDATE** | Allows a client to update parameters of a session (such as the set of media streams and their codecs), but has no impact on the state of a dialog |

## 15.1 SIP messages

SIP messages are sent between **SIP peers** which govern establishment, termination and other essential elements of a session. SIP can be used for creating, modifying and terminating sessions consisting of one or several media streams.

SIP is a **text-based protocol** with syntax similar to that of HTTP/1.1. There are two different types of SIP messages: requests and responses. The first line of a request has a method, defining the nature of the request, and a Request-URI, indicating where the request should be sent. The first line of a response has a response code.

Table 15.3 *SIP response codes*

| Response code | Description |
|---|---|
| **Provisional (1xx)** | Zero, one or multiple provisional responses may arrive before one or more final responses are received. Provisional responses for an INVITE request can create ”early dialogs” |
| **Success (2xx)** | The action was successfully received, understood, and accepted |
| **Redirection (3xx)** | Further action needs to be taken (typically by a sender) to complete the request (may contain one or more Contact header field values providing new addresses where the callee might be reachable) |
| **Client Error (4xx)** | The request contains a bad syntax or cannot be fulfilled on the server |
| **Server Error (5xx)** | The server failed to fulfill an apparently valid request |
| **Global Failure (6xx)** | The request cannot be fulfilled by any server |

## 15.2 SIP URI

Each resource of a SIP network, such as a user agent or a voicemail box, is identified by a uniform resource identifier (URI), based on the general internet message standard also used in Web services or e-mail.

The URI scheme used for SIP is “sip:” or “sips:” and a typical SIP URI is as follows (port and parameters are optional):

```
sip:username@host:port;parameters
```

**Example:** `sip:Libor.Dostalek@example.org`

If secure transmission is required, the scheme sips: is used and mandates that each hop is over TLS.

## 15.3 TEL URI

Alternatively to SIP or SIPS, URI can be used tel URI (parameters are optional):

```
tel:telephone-number;parametrs,
```

where telephone-number may be E.164 number or private number. Telephone number is important especially for backward compatibility in legacy networks. SIP agent must translate first the telephone number from tel URI using DNS ENUM to SIP URI.

Visual separators: dash, slash and period are ignored.

**Example**: `tel:+420-201-555-0123`

## 15.4 URN

URN URI is for case when the caller does not want to solve how to call the emergency services, he needs help. For emergency calls are defined 'sos' URN Sub-Services (RFC 5031 [83]).

Table 15.4 *SIP emergency codes*

| Service | Description |
|---|---|
| **sos** | Emergency (general) |
| **sos.ambulance** | Ambulance service |
| **sos.animal-control** | Animal control |
| **sos.fire** | Fire service |
| **sos.gas** | leaks and gas emergencies |
| **sos.marine** | Maritime search and rescue |
| **sos.mountain** | Mountain rescue |
| **sos.physician** | Physician referral service |
| **sos.poison** | Poison control center |
| **sos.police** | Police, law enforcement |

Examples:

```
urn:service:sos
urn:service:sos.fire
```

## 15.5 SIP message format

The SIP message (Figure 15.3) has Internet Message Syntax [84], like HTTP/1.1 or SMTP messages. The messages consist of a **header** and a **body**, which are separated by **empty line**. The interpretation of the body depends on the request method. SIP body consists of the header fields similar to SMTP header fields. However, there is a fundamental difference. The response contains the same Form: and To: header fields as the request!

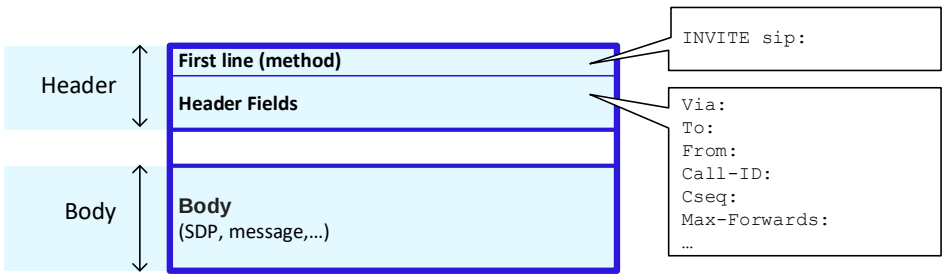



Figure 15.3 *SIP message format*

In the SIP protocol, the client sends a message that contains the method in the first line of the header (similar to the HTTP/1.1 protocol). The server then responds with a message that contains the result code in the first line of the header.

Similar to SMTP (chapter 20 SMTP), SIP protocol optionally uses Display Name in some Header Fields. In this case optional Display Name indicates the name that will be displayed to the user of a SIP application, while SIP URI is in angle brackets (”<” and ”>”).

However, as it has already been emphasized, there is one absolutely fundamental difference compared to the SMTP protocol, which can confuse the reader, especially when first encountering the analysis of SIP protocol packets: the **SIP server copies some headers from the client's request into the response**. These are mainly the following headers: `To, From, Call-ID, CSeq` and `Via`. On the contrary, it does not do so for other headers (Contact, Accept-*, etc.). It can be seen in the example (source RFC-3261 [82]) on Table 15.5.

Table 15.5 *SIP example*

| **Request** | **Response** |
|---|---|
| `INVITE sip:bob@biloxi.com SIP/2.0` | `SIP/2.0 200 OK` |
| … | …. |
| `To: Bob <sip:bob@biloxi.com>` | `To: Bob <sip:bob@biloxi.com>;tag=a6c85cf` |
| `From: Alice` | `From: Alice` |
| `<sip:alice@atlanta.com>;tag=1928301774` | `<sip:alice@atlanta.com>;tag=1928301774` |
| `Call-ID: a84b4c76e66710@pc33.atlanta.com` | `Call-ID: a84b4c76e66710@pc33.atlanta.com` |
| `CSeq: 314159 INVITE` | `CSeq: 314159 INVITE` |
| `Contact: <sip:alice@pc33.atlanta.com>` | `Contact: <sip:bob@192.0.2.4>` |

## 15.6 SIP registration (first part)

After turning on the SIP device, the first dialog is usually device registration. The goal of registration is to tell the network at which URI (or IP address) the client is located. More precisely: registration means the creation of a so-called address-of-record (AOR) in the Registrar Localization Service (Localization Database).

The **Registrar writes this association**, also called a binding, to a database, called the location service. The location service is just an abstract concept. It generally contains information that allows a proxy to input a URI and receive a set of zero or more URIs that tell the proxy where to send the request, i.e., provides information that is used to locate the callee.

The URI specified in the ”Contact” header field of the REGISTER request is entered into the localization service. For example:

```
REGISTER …
Contact: sip:pepa@example.org;expires=300
```

creates a record in the localization service for the participant sip:pepa@example.org for a period of five minutes. If you are surprised by the 5 minutes, it should be noted that the

participant has to renew his registration again and again. In the event that the subscriber turns off the SIP device, the network will find that the subscriber is unavailable because he has stopped renewing his registration.

Registration is important for the client to receive incoming calls. Even without registration, the participant can contact the outgoing proxy, for example, in mobile networks, an unregistered subscriber can make emergency calls (does not have to be implemented)

Networks often require authentication of subscribers as part of registration, if they have not already been authenticated. As shown in Figure 15.4, during SIP registration, the network may require subscriber authentication. Authentication mechanisms are taken from the HTTP protocol (see section 8.13 HTTP authentication).

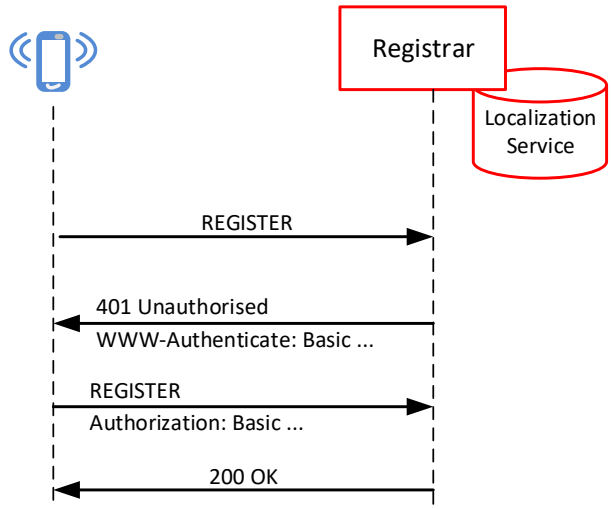



Figure 15.4 *SIP REGISTER*

During registration, the client sends a request carrying the REGISTER method to the Registrar. The data that the participant wants to enter into the Location Service are usually transferred in the Contact header field. The request with the REGISTER method is subsequently repeated by the client at regular intervals as long as it is logged in. Registration is used for SIP request routing, but is not related to the authorization of outgoing requests.

Note that in Figure 15.4 there is no media transfer at all (RTP/RTCP). The medium is not involved in the registration in any way.

Deregistration means the invalidation of the address-of-record (AOR) in the Location Service. Deregistration can occur at both the initiative of the subscriber and at the initiative of the network. A network-initiated deregistration occurs when a subscriber suddenly turns off their device. From the point of view of the network, this means that the subscriber will stop making repeated requests using the REGISTER method and after a set time interval, the network will deregister the subscriber.

Unsubscribing at the subscriber's initiative is done by a request with the REGISTER request with the `Expires: 0` parameter in the `Contact:` header field. The URI specified in the `Contact:` header field of this request is invalidated in the localization service. If `Contact: *` was specified, all URIs associated with that subscriber would be invalidated.

### 15.6.1 Authentication

Similar to the HTTP protocol, header fields are used for authentication:

- **WWW-Authenticate**, which is an authentication prompt. A SIP message with this header is usually associated with the previous result code 401 Unauthorized.
- **Authorization**, which contains the authorization data.

The SIP protocol allows several authentication methods, which are mostly taken from the HTTP protocol:

- **Basic authentication**, for example, using a user name and password.
- **Authentication by the Kerberos** protocol, which is particularly interesting for intranets where Windows domains are used. The Kerberos protocol ticket is wrapped in the SPNEGO envelope (RFC-4559) and then in the Authorization header of the "Negotiate" authentication method. This method also enables NTLM authentication used in older Windows networks.
- **Digest authentication**, which also uses password authentication, but the password is not directly transmitted by the network (it is hidden in the hash). This is challenge-response authentication. The query is placed in the WWW-Authenticate header and the response in the Authorization header.

In mobile networks, the Digest method uses both the WWW-Authenticate and Authorization header fields. But that is where the similarity ends with Digest authentication. In the mentioned headers, the authentication is the AKA protocol. AKA uses a secret shared between the USIM/ISIM smart card (popularly known as SIM) and the mobile network provider's subscriber database for authentication (see section 15.15 Lawful Interception).

### 15.6.2 SIP INVITE

With the INVITE method "we call the subscriber at his SIP URI (or TEL URI)". If we only know his phone number (TEL URI), then using DNS ENUM (subsection 15.9.2 DNS ENUM) we must first translate the phone number to his SIP URI. If we have already had a SIP URI, we can use DNS NAPTR to determine the SIP server (proxy) of the called party and send a SIP INVITE message to the server identified in this way.

A typical example (taken from RFC 3261 [82]) is a SIP message exchange between two users, Alice and Bob. In this example, Alice uses a SIP application (UA) on her smartphone to call Bob on his SIP phone over the Internet. Also shown are two SIP proxy servers that act on behalf of Alice and Bob to facilitate the session establishment. This typical arrangement is often referred to as the "SIP trapezoid" as shown by the geometric shape in Figure 15.5

The first line of the request message contains the method name (INVITE) followed by header fields:

- `Via:` contains the address (pc33.atlanta.com) at which Alice is expecting to receive responses to this request. It also contains a branch parameter that identifies this transaction.
- `To:` contains a display name (Bob) and a SIP or SIPS URI (sip:bob@biloxi.com) towards which the request was originally directed.
- `From:` also contains a display name (Alice) and a SIP or SIPS URI (sip:alice@atlanta.com)

that indicate the originator of the request. This header field also has a tag parameter containing a random string (1928301774) that was added to the URI by the smartphone. It is used for identification purposes.

- `Call-ID`: contains a globally unique identifier for this call, generated by the combination of a random string and the smartphone's host name or IP address. The combination of the To tag, From tag, and Call-ID completely defines a peer-to-peer SIP relationship between Alice and Bob and is referred to as a dialog.
- `CSeq`: (Command Sequence) contains an integer and a method name. The CSeq number is incremented for each new request within a dialog and is a traditional sequence number.
- `Contact`: contains a SIP or SIPS URI that represents a direct route to contact Alice, usually composed of a username at a fully qualified domain name (FQDN). While an FQDN is preferred, many end systems do not have registered domain names, so IP addresses are permitted. While the Via header field tells other elements where to send the response, the Contact header field tells other elements where to send future requests.
- `Content-Length`: contains the length of the message body.
- `Max-Forwards`: serves to limit the number of hops a request can make on the way to its destination. It consists of an integer that is decremented by one at each hop.
- `Content-Type`: contains a description of the message body (not shown in Figure).

The details of the session, such as the type of media, codec, or sampling rate, are not specified by SIP protocol. This information is communicated in the **body of a SIP message**.

Since the Alice's smartphone does not know the location of Bob or his SIP server Alice sends the INVITE to her Outbound Proxy atlanta.com. The address of the atlanta.com SIP server could have been configured in Alice's smartphone, or it could have been discovered by DHCP, for example.

The atlanta.com SIP server is the SIP proxy server. A proxy server receives SIP requests and forwards them on behalf of the requestor. In this example, the proxy server receives the INVITE request and sends a 100 (Trying) response back to Alice's smartphone. The 100 (Trying) response indicates that the INVITE has been received and that the proxy is working on her behalf to route the INVITE to the destination. This response contains the same `To`, `From`, `Call-ID`, `CSeq` and branch parameter in the `Via` as the INVITE, which allows Alice's smartphone to correlate this response to the sent INVITE.

The atlanta.com proxy server locates the proxy server at biloxi.com, possibly by performing a particular type of DNS (Domain Name Service) lookup to find the SIP server that serves the biloxi.com domain. As a result, it obtains the IP address of the biloxi.com proxy server and forwards, or proxies, the INVITE request there. Before forwarding the request, the atlanta.com proxy server adds an additional Via header field value that contains its own address (the INVITE already contains Alice's address in the first Via). The biloxi.com proxy server receives the INVITE and responds with a 100 (Trying) response back to the atlanta.com proxy server to indicate that it has received the INVITE and is processing the request. The proxy

server consults a database Location Service, which contains the current IP address of Bob. The biloxi.com proxy server adds another Via header field value with its own address to the INVITE and proxies it to Bob's SIP phone.

Bob's SIP phone receives the INVITE and alerts Bob to the incoming call from Alice so that Bob can decide whether to answer the call, that is, Bob's phone rings. Bob's SIP phone indicates this in a 180 (Ringing) response, which is routed back through the two proxies in the reverse direction. Each proxy uses the Via header field to determine where to send the response and removes its own address from the top. As a result, although DNS and location service lookups were required to route the initial INVITE, the 180 (ringing) response can be returned to the caller without maintaining lookups or without state in the proxies. This also has the desirable property that each proxy that sees the INVITE will also see all responses to the INVITE.

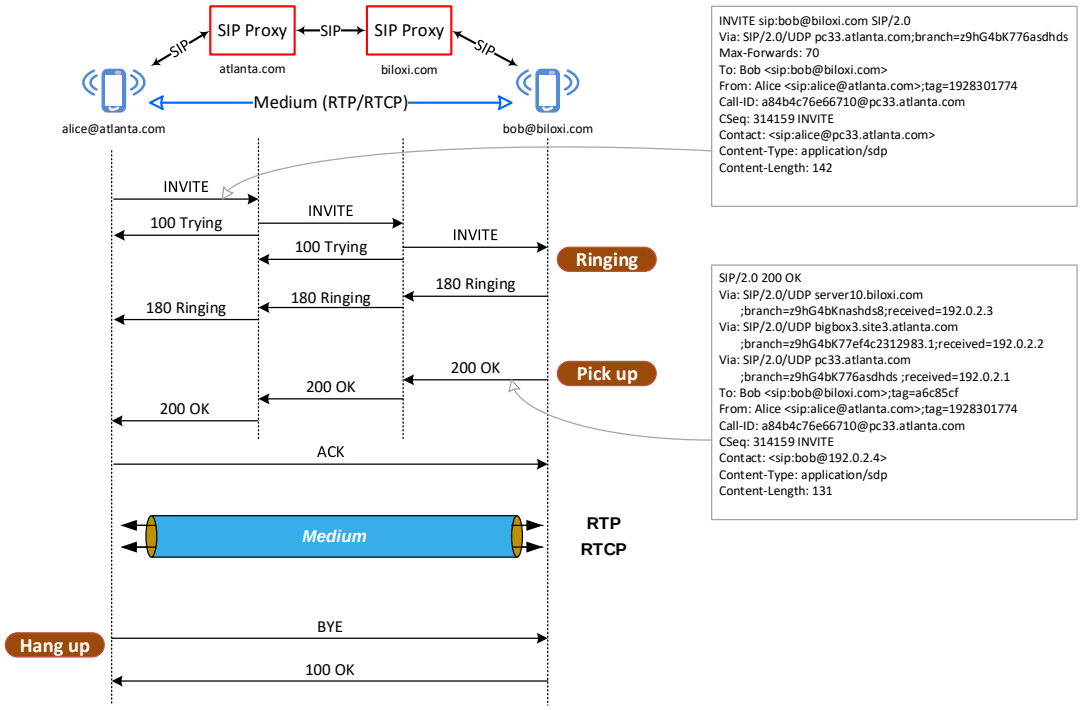



Figure 15.5 *SIP message format*

When Alice's smartphone receives the 180 (Ringing) response, it passes this information to Alice, perhaps using an audio ring back tone or by displaying a message on Alice's screen. In this example, Bob decides to answer the call. When he picks up the handset, his SIP phone sends a 200 (OK) response to indicate that the call has been answered. The 200 (OK) contains a message body with the SDP media description of the type of session that Bob is willing to establish with Alice. As a result, there is a two-phase exchange of SDP messages: Alice sent one to Bob, and Bob sent one back to Alice. This two-phase exchange provides basic negotiation capabilities and is based on a simple offer/answer model of SDP exchange.

If Bob did not wish to answer the call or was busy on another call, an error response would have been sent instead of the 200 (OK), which would have resulted in no media session being established.

The first line of the response contains the response code (200) and the reason phrase (OK). The remaining lines contain header fields. The `Via`, `To`, `From`, `Call-ID`, and `CSeq` header fields are copied from the INVITE request. (There are three Via header field values – one added by Alice's SIP phone, one added by the atlanta.com proxy, and one added by the biloxi.com proxy.) Bob's SIP phone has added a tag parameter to the To header field. This tag will be incorporated by both endpoints into the dialog and will be included in all future requests and responses in this call. The Contact header field contains a URI at which Bob can be directly reached at his SIP phone. The Content-Type and Content-Length refer to the message body (not shown) that contains Bob's SDP media information.

In this case, the 200 (OK) is routed back through the two proxies and is received by Alice's smartphone, which then stops the ring back tone and indicates that the call has been answered. Finally, Alice's smartphone sends an acknowledgment message, ACK, to Bob's SIP phone to confirm the reception of the final response (200 (OK)). In this example, the ACK is sent directly from Alice's smartphone to Bob's SIP phone, bypassing the two proxies. This occurs because the endpoints have learned each other's address from the Contact header fields through the INVITE/200 (OK) exchange, which was not known when the initial INVITE was sent. The lookups performed by the two proxies are no longer needed, so the proxies drop out of the call flow. This completes the INVITE/200/ACK three-way handshake used to establish SIP sessions.

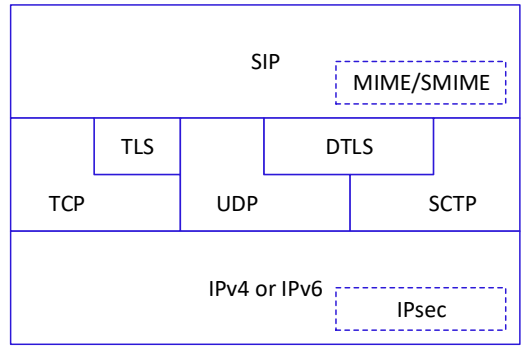



Figure 15.6 *SIP protocol stack*

Alice and Bob's media session has now begun, and they send media packets using the format to which they agreed in the exchange of SDP. In general, the end-to-end media packets take a different path from the SIP signaling messages.

## 15.7 Underlying protocols

SIP works in concert with several undelaying protocols (Figure 15.6). SIP clients typically use TCP or UDP on port numbers 5060 and/or 5061 to connect to SIP servers and other SIP endpoints. Port 5060 is commonly used for non-encrypted signaling traffic whereas port 5061 is typically used for traffic encrypted with Transport Layer Security (TLS). Especially for communication between the SIP intermediate entities (proxies, SBC), SIP protocol uses as undelaying protocol SCTP.

## 15.8 SIP protocol security

Encryption of SIP messages may be based on several mechanisms:

- IPsec.
- S/MIME.
- TLS/DTLS.

IPsec and TLS/DTLS mechanisms secure whole communication (SIP message header and SIP message body). In contrast, S/MIME secure SIP message body only.

Is it necessary to secure the SIP message body? A frequently asked question is why securing the SIP body is necessary. One important reason is the need to secure the media stream (e.g., using SRTP). This is because the cryptographic material for securing media streams (SRTP) is often transmitted in the SIP message body.

The media streams, which occur on different connections to the stream, can be subsequently encrypted with SRTP. The key exchange for SRTP is performed with SDES (RFC 4568 [85]), or the newer and often more user friendly ZRTP (RFC 6189 [86]), which can automatically upgrade RTP to SRTP using dynamic key exchange (and a verification phrase). One can also add a MIKEY (RFC 3830 [87]) exchange to SIP and in that way determine session keys for use with SRTP.

## 15.9 SIP and DNS

**Note:** DNS is explained in more detail in chapter 24 DNS and DNSSEC . Here, we focus only on the specifics relevant to the SIP protocol.

In particular, SIP uses two types of URIs:

- TEL URI, e.g. `tel:+420-123-456-789`, where so-called visual separators (dash, slash and period) are ignored and the result is converted to a SIP URI using DNS ENUM.
- SIP URI, e.g., `sip:novak@example.org;transport=udp`.

We have to use DNS to translate these URIs to IP addresses, transport protocols and their ports. NAPTR records are used for this in DNS. Its use is a little complicated at first glance, but it is easy to understand with examples.

### 15.9.1 NAPTR

To understand how phone numbers are used in the SIP protocol, we must first describe the DNS resource record Name Authority Pointer (NAPTR). NAPTR is a DNS resource record similar to A or CNAME records. The goal of this record is to translate a string into another string (e.g., phone number to SIP URI).

NAPTR generally has the form:

```
Domain TTL IN NAPTR Order Pref Flags Service Regexp Replacement
```

where:

- `Domain`, `TTL` and `IN` have a standard meaning in DNS.
- `NAPTR` indicates NAPTR resource record.
- `Order` specifies the order in which NAPTR records (of the same domain) must be processed. Lower numbers are processed before higher numbers. Once a record with a specific order is processed, no more records are searched.
- `Pref` determines the order in which NAPTR records will be processed if they have the same order entry.
- `Flags` – we have `S`, `A`, `U` and `P` flags. The flag tells what should be the next step in DNS translation:
  - `S` – the result of translation of the PTR record is a DNS name referring to the SRV record.
  - `A` – the result of translation of the PTR record is a DNS name referring to the A or AAAA record.
  - `U` – the result of translation of the PTR record is URI.
  - `P` – application specific.
- `Regexp` – the regular expression to apply to the DNS query.
- `Replacement` – what to replace the result of the regular expression with.
- `Service` – most often specifies lower layer protocols. An item can contain an empty string.

Table 15.6 *NAPTR Services*

| Service | Application protocol | Transport protocol |
|---|---|---|
| **SIP+D2T** | SIP | TCP |
| **SIPS+D2T** | SIP | TCP |
| **SIP+D2U** | SIP | UDP |
| **SIP+D2S** | SIP | SCTP |
| **SIPS+D2S** | SIP | SCTP |
| **SIP+D2W** | SIP | WebSocket [88] |
| **SIPS+D2W** | SIP | WebSocket [88] |

### 15.9.2 DNS ENUM

DNS ENUM (tElephone NUMber Mapping) is a system for translating telephone numbers to SIP URIs.

The e164.arpa domain has been registered for translating phone numbers according to the ITU E.164 standard to SIP URI.

**Example:** The following fully qualified DNS name corresponds to the telephone number:

+420 123 45 67 89:

```
9.8.7.6.5.4.3.2.1.0.2.4.e164.arpa.
```

It can then have following record in DNS:

```
$ORIGIN 9.8.7.6.5.4.3.2.1.0.2.4.e164.arpa.
      IN NAPTR 100 10 "u" "E2U+sip"  "!\^.*\$!sip:user@example.org!"
```

The `"u"` flag means that the result will be an absolute URI. `"E2U+sip"` (E164 to URI) [89] means that the E164 phone number will be translated into SIP URI. The regular expression `"!.*$!sip:user@example.org!"` consists of two parts (separated `!`): the first part `.*` means take the entire input (whole phone number) and replace it with the second part: sip:user@example.org., i.e., the call to the phone number +420 123 45 67 89 was transformed into a call to the SIP URI sip:user@example.org.

### 15.9.3 SIP URI translation

Finally, `sip:user@example.org` still needs to be investigated into information leading to finding the SIP server of called subscriber. In DNS, for example, we have:

```
$ORIGIN example.org.
IN NAPTR 100 10 "S" "SIP+D2U”  "" _sip._udp.example.org.
IN NAPTR 102 10 "S" "SIP+D2T" "!^.*$!sip:helpdesk@example.org!"
                    _sip._tcp.example.org.
```

First, the record with order 100 is processed. It says that the SIP server of the called party is available via the UDP transport protocol and has the SRV record `_sip._udp.example.org`. If this record fails, then the next record is used to and the SIP URI `sip:user@example.org` is rewritten to `sip:helpdesk@example.org` and a connection is established via the TCP protocol to the helpdesk on SIP server that has an SRV record in the DNS: `_sip.tcp.example.org`.

## 15.10 Network architecture

Figure 15.7 shows the connection of home WiFi to the Internet using a router (Default gateway), which is connected to the Internet through a port (landline, optical cable, WiFi of internet provider, etc.)

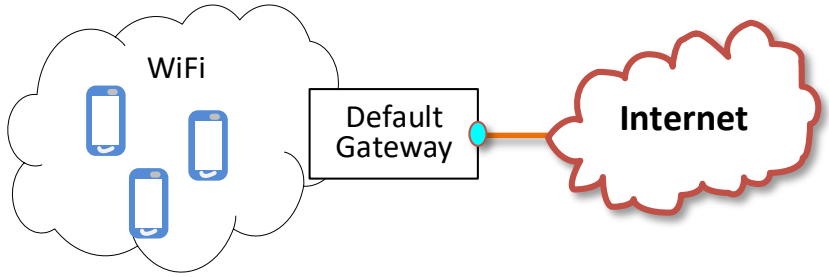



Figure 15.7 *Classic home WiFi connection to the Internet*

The mobile network has a more complicated infrastructure, but from a very high level, it can appear similar from a user's perspective. Only the Gateway has more "ports". They are called APN (Access Point Name) in 4G networks or DNN (Data Network Name) in 5G networks. APNs/DDNs specify which network IP datagrams should be routed to. They can be routed:

- To the Internet, i.e., "Mobile Data".
- To the IPX network (IP eXchange) through which it is connected to other providers. IPX is a worldwide IP network not connected to the Internet. It has its own Root Name Servers, strictly prescribed how DNS names should be formed, etc. It has its own providers, for example, Deutsche Telekom.
- To IMS (IP Multimedia Subsystem), which ensures the provision of multimedia services replacing historical call switching. It is a system of SIP proxy that is protected at the input (and output) by SBC (Session Border Controller). IMS may or may not be operated by a mobile operator together with a mobile network.
- To private network (e.g., company internal network).

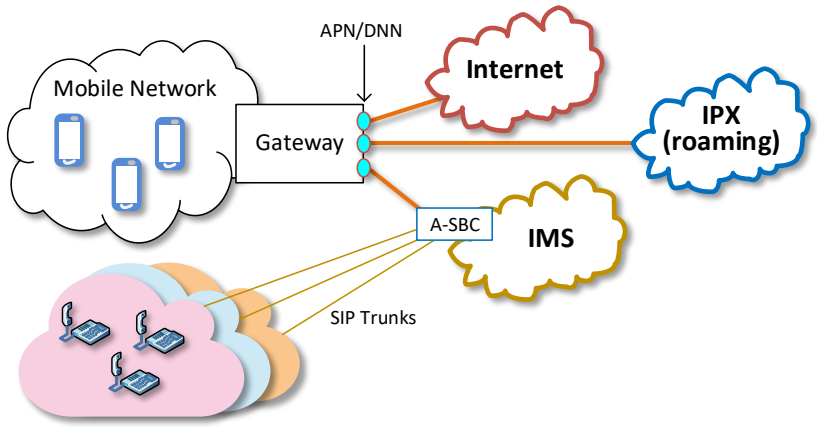



Figure 15.8 *Mobile network*

**Conclusions:**

1. The mobile network provides IP connectivity only. At the application layer (SIP and RTP/RTCP protocols) there is a communication between the mobile and the A-SBC, which is a SIP B2BUA entity.
2. The mobile network guarantees the transmission bandwidth for multimedia sessions. This is different from mobile applications such as WhatsApp.

## 15.11 SBC

A SBC (*Session Border Controller*) is Back-to-Back User Agent (SIP B2BUA entity) combined with security functions and other functionalities (Figure 15.9). The SBC usually consists of:

- **Media Gateway Controller (MGC)** is a SIP B2BUA entity that accepts not only the SIP protocol, but also must ensure termination of the transmission medium (RTP/RTCP). It then forwards accepted requests on behalf of the origin subscriber to the target server. If the request was encrypted, it must be decrypted first before accepting the session.
- **Media Gateway (MG)** that accepts the transmission medium (RTP/RTCP). If the SBC accepts the session on behalf of the callee, then it must terminate not only the SIP connection, but also the transmission medium (RTP/RTCP). Termination of SIP and termination of the transmission medium must be coordinated by, for example, the H.248 protocol.
- **Security functions** (”firewalls”), which are usually different on the input side and on the output side.
- **Other functions** (not shown in Figure 15.9), which can be, for example, authentication of participants against an external database of participants, accounting, etc.

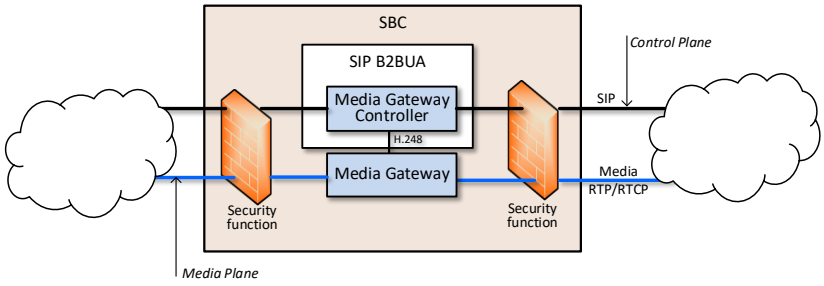



Figure 15.9 *Session Border Controller (SBC)*

Some manufacturers supply multifunctional boxes where a gateway can also be configured instead of a B2BUA entity. The topology remains similar, but on the one side there are SIP/RTP/RTCP, on the other side there are different protocols, e.g., SS7 [74]. There are two types of SBC:

- **A-SBC** (A = Access) accessed by subscribers. As a rule, A-SBC authenticates subscribers. In the case of mobile networks, the subscriber is authenticated using a shared secret stored in his USIM smart card. However, USIM cards are primarily intended for mobile

network authentication, not for IMS authentication. The standard makes it possible to use a different shared secret for the mobile network and another for IMS. Smart cards for IMS are referred to as ISIM (I = IMS). It is possible to have both secrets on one smart card (in different applications on the same smart card). However, since mobile operators often also provide IMS, they use USIM for authentication to both networks.

- **I-SBC** (I = Interconnect), through which the provider connects with other providers. This is where provider systems authenticate each other. Asymmetric cryptography (more specifically PKI) is used here.

## 15.12 IMS

IMS (IP Multimedia Subsystem)[90] ensures the connection of network participants using the system of SIP proxies. The mobile network is connected to the A-SBC (Access Session Border Controller) via the IP protocol. However, other networks that use the SIP protocol can also be connected to the A-SBC. This connection is referred to as a **SIP Trunk**.

There are two types of connections on A-SBC:

- Public telephone networks generally switched to IP. While the provision of telephone calls is declining, landlines are used to provide Internet connections. If they are used for telephone calls, then they switch to the SIP protocol.
- Private connection of companies (SIP Trunks).

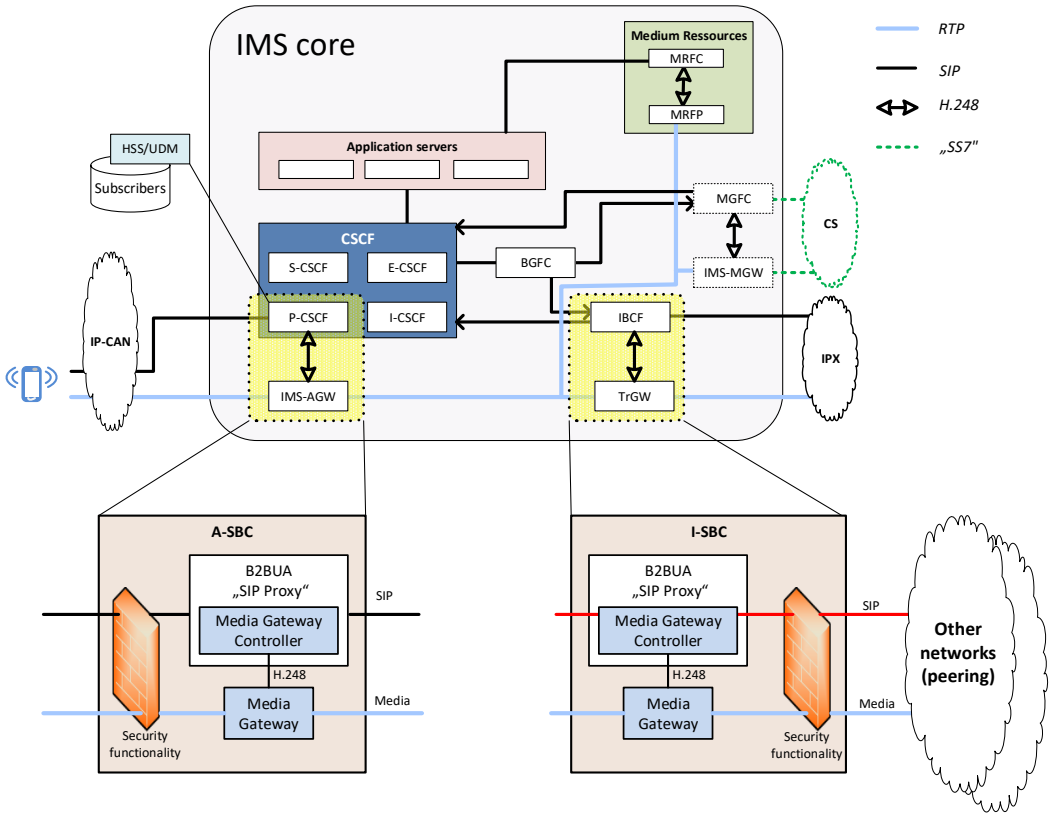



Figure 15.10 *Example of IMS connection (IMS terminology is used in the upper part, SIP terminology in the lower part)*

The center of IMS is Cell Session Control Functions (CSCF), which is responsible for processing SIP protocol requests. CSCF consists of four SIP proxies. CSCF does not contain any security features. It is assumed that IMS core is protected by SBC (especially by A-SBC).

Application services are superstructure services on top of IMS. Individual application functions can also be implemented by third-party servers. Examples of application functions are presentation services, conference servers, servers for Instant Messaging, servers providing ”push-to-talk” services, converting messages to SMS and forwarding SMS to the SMS center, etc.

Table 15.7 *IMS Entities (CSCF = Call Session Control Function)*

| | |
|---|---|
| **P-CSCF** | It is the first IMS entity to contact the subscriber mobile device. Although it has P for Proxy in the name, in the SIP protocol terminology it is B2BUA. |
| **S-CSCF** | Processing calling requests and maintaining session status. It provides the authentication of subscribers (on behalf of the P-CSCF) using the HSS/UDM. In the HSS, it also searches for the subscribers who are called. |
| **E-CSCF** | Processing emergency calls ('112'). |
| **I-CSCF** | It is a contact point for incoming calls from foreign networks. |
| **MGCF** | **The Media Gateway Control Function** is a gateway to networks based on protocols other than the SIP (GSM, etc.). In particular, it provides conversion of network protocols and media conversion format (transcoding). |
| **BGCF** | **The Breakout Gateway Control Function** is a dispatcher of outgoing requests. It decides what external network the request is going to be transmitted to: whether to SIP-based networks or obsolete networks (switching circuits) based on SS7. |
| **IBCF** | An **Interconnection Border Control Function** makes appropriate modifications to SIP/SDP requests between networks of different operators. It can make IPv4 and IPv6 conversions, hiding the operator topology, generating charging, etc. |
| **TrGW** | The **Transition Gateway** performs IP address/port translation, IPv4 and IPv6 conversion, etc. Under certain circumstances, it can also convert media (transcoding). |
| **EATF** | The **Emergency Access Transfer Function** is a gateway to the integrated rescue system. Requirements are received both from the E-CSCF network itself and from external networks (I-CSCF). |
| **IMS-ALG** | The **IMS Application Level Gateway** performs SIP/SDP level adjustments, e.g., the authenticated subscriber may be in the appropriate SIP header marked as trusted applying to communication with other entities. The IMS-ALG also converts IP addresses, etc. |
| **IMS-AGW** | **IMS Access Gateway** performs IP address / port translation, IPv4 and IPv6 conversion, etc. Under certain circumstances, it can also convert media format (transcoding). |
| **AF** | **Application Functions** – application based on SIP/RTP protocols. |
| **HSS/UDM** | **Home Subscriber Service (4G) / Unified Data Management (5G)** |
| **IP-CAN** | **IP-Connectivity Access Network** |
| **CS** | **Circuit Switched** Network (e.g. GSM) |
| **IPX** | **IP eXchange** |

The media resource provides audio announcements, or the mixing of multimedia streams, for example, for CSCF, it can provide audio: "The called party is temporarily unavailable".

In addition to these entities, IMS uses the already mentioned HSS/UDM. HSS/UDM contains information about subscribers, their services, the subscriber's SIP URI (or phone URI), location information (in SIP terminology: location service) and for each subscriber also a shared secret that it shares with the subscriber's smart card (USIM /ISIM). Based on this secret, the subscriber authenticates itself to the P-CSCF, which forwards the request to the S-CSCF, which handles it with the help of HSS/UDM.

As a result of successful authentication, access is not only enabled, but the participant also

receives their SIP URI (respectively tel URI). In layman's terms: the phone number is not configured in the mobile phone, nor on the USIM/ISIM smart card, but the subscriber receives it only as part of his authentication – they are only records in the HSS/UDM database. This is also the reason why the subscriber has to re-login to the network if his "telephone number" changes.

If the called party is not in the same network (in the same IMS), the request is forwarded to the BGFC entity, which determines whether the called party is a subscriber of another IMS provider, then forwards the request to the IBCF entity. If it is a subscriber of an outdated network (e.g., GSM), then it forwards the request to the MGFC entity.

In Voice over LTE (VoLTE) and Voice over 5G (VoNG), participants are addressed not by a phone number but by a SIP URI resembling an e-mail address. If classic phone numbers are used, then it is necessary to use DNS ENUM to convert the phone number to the SIP URI of the called party.

**Note:** SIP URI is basically used for communication in IMS, tel URI is just a complication. It can become puzzling why users do not want to use a SIP URI when it has a similar syntax to the email they use regularly. It is just a tradition. The interesting part is that if a SIP URI is already used, users do not enter the first and last name (as in the case of an email address), but use the name in the form of a phone number.

## 15.13 Roaming

In the case of roaming (Figure 15.11), we distinguish the following networks:

- **Home network**, where the subscriber is registered. The provider of this network issued him a USIM/ISIM smart card. This means that the provider's database stores the shared secret also stored on subscriber's smart card, which is used to log the subscriber into the network.
- **Visited network**, where the subscriber is located, if his home network is unavailable. If a subscriber wants to login to a visited network, then its home network must provide (via IPX) a one-time login vector derived from the shared secret to the visited network (in no way does it provide a shared secret). In the Czech Republic, local providers do not provide roaming to each other, so it can only be used abroad.

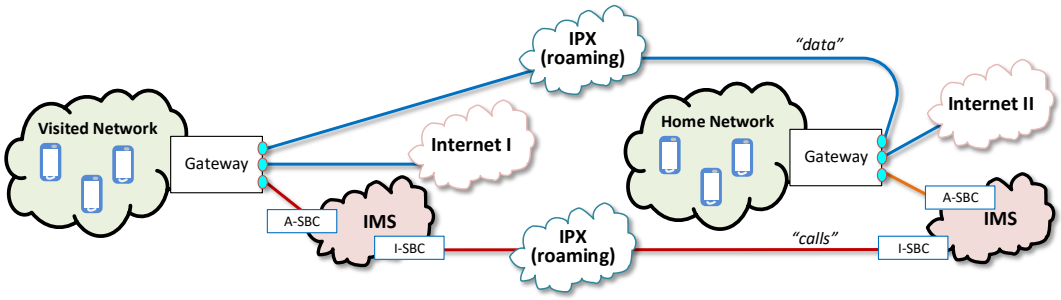



Figure 15.11 *Roaming*

Since we have two different types of networks: mobile network (providing IP connectivity) and IMS (providing multimedia sessions), we must solve the roaming problem at the level of both networks.

### 15.13.1 SIP roaming

In Figure 15.12 the IMS has SBCs at both ends: the A-SBC (Access SBC) and the I-SBC (Interconnect SBC). SBCs are also separated terminologically, mainly because there are different security requirements imposed on them. Roaming is then possible either by direct connection (e.g., optical cable) between two I-SBCs, or via IPX.

The S-CSCF of the visited network detects that the called party is from a foreign network. Through the foreign network's I-CSCF, it forwards the request to the foreign network's S-CSCF, which forwards the request to the called subscriber. Entities of the Medium are shown in dashed lines, because the principle of the architecture implies that the medium can be transmitted outside those IMSs [90].

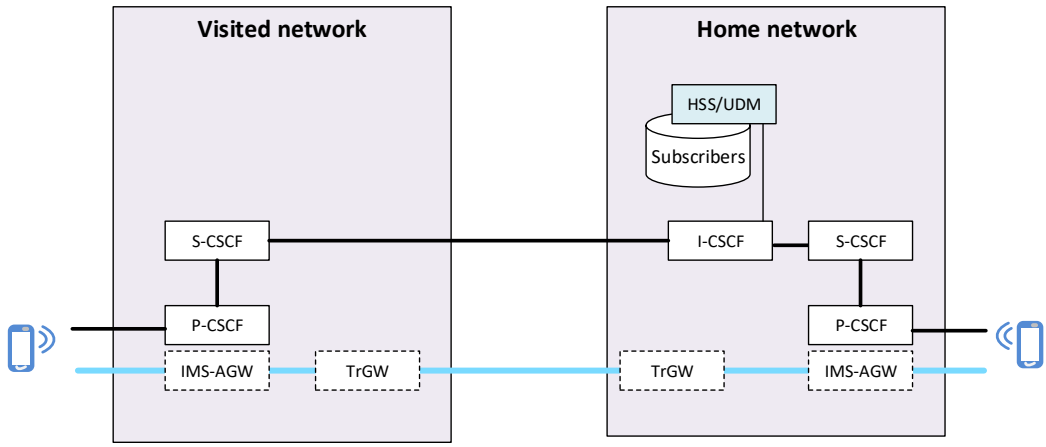



Figure 15.12 *Roaming at the IMS level (simplified)*

### 15.13.2 IP roaming

In the case of ”Data” roaming, the situation is more complicated. Data can be passed directly to the internet by the Visited Network. Or they can be tunneled through IPX to the subscriber’s home network. In the case of tunneling through IPX, the subscriber pays hefty fees for using the IPX network. European regulation is trying within the EU to push providers to pass data to the Internet straight from the visited network and thus reduce fees.

However, it is not true that if ”Data” is sent to the Internet directly from the Visited Network, then IPX would not be used at all. The pictures do not show the Control Plane of Mobile Networks at all, which must transmit, for example, the login vector for the subscriber’s login in the Visited Network via IPX. Billing information is problematic because if the Home Network cannot see what is going through it, then it cannot even measure it – it has to trust the information from the visited network. And if the visited network is outside the EU, then anything is difficult to prove.

## 15.14 SIP registration (second part)

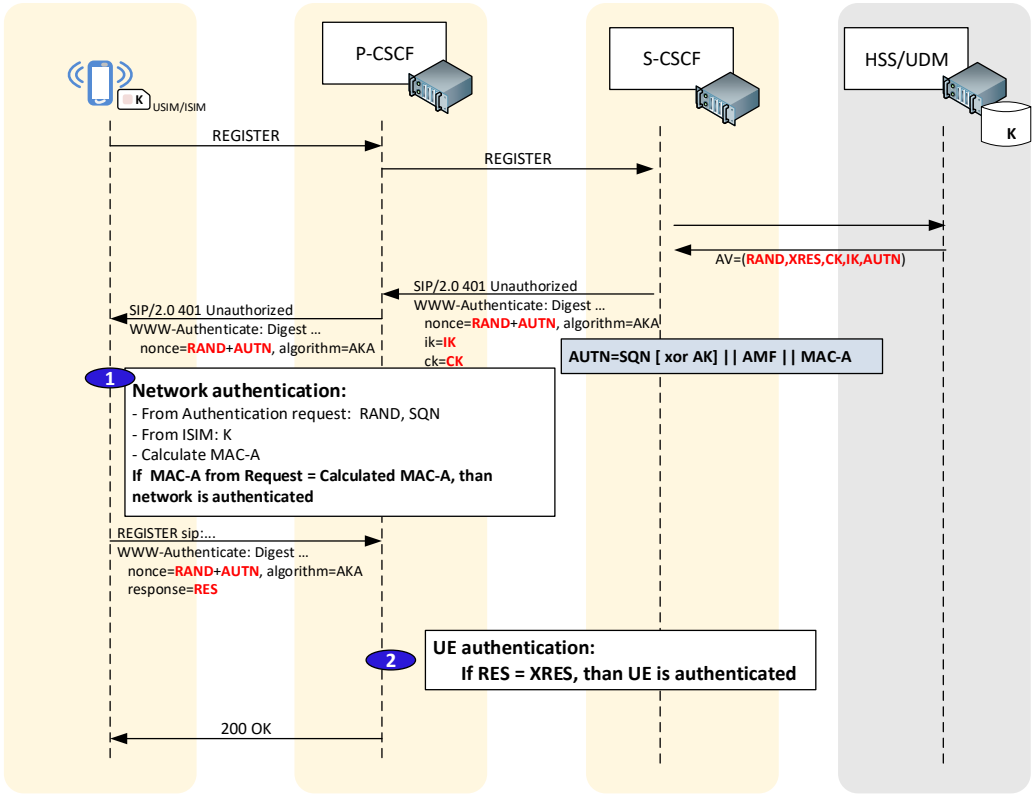



Figure 15.13 *3GPP SIP Registration*

This registration is again based on the challenge-response method using AKA (see section 3.4 AKA). Figure 15.13 shows the process in a simplified way (not all communication entities are shown, but only those important to understand the process).

In general, a subscriber is in a visited network. Therefore, the registration request is sent to the P-CSCF (part of the A-SBC) of the visited network. The visited network does not have cryptographic material for subscriber authentication, so it sends the request to the subscriber's home network. In the home network, the request goes to the S-CSCF entity that is responsible for handling the request. The S-CSCF entity asks for the appropriate cryptographic material of the HSS entity (only the home HSS shares the K secret with the USIM / ISIM subscriber). The HSS generates the AV Authentication Vector and passes it to the S-CSCF, which retrieves the RES from the authentication vector and passes the rest to the response.

Symmetric keys for encrypting communication are derived from `CK` and `IK`.

## 15.15 Lawful Interception

IMS is designed so that communication is secure between the mobile device and the edge of the network, i.e., in the core of the network the communication is not secure, so it is possible to eavesdrop, i.e., perform lawful interception.

Lawful interception in the Czech Republic is defined in §§ 88 and 88a of Act 141/1961 Coll., Criminal Code. Section 88 specifies the circumstances under which wiretapping may be carried out, and Section 88a then specifies the circumstances under which "data on telecommunications traffic, which are the subject of telecommunications secrecy or which are covered by the protection of personal and intermediary data", can be found out.

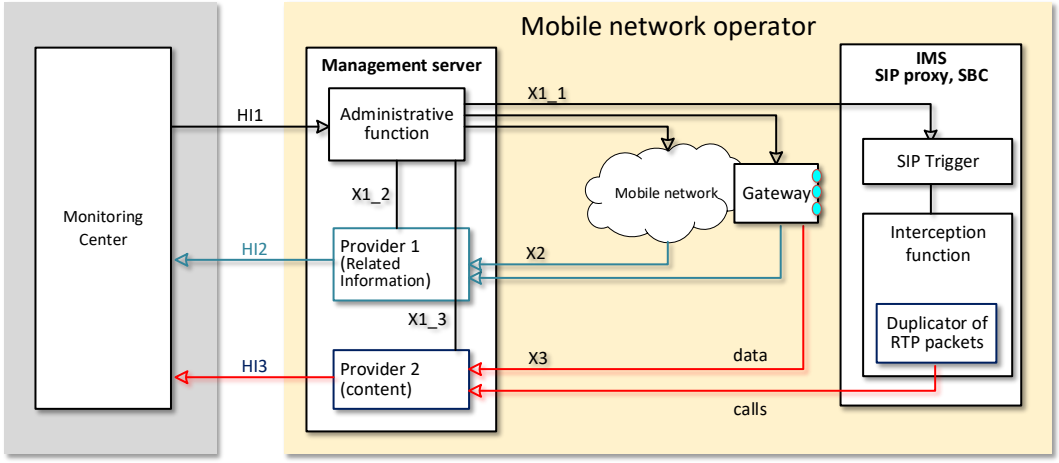



Figure 15.14 *Architecture of lawful interception*

The architecture of legal wiretapping [91] is shown in Figure 15.14 The monitoring center communicates with the operator using the HI1 to HI3 interfaces (Handover Interface). For

these purposes, the operator runs a Management server that transforms communication from the HI1 to HI3 interfaces to the X1 to X3 internal interface. The management server consists of three entities:

- Administration Function (ADMF) – this entity converts the requirements of the Monitoring Center into individual sub-requirements.
- One or more Provider entities that provided intercepted information. Official name of this entity is Law Enforcement Monitoring Facility (LEMF).

Via the HI1 interface, the Monitoring Center enters requests for legal interception, via the HI2 interface it obtains the required information about telecommunications traffic (Intercept Related Information) and via the HI3 interface it then obtains data (media), i.e., call content, IP traffic, etc.

Inside the operator's network, the X1 interface is divided into three types of communication:

- X1_1 – entering requests for individual entities of the operator's network.
- X1_2 – communication with Provision 1.
- X1_3 – communication with Provision 2.

Call interception is a bit more complicated, because to duplicate the call we need to have access to both the SIP protocol and the RTP protocol. If you think about it, this can be done on all entities that use the H.248 interface (e.g., SBC).

# 16 H.248

Multimedia communication (not only in mobile networks) consists of two planes:

- **Control Plane** – this plane deal with the creation, management and termination of a multimedia session. SIP (chapter 15 SIP), SS7 (Signaling System No. 7) [74], etc. protocols are used here.
- **Media Plane** – this plane deals with the transfer of multimedia data. Here, for example, RTP/RTCP, TDM, etc. protocols are used.

Each of these planes is handled by a separate group of network protocols. At first glance, it might seem that they are completely independent. Only in the end user's application (e.g., mobile device) do they come together to jointly provide the desired service to the end user.

However, the problem occurs at the edge of the network. For example, to the SBC, which accepts the multimedia session as if it were an end participant, eventually transforms it and passes it on. So, we have two gateways at the border of the network:

- For the Control Plane, we use a gateway called **Media Gateway Controller (MGC)**.
- For Media Plane, we use a gateway called **Media Gateway (MG)**.

The two gateways must mutually coordinate the session flow, e.g., because the metadata for the Media Plane is transferred in the Control Plane, errors generated in the Media Plane need to be signaled via the Control Plane, etc. If it is a transformation from one protocol family to another (e.g., from SIP/RTP to SS7/E-carrier), then the situation is even more complicated.

The solution is, for example, the H.248 protocol [92]. H.248 mediates the communication between MGC and MG. Among other things, this protocol describes the logical entities, or objects, in the MG that can be controlled by the MGC. The main abstractions used in the connection model are terminations and contexts.

## 16.1 Terminations

termination is described by a number of characterizing properties, which are grouped in a set of descriptors that are included in commands. Terminations have unique identities (**TerminationID**), assigned by the MG at the time of their creation.

Terminations representing physical entities have a semi-permanent existence. For example, a termination representing an E-carrier channel might exist for as long as it is provisioned in the gateway. Terminations representing ephemeral information flows, such as RTP flows,

would usually exist only for the duration of their use.

Ephemeral terminations are created by the command `Add Command`. They are destroyed by the command `Subtract Command`. In contrast, when a physical termination is Added to or Subtracted from a context, it is taken from or to the NULL Context, respectively.

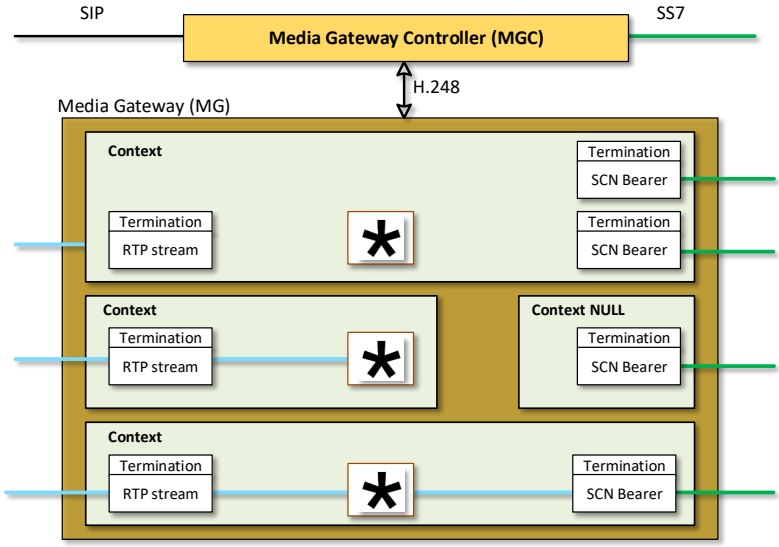



Figure 16.1 *Contexts and Termination - for illustration, the SS7 protocol is shown on the right side, i.e. this time the SBC acts as a Gateway (taken from [92])*

## 16.2 Contexts

The main goal of context is to connect terminations (Figure 16.1). The context describes:

- Forwarding topology.
- Media mixing (if more than two streams are connecting).
- Other connection parameters.

Each termination is added to exactly one context.

There is a special context called the NULL Context which contains all terminations that are not present in any other context and therefore not associated to any other termination. Terminations in this context you can:

- Configure them.
- Change their parameters.
- Look out for events.

Maximum number of terminations in context is based on capabilities of specific **Media Gate-**

**way (MG)**.

## 16.3 Commands

The H.248 protocol uses so-called commands to create contexts and modify its parameters. For example, an Add Command is used to add terminations to contexts. A termination may be removed from a context with a Subtract Command, and a termination may be moved from one context to another with a Move Command. A termination can only exist in one context at a time.

The maximum number of terminations in a context is an MG property. Media gateways that offer only point-to-point connectivity might allow at most two terminations per context. Media gateways that support multipoint conferences might allow three or more terminations per context.

The protocol H.248 can be used to (implicitly) create contexts and modify the parameter values of existing contexts. The protocol has commands to add terminations to contexts, subtract them from contexts, and to move terminations between contexts. Contexts are deleted implicitly when the last remaining termination is subtracted or moved out.

Table 16.1 *H.248 commands*

| Command | Description |
|---|---|
| **Add** | Adds a termination to a context. The Add Command on the first termination in a context is used to create a context. |
| **Modify** | Modifies the properties, events and signals of a termination. |
| **Subtract** | Disconnects a termination from its context and returns statistics on the termination's participation in the context. The Subtract Command on the last termination in a context deletes the context. |
| **Move** | Atomically moves a termination to another context. |
| **AuditValue** | Returns the current state of properties, events, signals and statistics of terminations. |
| **AuditCapability** | Returns all the possible values for termination properties, events and signals allowed by the Media Gateway. |
| **Notify** | Allows the Media Gateway to inform the Media Gateway Controller of the occurrence of events in the Media Gateway. |
| **ServiceChange** | Allows the Media Gateway to notify the Media Gateway Controller that a termination or group of terminations is about to be taken out of service or has just been returned to service. ServiceChange is also used by the MG to announce its availability to an MGC (registration), and to notify the MGC of impending or completed restart of the MG. The MGC may announce a handover to the MG by sending it a ServiceChange Command. The MGC may also use ServiceChange to instruct the MG to take a termination or group of terminations in or out of service. |

# 17 SDP

The Session Description Protocol (SDP) [93] provides information for session description (multimedia teleconferences, voice-over-IP calls, streaming video, or other sessions). SDP conveys media details, transport addresses, and other session description metadata. It may also carry cryptographic material for media security, e.g., RTP/RTCP (chapter 18 RTP/RTCP). The SDP protocol message usually forms the body of the SIP protocol message.

The SDP protocol message is not inserted into the transport protocol. One could even say that the ”transport protocols” for the SDP message are application protocols such as SIP or Real Time Streaming Protocol (RTS) [94].

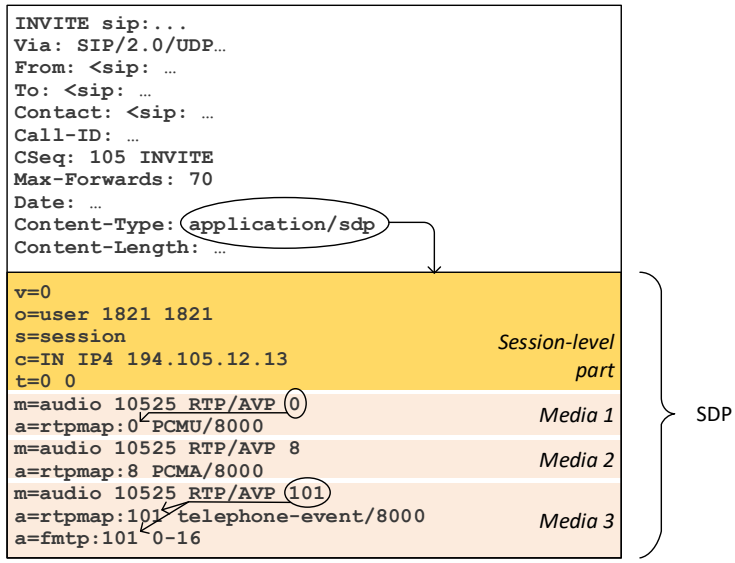



Figure 17.1 *Example of SDP message*

A session is described by a series of fields, one per line. The form of each field is as follows.

<type>=<value> ,

where <type> is a single case-significant character and <value> is structured text whose format depends upon attribute type. Values are typically UTF-8 encoded.

Table 17.1 *Protocol types*

| <type>= | Name | |
|---|---|---|
| v= | protocol version | v=0 |
| o= | originator and session identifier | o=<username> <sess-id> <sess-version><br>Originator of the session (username and the address of the user's host) plus a session identifier, version number and address:<br><username> - originator's name or "- ", if the originating host does not support the concept of the user IDs.<br><sess-id> - is a globally unique identifier for the session<br><sess-version> – version number for this session description |
| s= | session name | Textual session name |
| i= | session information | Textual information about the session |
| u= | URI | Uniform Resource Identifier as used by WWW clients |
| e= | e-mail | contact information for the person responsible for the conference |
| p= | phone number | contact information for the person responsible for the conference |
| c= | connection data | c=<nettype> <addrtype> <connection -address><br>Specify connection data:<br><nettype> - network type<br><addrtype> - address type<br><connection-address> - address of the machine from which the session was created. |
| b= | bandwidth | Denotes the proposed bandwidth to be used by the session or media |
| z= | time zones adjustment | Optional modifier to the repeat-fields (r=) it immediately follows |
| t= | timing | Specifies the start and stop times for a session |
| r= | repeat times | Specifies repeat times for a session |
| m= | medium | m=<media> <port> <proto> <fmt> ...<br>Specifies the media type, format, and transport address:<br><media> - media type<br><port> - transport port<br><proto> - protocol<br>RTP/AVP - RTP over UDP<br>RTP/SAVP - SRTP over UDP<br><fmt> is a media format description. |
| a= | attributes | An SDP message can contain none, one or more attributes. Attributes can appear in both the session description and the flow description.<br>It is used for more detailed specification of media using the RTP protocol, such as sampling frequency, codec, etc. |

The SDP message always consists of two parts:

- Description of the session (always starts with v=), which contains parameters common

to the entire session (i.e., for all multimedia streams of the session).

- Description of none, one or more multimedia streams. The description of each individual multimedia stream always begins with `m=`.

# 18 RTP/RTCP

The Real-time Transport Protocol (RTP) application protocol [95] provides **transport services for applications that require real-time data transfer, for example, audio, video, simulation data, etc.** It uses unacknowledged protocols (e.g., UDP [81]), as a transport protocol because packet loss is a lesser evil than the real-time mode violation that would be caused by requesting a lost packet. For example, it is more pleasant for the telephone subscriber when the lost packet is replaced by artificial noise than if he waits for the lost packet to be repeated and then, for example, the information is quickly played back. RTP therefore uses either UDP or unacknowledged threads of the SCTP protocol as transport protocols.

It should also be noted that RTP transmits data without knowing what it is transmitting, because next layer is inserted between RTP and the application – the so-called media **codec** (from an abbreviation of coder-decoder). A codec is an envelope that packs multimedia data into an RTP stream. The codec solves the problem of how multimedia information is digitized and with which frequency it is sampled. The type and parameters of the used codec are communicated by the SDP protocol and are transmitted in the body of the SIP message. A codec is in principle implemented by a program that encodes/decodes audio/video into the so-called format (e.g., MP3).

From the point of view of the pure TCP/IP network model, RTP is an application protocol (it lies on top of the TCP, UDP, SCTP layer), but from the end-user point of view it is a ”transport” protocol, since the end-user applications put data into the codec envelope and only this envelope is put into RTP.

RTP supports both unicast and multicast IP addresses.

RTP provides the following support for real-time transport among others:

- **Jitter** elimination with the help of buffers.
- **Synchronization** of audio and video streams belonging to the same multimedia session.
- **Multiplexing** audio and video streams in one session.
- **Transcoding** of multimedia stream.

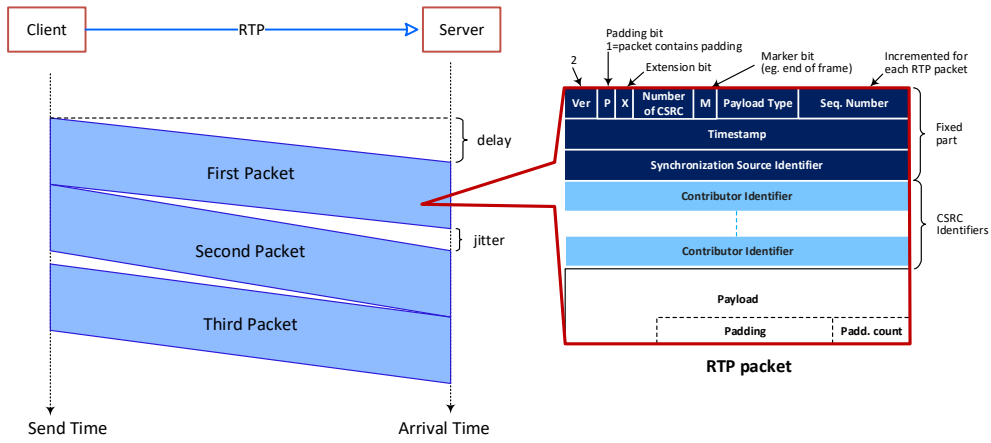



Figure 18.1 *RTP communication and RTP packet*

Figure 18.1 shows the RTP communication between the client and the server. The RTP packet is also shown, which contains the following items:

- The **Version** fields have a value of 2.
- **Padding bit** (P) – if the padding bit is set, the packet contains one or more additional padding bytes at the end, which are not part of the payload. The last octet of the padding contains a count of how many padding octets should be ignored, including itself. Padding may be used by block encryption algorithms.
- **Extension bit** (X) – if the extension bit is set, the fixed header must be followed by exactly one header extension.
- **CSRC count** contains the number of CSRC identifiers that follow the fixed header.
- **Marker bit** (M) is intended to allow significant events such as frame boundaries to be marked in the packet stream.
- **Payload type** (PT) – field that identifies the format (including coding) of the RTP payload and determines its interpretation by the application.
- **Sequence number** increments by one for each RTP data packet sent, and may be used by the receiver to detect packet loss and to restore packet sequence.
- **Timestamp** reflects the sampling instant of the first octet in the RTP data packet. The sampling instant MUST be derived from a clock that increments monotonically and linearly in time to allow synchronization and jitter calculations.
- **Synchronization source identifier** (SSRC) uniquely defines the multimedia source. The SSRC identifier is a randomly chosen value meant to be unique within a particular RTP session.
- **Contributing source identifier list** (CSRC) contains a list of SSRC streams that have been combined using the so-called RTP mixer. The RTP mixer inserts the list of SSRC sources it has mixed at the beginning of the RTP packet. For example, the RTP audio con-

ference mixer mixed the audio streams of the individual participants who had their microphones on.

Table 18.1 *Payload Types registered for RTP*

| Payload Type | Name | A/V | Clock Rate (Hz) |
|---|---|---|---|
| 0 | PCMU | A | 8000 |
| 3 | GSM | A | 8000 |
| 4 | G723 | A | 8000 |
| 5 | DVI4 | A | 8000 |
| 6 | DVI4 | A | 16000 |
| 7 | LPC | A | 8000 |
| 8 | PCMA | A | 8000 |
| 9 | G722 | A | 8000 |
| 10 | L16 | A | 44100 |
| 11 | L16 | A | 44100 |
| 12 | QCELP | A | 8000 |
| 13 | CN | A | 8000 |
| 14 | MPA | A | 90000 |
| 15 | G728 | A | 8000 |
| 16 | DVI4 | A | 11025 |
| 17 | DVI4 | A | 22050 |
| 18 | G729 | A | 8000 |
| 25 | CelB | V | 90000 |
| 26 | JPEG | V | 90000 |
| 28 | nv | V | 90000 |
| 31 | H261 | V | 90000 |
| 32 | MPV | V | 90000 |
| 33 | MP2T | AV | 90000 |
| 34 | H263 | V | 90000 |

## 18.1 RTCP

RTP provides transport layer for real time applications but does not provide any mechanism for error and flow control, congestion control, quality feedback and synchronization. For that purpose, the Real-Time Control Protocol (RTCP) is intended as a companion to RTP to provide end-to-end monitoring and data delivery. RTP and RTCP are designed to be independent of the underlying transport and network layers. The protocol supports the use of RTP-level transcoding and mixers.

RTCP is based on the periodic transmission of control packets to all participants in the session, using the same distribution mechanism as the RTP. RTP typically uses even numbered

UDP ports. RTCP then uses the next higher odd number for the UDP port number. It is recommended that the fraction of the session bandwidth allocated to the RTCP is 5%. RTCP is responsible for three main functions:

- **Gathers statistics** on quality aspects of the media distribution during a session and transmits these data to the session media source and other session participants. Such information may be used by the source for adaptive media encoding and detection of transmission faults.
- **Carries a unique identifier** for an RTP resource called a canonical name or CNAME. Because the SSRC identifier can change if a conflict is detected or the program is restarted, receivers require the CNAME to keep track of each participant.
- **Conveys the identity** of the sender for displaying in the user interface.

Figure 18.2 shows how the individual packets are used during monitoring: the sender sends an RTP packet and the RTCP passes Sender info (information about the sender), which RTCP inserts into the SR packet and sends. The RTP receiver evaluates the reception statistics and the RTCP hand over Receiver Info (information for the originator of the RTP stream), which sends them using an RR packet. The originator receives the RR packet and passes the information in the form of RR feedback to the originator of the RTP frame.

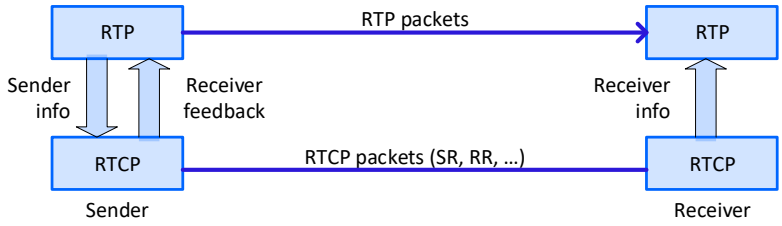



Figure 18.2 *RTP and RTCP communication*

RTCP specificity: several RTCP packet types carry a variety of control information.

## 18.2 SRTP

Secure Real-time Transport Protocol (SRTP) [96] provides a framework for encryption and message authentication of RTP. SRTP takes RTP packets, secures them and then sends them as a secured SRTP packet (Figure 18.3) to the receiving side. The recipient proceeds in the reverse order.

Each SRTP stream requires the sender and receiver to maintain cryptographic state of information. This information is called the ”cryptographic context”. SRTP uses two types of keys: session keys and master keys:

- The **session key** is used directly for encryption or message authentication.
- The **master key** is a random bit string from which session keys are derived in a cryptographically secure way. The master key(s) is negotiated outside of SRTP (externally, e.g., by SIP).

Table 18.2 *RTCP Payload Types*

| Payload Type | Name |
|---|---|
| 200 | **SR** – Sender report packets are intended for transmission and reception statistics from participants that are active senders. SR are regularly sent by an entity that is both a receiver and transmitter. It contains both information about the stream being sent to them, as well as any information about received streams. Time stamps in the Originator Information Block are used, for example, for the synchronization of streams belonging to the same session (e.g., audio and video synchronization). The reporting block for the originator, in turn, contains statistics on lost packets and jitter. This, in turn, is useful for the originator of the stream, for example, based on these statistics, they can reduce the quality of the provided stream content, thus probably achieving a lower error rate. |
| 201 | **RR** – Receiver report, for reception statistics from participants that are not active senders and in combination with SR for active senders reporting. |
| 202 | **SDES** – Source Description packets send a transmitter to advertise information about itself. The receiver then displays this information to the participants.<br>In an SDES packet, each type of information has its own entry type:<br>• CNAME – Canonical end-point identifier unique among all participants like user@host.<br>• NAME – User name of source.<br>• EMAIL – E-mail address.<br>• PHONE – Telephone number.<br>• LOC – Geographic location.<br>• TOOL – Name of application generating the stream.<br>• NOTE – Transient message describing the current state of the source.<br>• PRIV – Private experimental or application-specific extensions.<br>• END – End of SDES list. |
| 203 | **BYE** – Identifies end of participation. |
| 204 | **APP** – Application specific functions. |
| 207 | **XR** – RTCP Extension. |

An example of provisioning of the master key is the SDP protocol, which transfers the master key in a SIP payload, in the SDP message in ”a=crypto:” attribute in unsecured form. There are more sophisticated methods, for example, described in [86].

An SRTP packet is a secured RTP packet with the addition of a footer containing two items: SRTP MKI and Authentication Tag (Figure 18.3). Integrity is ensured for the entire RTP packet, encryption only for payload. Meaning of the remaining items:

- **SRTP MKI (Master Key Identifier)** identifies the master key, which is used to derive session keys.
- **Authentication Tag** contains authentication data that secures the serial number of the packet in particular and prevents replay attack.

Note that the SRTP MKI and Authentication Tag are not secured.

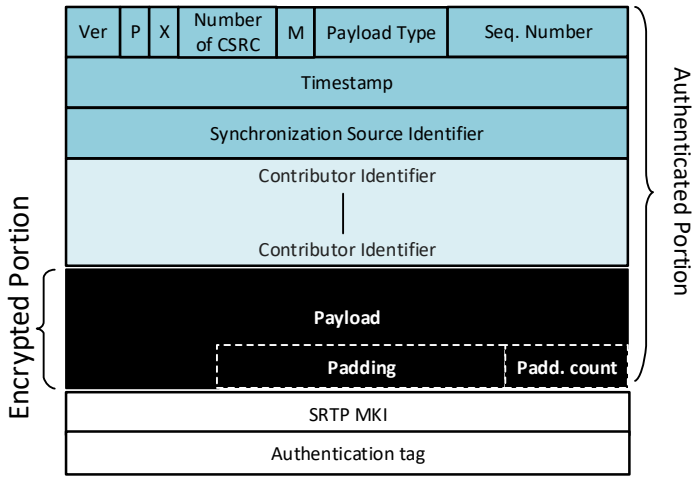



Figure 18.3 *SRTP packet*

## 18.3 SRTCP

SRTCP (Secure RTCP) provides similar security for RTCP as SRTP provides for RTP. Unlike SRTP, the SRTCP footer also contains an SRTCP index entry, which contains the sequence number of the SRTCP packet.

SRTCP (Secure Real-Time Transport Control Protocol) is an extension of the standard RTCP (Real-time Transport Control Protocol) and is used to ensure secure real-time data management. SRTCP, like SRTP (Secure Real-Time Transport Protocol), is used to encrypt and ensure authenticity of the control of multimedia streams in VoIP networks, video conferencing and other real-time applications.

Basic SRTCP Features:

1. **Data encryption** - SRTCP encrypts RTCP control packets to prevent unauthorized access and protect data from eavesdropping.
2. **Data authentication** - SRTCP uses hashing mechanisms to verify the authenticity of

RTCP packets, ensuring that the data have not been modified during transmission.

3. **Data integrity** - The protocol ensures the integrity of control data, which prevents it from being modified or tampered with.
4. **Replay Attack Protection** - SRTCP includes replay protection, which prevents previously captured packets from being reused.

SRTCP packets include the following main components:

- **Packet Type** - Indicates the RTCP packet type.
- **Packet Length** - Defines the length of the packet.
- **Secret data (Encrypted Data)** - Encrypted contents of the RTCP packet.
- **Authentication Tag** - Used to verify the authenticity and integrity of the package.

Benefits of Using SRTCP:

- **Confidentiality** - Packet encryption protects control information from eavesdropping.
- **Authenticity and integrity** - Authenticity and integrity mechanisms protect data from tampering.
- **Compatibility** - SRTCP is compatible with SRTP, allowing it to be used on the same networks and applications.

# 19 Mail

To this day, the RFC 821 [97] and RFC 822 [32] standards from 1982 remain the foundation of email communication on the Internet. The former defines SMTP, while the latter specifies the format of an email message. Although these standards have since been superseded by newer ones—RFC 5321 [98] and RFC 5322 [99]—the term ”**RFC 822-type message**” is still commonly used to refer to the message format, even in many current standards. Officially, however, RFC 5322 recommends using the term ”**Internet Message Format**” instead of ”RFC 822 message.”

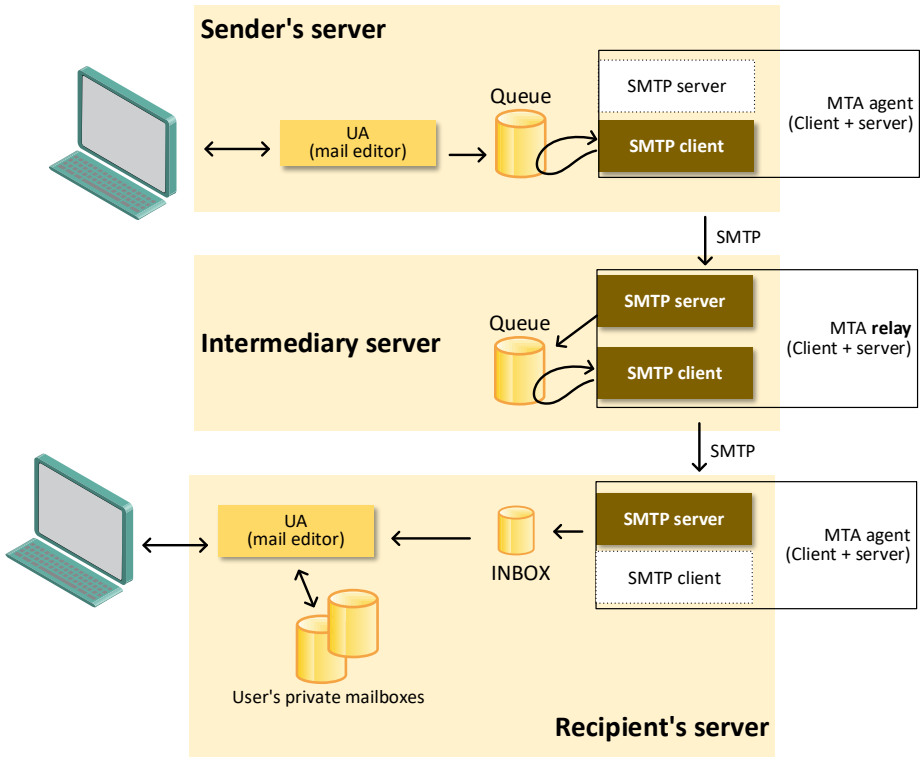



Figure 19.1 *SRTP packet*

Back in the 1980s (when ”Internet” was still spelled with a capital ”I”), users worked at terminals from which they launched mail User Agents (UAs)—programs that allowed them to

receive and send electronic mail. However, the mail UAs of that time had nothing to do with network communication; they were essentially specialized text editors that enabled users to compose and read email messages. Sending a message did not involve any immediate network activity—it simply meant saving the message to the local mail queue. This queue was then processed by a local **Message Transfer Agent (MTA)**, which forwarded the message to a remote MTA using the Simple Mail Transfer Protocol (SMTP see chapter 20 SMTP).

The remote MTA receives the message and checks whether it is addressed to its local user. If it is not intended for any of its local users, it again stores the message in a mail queue served by its MTA, which tries to deliver the message further towards the recipient (we call such an MTA a Relay MTA). If the recipient of the delivered message is a local user, the MTA saves the received message in the local user's mailbox. The user can then view the received message (email) using his UA.

As a rule, the user has one "system" mailbox INBOX in the system, where the MTA stores his incoming mail. Physically, the mailbox is not called **INBOX** as a file, but, for example, in UNIX, it has the same name as the user's name (e.g., `/var/spool/mail/$USER`).

The user can also set up private mailboxes to save incoming and outgoing mails from the INBOX. Private mailboxes are not served by the MTA. The goal is to get users not to archive incoming mail in the "system" INBOX, but to either delete read messages or move them to their private mailboxes, but that is none of the MTA's business.

Note that in Figure 19.1 the whole situation appears as if we had a special UA for sending mail and another UA for working with INBOX (for "receiving" mail). In fact, the user always has universal software at his disposal, which has the ability to both receive and send mails (i.e., save/read to/from the mail queue). In addition, UA usually makes other services available to the user, such as the already mentioned handling of private mailboxes.

As users gradually migrated from the server terminals to PCs, they began to require UA on their PC. At its core, emails remained unchanged even after this shift. Only network communication between UA and MTA has been added. While the SMTP protocol can be used again without any problems when sending mail from a PC, two competing protocols have been created for receiving mail: the simple POP3 (chapter 21 POP3) and the robust IMAP (chapter 22 IMAP4).

By the term Mail server, we no longer mean only the SMTP agent (MTA), but the entire ecosystem: SMTP client, SMTP server, queues, mailboxes, and now also IMAP4 and POP3 servers.

The user has his INBOX on the mail server, i.e., in terms of the SMTP protocol, the mail server is the target system. The task of the POP3 and IMAP4 protocols is to pick up mails from INBOX on server to the user's **mail box on her/his PC**.

The user thus downloads the message to his PC using one of the POP3/IMAP4 protocols. Most likely he has a more than one private mailbox on his PC as well. Important is that one of them is usually called Inbox and messages from the mail server are implicitly downloaded into it.

## 19.1 Webmail

Today, the trend is to place applications in a web environment with the goal that the end user does not have to have anything other than a common web browser at his disposal. UA's web solution is **webmail** (Figure 19.2). On the side of the mail server, a web server runs over the client mailboxes, whose scripts can not only send mail, but also handle mailboxes in a similar way to any other UA. HTTPS, not SMTP, POP3 or IMAP4 protocols are then used between the PC and the mail server. From the point of view of this chapter, however, webmail has one major disadvantage. In the case of webmail, it is not possible to effectively ensure end-to-end security of e-mail (e.g., S/MIME - see section 23.2 S/MIME). The reason is simple. In the case of webmail, the postal envelope is created/unpacked on the mail server (using scripts running on the web server), whereas the user's private key required for creating a digital signature (or decrypting messages) is available on the user's PC, i.e., generally in a different physical location.

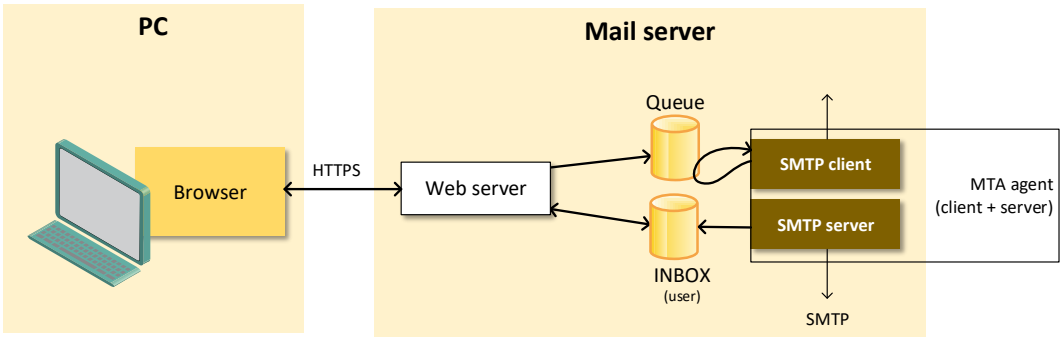



Figure 19.2 *Webmail*

## 19.2 Mail delivery

The fundamental feature of electronic mail is that its **delivery is not guaranteed**. This problem is solved by a delivery notification. We have three types of delivery notification:

- The message was delivered to the recipient's SMTP server.
- The message was received and displayed by the recipient's UA.
- The recipient's UA verified the sender's digital signature.

However, the email delivery notification has two significant limitations:

- The recipient is not obliged to generate a notification message. Therefore, it depends on his discretion whether the notification message will be generated.
- Delivery notification takes place via electronic mail, the delivery of which, as we already know, is generally not guaranteed.

## 19.3 Securing e-mails

We will also deal with the securing e-mails, which can be viewed from two diametrically different perspectives:

- **Point-to-point security** (Security of mail transport between two neighboring systems), i.e., between neighboring MTUs (e.g., SMTP over TLS) or between the PC and the Mail Server, i.e., POP3 over TLS or IMAP4 over TLS.
- **End-to-end security**. When a message is protected all the way through the Internet from the sender's UA to the recipient's UA, i.e., S/MIME.

There is no way to say which security method is worse or better. Each of them will be suitable for a different situation. Ultimately, it may happen that we use both methods of security at the same time. Let us imagine the following situation: We create a message, which we are obliged to add a digital signature to make it acceptable to the recipient (end-to-end security). But we have to send it through some SMTP server. However, the SMTP server prevents itself from distributing spam, so it requires authentication with a permanent password. But the password can be intercepted by an attacker, which the SMTP server prevents by securing the connection using SSL/TLS.

## 19.4 Internet Message Format

The mail consists of a header and a body. The **header** is separated from the **body** by one **blank line**, i.e., the characters CR LF CR LF. [1]. The message header contains header fields consisting of ASCII characters only. The body of the message does not have to be made up of ASCII characters only if the client and the server agree on this in advance.

This characteristic of an email message has been true for years. The pressure from non-Latin characters writing nations is so great that they have pushed for "**Internationalized Email**". Currently, it is a set of standards RFC 6530 [100], RFC 6531 [101], RFC 6532 [102] and RFC 6533 [103], which allow the use of non-ASCII characters in the header, including the email address. However, the problem is that the software may not support these standards. It will probably take some time before mail server operators start supporting these standards even in countries that use the Latin alphabet, if it ever happens.

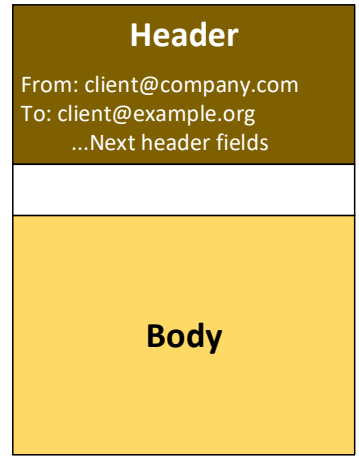



Figure 19.3 *Internet message format ("RFC 822 format")*

The header consists of individual header fields. The header field starts with field name, followed by a colon (:) and the field body, which may contain parameters. The header field ends with the end of the line, i.e., the characters CR and LF.

[1] CR means (**Carriage Return**), LF means (**Line Feed**). Characters CR and LF are commonly being shortened to CRLF.

White spaces, i.e., spaces (SP) or horizontal tabs (HTAB) can be arbitrarily inserted between individual parts of the header. The header field can continue to the next line. However, the next line must start with a white space(s). If a comment were to be used, then it would be enclosed in parentheses `()`.

The header and body of the message consist of lines terminated by the characters CR and LF. Line length including CR and LF characters should not exceed 80 characters. However, it cannot exceed 1000 characters.

Table 19.1 *SMTP Header fields*

| **Header fields** | **Meaning** |
|---|---|
| `From:` | The author(s) of the message, that is, the mailbox(es) of the person(s) or system(s) responsible for the writing of the message. |
| `Sender:` | The mailbox of the agent responsible for the actual transmission of the message. If the field contains more than one mailbox specification, then the sender field, containing the field name "Sender", must appear in the message. |
| `Date:` | The date of the message. |
| `Reply-To:` | The address(es) to which the author of the message suggests that replies would be sent. |
| `In-Reply-To:` | We are responding to your message: Identification of the original message. |
| `To:` | The address(es) of the primary recipient(s) of the message. |
| `Cc:` | *Carbon Copy:* The addresses of others, who are to receive the message, though the content of the message may not be directed at them. |
| `Bcc:` | *Blind Carbon Copy:* The addresses of recipients of the message whose addresses are not to be revealed to other recipients of the message. |
| `Message-Id:` | The unique message identifier that refers to a particular version of a particular message. |
| `Keywords:` | Keywords characterizing the content. |
| `Subject:` | A brief description of the content of the message. |
| `Comments:` | Contains any additional comments on the text of the body of the message. |
| `Resent-` | When forwarding a message automatically (e.g., returning an undeliverable message), the string Resent- is inserted before the original header enters. For example: `Resent-From:` or `Resent-Cc:`, etc. |
| `Received:` | Each mail gateway (intermediate mail server) through which the message passes adds this header to the beginning of the mail. So, if we read the Received: headers from bottom to top, we find out the entire route through which mail servers the message went. In this header entry, parameters can appear, among others:<br>`From` – the computer from which the mail was received.<br>`By` – computer that received the mail.<br>`With` – network or mail protocol.<br>`Id` – identification of the received email message (e.g., in the mail queue).<br>`For` – for whom the message is intended (for example, if the addressee is a distribution list, then the original addressee, i.e., the distribution list, will be preserved here). |

## 19.5 Mail address

In the past, more complicated addresses containing the % character or the ! character were used, but these were gradually abandoned. Email addresses used today consist of only two parts:

- **Local part** that identifies the recipient's mailbox. It can be a person's mailbox, for example, a distribution list, a mailbox automatically processed by a script, etc. However, the Destination mail server can be a mail gateway to another mail system, then the local part can be interpreted as an address in that other mail system. An example is historical mail gateways that accepted, for example, the local part (mailbox) of `a%b.com` (from the email address `a%b.com@c.com`), rewrote the percentage (%) to `@` and sent the email to the recipient `a@b.com`.
- **Domain identification** (DNS names that may only contain ASCII characters, numbers and dashes.

Then both parts are separated by the `@` character. For example:

```
Bob.Novak@example.org
```

The email address must consist exclusively of ASCII characters (we will mention non-ASCII characters in the next paragraph).

One or more addresses can appear in individual header fields. In the case of multiple addresses, individual addresses are separated by commas.

However, the syntax of a specific email address in the header fields is richer. The address given in the header fields can consist of two parts:

- The string displayed by the user. It usually contains the user's name, as it is used in non-electronic communication.
- An email address of the form mailbox@domain enclosed in angle brackets <>.

**Example:**

```
To: Dr. Jan Sip <Jan.Sip@example.org>
```

But Dr. Ján Šíp would like to be displayed as Dr. Ján Šíp and not Dr. Jan Sip. So, how to express non-ASCII characters in the header fields, when the email header fields can contain only ASCII characters?

Non-ASCII characters should never appear in a message header fields. If such a character occurs in the header field, then the message might reach the recipient without a problem, the message might be returned by some server on the way, or it might be lost.

RFC-2047 [104] addresses the issue of how to supply non-ASCII characters to header fields. The problem consists of two parts:

- What does an ASCII character represent? For example, hexadecimal character F8 can represent ř in one alphabet and š in another. We need to specify the character set (charset).
- How to encode it to ASCII, for example, Base64 or Quoted-printable.

The header fields string syntax containing non-ASCII characters is:

`=?charset?encoding?encoded-string?=  ,`

where `encoding` is `B` for Base64 or `Q` for Quoted-printable.

If the sender wants to enter the name Ján Šíp with diacritics in the header field, then:

`To: =?iso-8859-2?B?SuFuIKntcA==?= <Jan.Sip@example.org>  ,`

where:

- Charset is iso-8859-2.
- Encoding is B, i.e. Base64.
- Encoded string is:

  `Base64(Ján Šíp) = SuFuIKntcA==`

In the following example, Quoted-printable coding is also used in the header field From:. In this encoding, ASCII characters are left in place and non-ASCII characters are preceded by an "=" character followed by the hexadecimal value of the character in the used encoding.

**Example:**

```
Received: from prometheus.terms.cz (prometheus.terms.cz [81.90.160.70])
   by tomcat.bf.jcu.cz (Postfix) with ESMTP id 041B9A88308
   for <Libor.Dostalek@bf.jcu.cz>; Sun,  9 Jan 2005 17:20:20 +0100 (CET)
Received: from dostalek ([81.90.165.93])
   by prometheus.terms.cz (8.12.3/8.12.3/Debian-7.1)
   with SMTP id j09GKJMB002078
   for <Libor.Dostalek@bf.jcu.cz>; Sun, 9 Jan 2005 17:20:19 +0100
Message-ID: <000901c4f663$eb0f4c20$7002a8c0@bf.jcu.cz >
From: =?iso-8859-2?Q?Libor_Dost=E1lek?= <Libor.Dostalek@bf.jcu.cz>
To: =?iso-8859-2?B?SuFuIKntcA==?= <Jan.Sip@firma.cz>
Cc: =?iso-8859-2?Q?Libor_Dost=E1lek?= <Libor.Dostalek@bf.jcu.cz>
Subject: Email subject
Date: Sun, 9 Jan 2005 16:57:27 +0100
```

When parsing an email, you divide the email header fields into `Received:` header fields and other header fields. `Received:` header fields are always added to the top of the message by the mail servers through which the message passes. That is why order matters for `Received:` header fields. The `Received:` header field added by the first SMTP server is the lowest. Header field `Received:` above it is a header field added by next intermediate SMTP server, etc.

If we read the Received: header fields from bottom to top, we can see that the mail was sent from the dostalek computer with the IP address 81.90.165.93. It was first received by the MTA prometheus.term.cz, which forwarded it to the destination MTA (destination SMTP server) tomcat.bf.jcu.cz.

**Note:** IP addresses are given in square brackets. In the case of IPv6 addresses, the string "IPv6:" is inserted before the IPv6 address.

**Example:**

`user@[IPv6:2001:db8:1ff::a0b:dbd0]`

## 19.6 Internationalized email adress

As already mentioned, standards RFC 6530 [100], RFC 6531 [101], RFC 6532 [102] and RFC 6533 [103] allow the use of non-ASCII characters in the header, including the email address. Standards allow the use of characters encoded in UTF-8, which supports not only all languages, emojis, but also marks of directionality of writing. Directionality marks are especially important in languages, where text is written from right to left, but, for example, numbers are written from left to right (e.g., Arabic or Hebrew). Marks of directionality of writing are not used if the font direction is implied by the context.

The email address can be, for example,

**अजय@डाटा.भारत**

(@ separates the local part and the domain)

If UTF-8 characters are allowed in the header, then MIME transformation of non-ACII characters is not required. On the other hand, the main problem is with the email address. The standards allow the use of UTF-8 characters both in the local part of the email address and in the domain.

Ideally, DNS would support UTF-8 characters in domains and mail systems would support UTF-8 characters in the local part of the mail address (mailbox).

But we do not live in an ideal world, and some mail systems still do not support UTF-8 in the header fields. The question is how to transform a UTF-8 e-mail address to ASCII characters, i.e., downgrading e-mail address.

First of all, this problem is solved by DNS for Domains RFC 3492 [105] and RFC 5890 [106], which converts non-ASCII parts of domain names using Punycode encoding.

**Example:** domain `www.lepší.tv` is in Punycode converted as `www.xn--lep-tma39c.tv`

The situation with the local part of the postal address (mailbox) is worse. It is also possible to transform this using Punycode. But only the sender's SMTP server or recipient's SMTP server can do this, not intermediate SMTP servers. The reason is that only the target SMTP server can interpret the local part (mailbox). As already mentioned, the destination server may be a gateway to another mail system and the local part may interpret it as a fully qualified address in that foreign mail system.

# 20 SMTP

In the case of paper mail, we entrust our shipment to a postal service provider, which gradually passes the shipment on to other providers so that it is delivered to the recipient's post office on the other side of the world. Similarly, we entrust electronic mail to our Internet provider using the **Simple Mail Transfer Protocol (SMTP)**, which forwards it on and on to the recipient's "mail server", where it is inserted into the addressee's mailbox (INBOX).

The recipient then retrieves the mail from his mailbox on the mail server. Which can be done locally or over the Internet using protocols POP3 (chapter 21 POP3), IMAP4 (chapter 22 IMAP4) and the mentioned Webmail (i.e., HTTPS).

The most basic mail protocol on the Internet is Simple Mail Transfer Protocol (SMTP). SMTP is a client-server architecture protocol. The protocol is called SMTP, where the first S means "simple". However, today's protocol is no longer very simple, and the configuration of the SMTP server does not have to be simple at all. This is because the SMTP protocol was gradually extended by extensions. We do not have a new **ESMTP** (extended SMTP) protocol, but just SMTP extended by a number of extensions. The principles still remain the same:

- Mail is sent by the SMTP client to the SMTP server.
- Communication runs in "seven-bit" ACSII code. If both parties agree, the data transfer can also take place in "eight-bit" format.

The client sends commands (`EHLO`, `MAIL`, `RCPT`, `DATA`, ...) and the server returns responses. The response consists of a three-digit machine-readable code and human-readable text.

Figure 20.1 schematically shows the fundamental SMTP handshake. The client establishes a TCP connection (1) and the server presents itself with its banner (2). Now the client sends an `EHLO` command containing the DNS name of the client's Computer (3), after which the server also introduces itself by its DNS name and sends a list of extensions it supports. Our server supports extensions `SIZE`, `VRFY`, `ETRN`, `AUTH`, `DSN` and `8BITMIME` (4) - see table 20.2.

Now SMTP client can send the email. First, the client uses the MAIL command to tell the server who is the sender of the mail (7), and the server will answer whether this sender is acceptable for it or not (8).

Mail can have multiple recipients. For each recipient, regardless of whether the recipient is marked as `To:`, `Cc:` or `Bcc:`, the client sends one `RCPT` command and the server responds if recipient is acceptable to it or not.

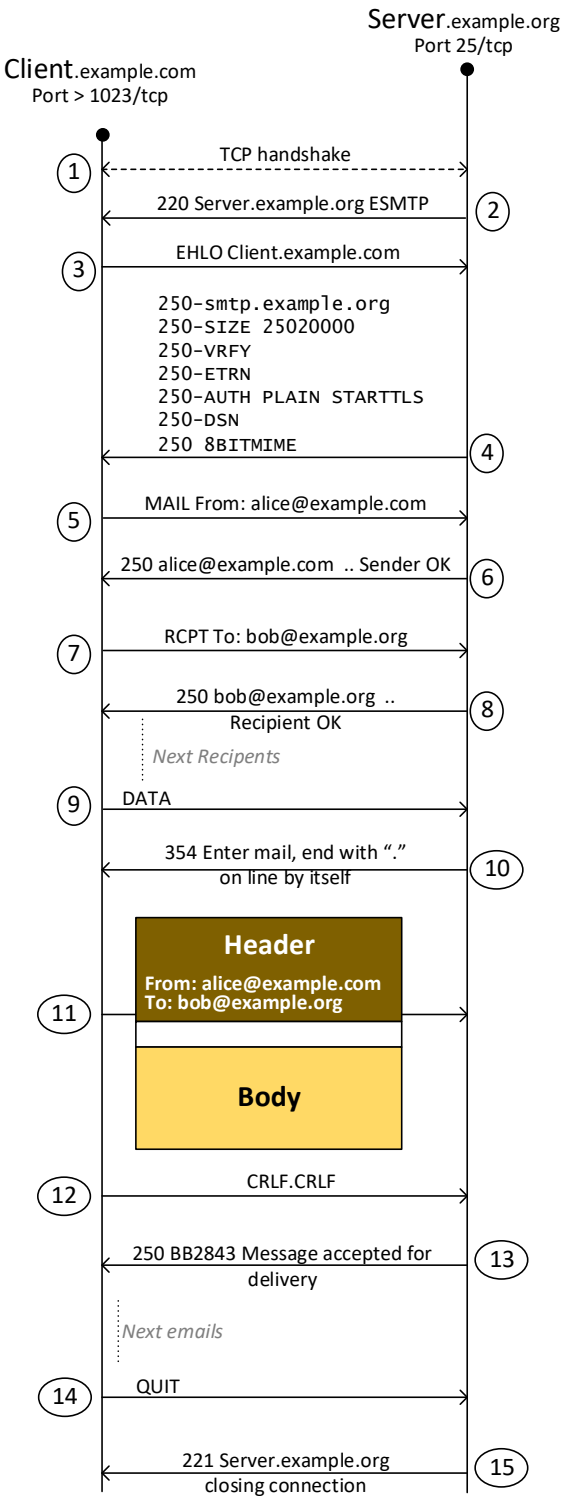



Figure 20.1 *Internet message format (ESMTP handshake)*

The `RCPT` command is followed by the `DATA` command, with which the client tells the server that it has already switched to mail transmission (9). The server warns the client that the mail ends with a line containing only a period (10). Next is the transmission of mail (11) terminated by a new line (CR and LF), a period and a new line (CR and LF). The server then returns the identification under which the mail was stored in the mail queue (13). This can be followed by the transmission of further emails.

The SMTP client terminates the communication with the `QUIT` command (14).

**Note:** The sender is listed both in the From: parameter of the MAIL command and in the From: header field of the same name in the mail header. In the same way, the recipients are listed both in the To parameter of the `RCPT` command and in the email header fields `To`, `Cc` or `Bcc`. Mail message, including its header, is as if wrapped in an SMTP envelope consisting of the commands `MAIL`, `RCPT`, `DATA`, etc. (not to be confused with an electronic envelope in PKI!). From the point of view of the SMTP protocol, the important information is precisely in the SMTP envelope. The information in the mail header is secondary to SMTP communication. For example, `Bcc` recipients will not appear in the message header at all, whereas they must be in the SMTP envelope, otherwise the mail would not be delivered to them at all. The mail header is intended for the addressee, while the content of the SMTP envelope is important for the SMTP server. The contents of the SMTP envelope can more or less be found in the SMTP server log.

Table 20.1 *SMTP commands*

| Command | Meaning |
|---|---|
| EHLO | Extended HELLO is used initiate the communication and to identify the SMTP client to the SMTP Server (contains the fully-qualified domain name of the SMTP client, if one is available). |
| HELO | The client starts the dialogue with the EHLO command, if the server does not support any extensions, then the EHLO command can be rejected and the client continues with the HELO command having the same meaning as the EHLO command. Reason for this is ensuring backward compatibility. If the server responds negatively to the EHLO command, the client can think: "Hmm, this is some old server, I will continue like it was done in the old days, when there were no SMTP extensions." |
| MAIL | The command initializes the mail transaction. The sender's address is specified as a parameter. |
| RCPT | Recipient specification (this command is repeated for each recipient). |
| RSET | The current mail translation will be aborted. Any stored sender, recipient, and mail data must be discarded and all buffers and state tables must be cleared. |
| DATA | A request to send email (i.e., data from the point of view of the SMTP). |
| NOOP | Empty command. |
| QUIT | Request to terminate the connection. |
| EXPN, VRFY, HELP | Commands supported by the standard but not used today to prevent them from being abused. |
| SIZE | Specifies the maximum length of mail that the server is willing to accept. |
| ETRN | In addition to standard commands, the server also supports the ETRN command. With the ETRN command, the client signals to the server that the server should go through its mail queue and try to re-deliver all mail directed to the client's system (the system from which the ETRN command was issued). |
| VERB | In addition to standard commands, the server also supports the VERB command. The VERB (Verbose) command causes the server to start a detailed log of the communication between the client and the server. The command is useful for troubleshooting. |

## 20.1 Internationalized mail support

An SMTP server signals that it supports internationalized mail by supporting both following extensions:

- 8BITMIME.
- SMTPUTF8.

It looks simple, but the devil is, among other things, in the similarity of some characters in different languages, which is a great opportunity for attackers.

Table 20.2 *SMTP Extension*

| Extension | Meaning |
|---|---|
| DSN | Delivery Status Notification (DSN) – support for notification of email delivery to the addressee's SMTP server. |
| 8BITMIME | The transmitted data do not have to be ASCII only (it only applies to the body of the mail). The SMTP client specifies with the 8BITMIME parameter in the MAIL command that it will send mail whose data part is eight bits. |
| BINARYMIME | Transmitted data can also be Binary. |
| CHUNKING | In addition to the standard commands, the server also supports the BDAT command (substitute for the DATA command), which can be used to send large volumes of data. The BDAT command allows you to send data in chunks as well. |
| PIPELINING | Figure 20.1 shows that the client always sends a command and waits for the server response. If the server supports PIPELINING, the client can send some commands without waiting for the server's response, which can then arrive later. |
| ENHANCEDSTATUSCODES | In its responses, the server returns not only the message number and the message text, but also the message text is prepended by the numerical identification of the message. This numeric identifier consists of three numbers separated by dots and precisely identifies the server message regardless of language environment. |
| AUTH | Specifies the supported authentication methods. |
| STARTTLS | TLS Supported. |
| SMTPUTF8 | Internationalized email supported. |

## 20.2 SMTP and DNS

The SMTP server either accepts mail locally or forwards it to other SMTP server, ideally directly to the recipient's SMTP server. Only if it fails, it tries to forward the mail to another MTA (on the MTA Relay) (see Figure 20.2). SMTP server does not try to deliver mail to arbitrary MTA, but exclusively to the MTA listed in DNS as the MTA relay for the addressee's domain. In DNS, MX records are used to specify which MTA relays are relevant for a given domain. The server with the lowest MX priority is the recipient's server.

## 20.3 Authentication

In the case of the SMTP protocol, the main goal of authentication is to limit attackers from sending spam to the SMTP server.

In its own configuration, the MTA has conditions under which it will accept emails from SMTP clients. The basic conditions are:

- SMTP servers, that are marked in DNS as MTA, relay for a specific domain, accepting mails for this domain.
- Authentication based on the sender's IP address is often used within the company intranet or university campus.
- Authentication built into the SMTP protocol.

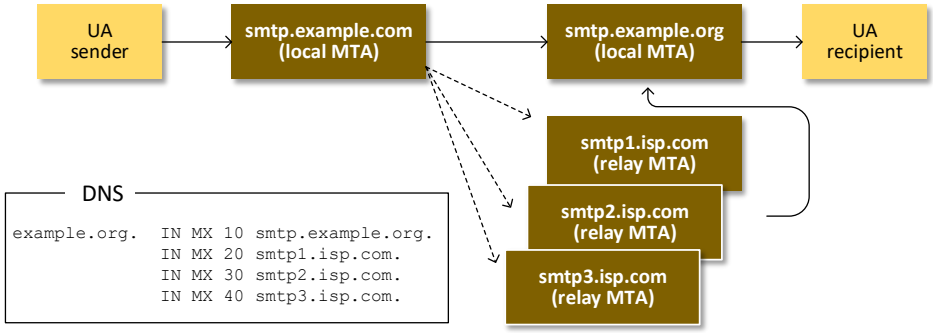



Figure 20.2 *Mail transport is controlled by DNS*

The SMTP server evaluates whether it will accept mail based on the processing of not only the listed basic conditions, but often also other conditions formulated in its configuration file.

The attackers try to send mail to the MTA relay in DNS marked as a relay for a specific domain. This can hardly be prevented other than adding the spammer's system to a blacklist of systems from which mail is not accepted.

What authentication methods the SMTP server supports are returned in response to the `EHLO` command as `AUTH` extension parameters.

Examples of authentication methods:

- `LOGIN` – authentication with username and password as if in a terminal dialog.
- `PLAIN` – authentication by username and password. But everything is entered in one line, parameters are separated by an ASCII character with the content `00` (binary).

The client usually performs authentication immediately after the EHLO command. The client starts the authentication dialog with the AUTH command with the parameter that is the selected authentication method. The server returns a response with the code 334. The client responses by authentication information. Both the server challenge and the client response are Base64 encoded.

**Example** (`C` = client, `S` = server):

```
C: Open TCP connection
S: 220 server.example.org  ESMTP server ready
C: EHLO client.company.com
S: 250 server.example.org
S: 250 AUTH LOGIN PLAIN
```

```
C: AUTH LOGIN
S: 334 VXNlcm5hbWU6
C: ZG9zdGFsZWs=
S: 334 cGFzc3cwcmQ=
C: aGVzbG8=
S: 235 Authentication successful   ,
```

where:

```
Base64(Username:) = VXNlcm5hbWU6
Base64(dostalek)  = ZG9zdGFsZWs=
Base64(Password:) = cGFzc3cwcmQ=
Base64(passw0rd)  = aGVzbG8=
```

After successful authentication, the session between the client and the server switches to the authenticated state. The client can then start sending individual emails.

## 20.4 SASL

RFC 4954 [107] implements Simple Authentication and Security Layer (SASL) as a framework for authentication in the SMTP protocol. SASL is a universal layer (implemented as a software library) for authentication in client/server protocols, which can also be used by other protocols, e.g. IMAP4 (chapter 22 IMAP4), POP3 (chapter 21 POP3), LDAP (Lightweight Directory Access Protocol) [108].

The basic operations of SASL are straightforward:

1. The server provides a list of supported authentication mechanisms.
2. The client says which one will be used (SMTP server provided authentication mechanisms are returned in response to the EHLO command in AUTH extension parameters).
3. Authentication will take place using the mechanism chosen by the client.

**Note:** SASL supports the PLAIN mechanism, but does not support the LOGIN mechanism.

## 20.5 SMTP over TLS

Using the STARTTLS extension, it is possible for the client and the server to agree to secure their communication using SSL/TLS.

The server responds to the client's `EHLO` command that it supports the `STARTTLS` extension. The client then issues the `STARTTLS` command and, if the server is ready, returns a "220 Go ahead". Now follows the classic SSL/TLS dialogue (`ClientHello`, `SeverHello`,...).

After successful completion of the initial SSL/TLS dialogue, the original information is forgotten and it starts again, i.e., the client repeats the EHLO command again. This happens because communication now runs through a secure channel. The server can thus provide a wider range of its services, i.e., a larger repertoire of SMTP protocol extensions.

**Example** (`C` = client, `S` = server):

```
C: Open TCP connection
S: 220 server.example.org SMTP service ready
C: EHLO client.example.com
S: 250 server.example.org
S: 250 STARTTLS
S: 250 HELP
C: STARTTLS
S: 220 Go ahead
C: start SSL/TLS dialog
```

*... SSL/TLS handshake ...*

```
C: EHLO client.example.com
S: 250 server.example.org
S: 250 8BITMIME
S: 250 AUTH LOGIN PLAIN
S: 250 DSN
```

*secured*

Note that our server did not offer password authentication before SSL/TLS dialogue. It only offers this in a secure channel. Therefore, the secured channel prevents password eavesdropping.

# 21 POP3

The SMTP protocol delivered the mail to the user's mailbox. **POP3 (Post Office Protocol version 3)** is a simple protocol by which the user can **download messages from his mailbox on the mail server to local mailboxes** on the PC. It is intended for offline work with the mail server, i.e., it downloads a batch of mails from the server to the client, and, then the user processes it. POP3 is specified in RFC 1939 [109].

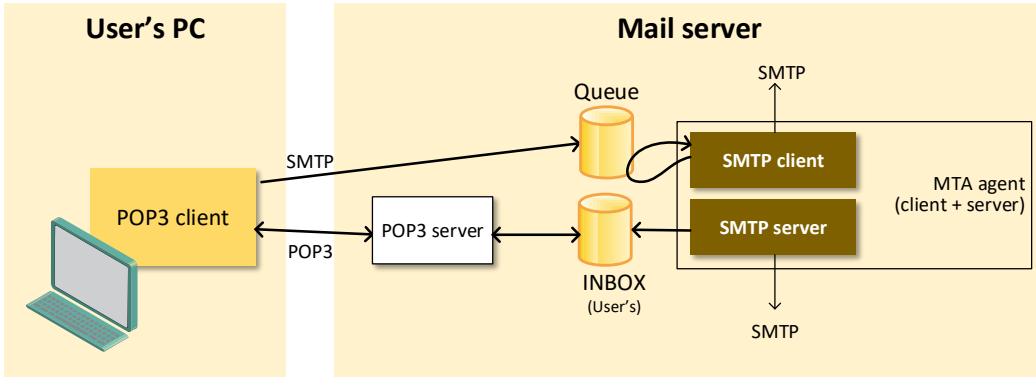



Figure 21.1 *POP3*

The POP3 client usually connects to well-known port 110/tcp of the server (if it is running with TLS security, then it usually uses port 995/tcp). After the connection is established, the server introduces itself and is in the so-called authentication state, i.e., waiting for user authentication. After the positive authentication of the user, the communication is switched to the transactional state, where the user can download/delete the mails in his mailbox in the POP3 server. At the end of the session, the client goes to the UPDATE state, when all changes are made to the mailbox on the server, i.e., even if the client in the transactional state, for example, cancels some emails in his mailbox on the server, such cancellation is still revocable during the transactional state.

For each client command, the server returns:

- A **positive answer** starting with `+OK`.
- A **negative response** beginning with `-ERR`.

The POP3 handshake is clarified with the following dialogue of the POP3 client (C) with the POP3 server (S):

C: *Open TCP connection*

```
S: +OK POP3 server ready
```

We are now in the authentication state. With the CAPA command, we can find out the list of supported capabilities:

```
C: CAPA
S: +OK Capability list follows
S: TOP
S: USER
S: EXPIRE 0
S: UIDL
S: RESP-CODES
```

Now we will authenticate. The classic authentication method in the POP3 protocol is authentication with a username and password:

```
C: USER dostalek
S: +OK Password required for user.
C: PASS password
S: +OK user has 2 message(s) (3605 octets).
```

Now we have reached the transaction state and we know that we have 2 emails with a total size of 3605 bytes in the mailbox on the server. We get the same information as with the STAT command:

```
C: STAT
S: +OK 2 3605
```

With the LIST command, we can get a list of mails in the mailbox and their sizes:

```
C: LIST
S: +OK 2 messages (3605 octets)
S: 1 1196
S: 2 2409
```

If we want to read the header and the first 10 lines from mail 2, then we use the command:

```
C: TOP 2 10
S: +OK Message follows
```

S: …. *The beginning of email 2*

If we want to download the entire mail 2, then we can use the RETR command:

```
C: RETR 2
```

S: +OK 1196 octets - *remained in the mailbox*

S: … *entire mail 2*

If we change out mind, we can restore all deleted emails again (until we go to the update state):

```
C: RSET
S: +OK Maildrop has 2 messages (3605 octets)
We go to the UPDATE state with the QUIT command:
C: QUIT
S: +OK Pop server at pop3.example.org signing off.
```

Connection was closed.

Table 21.1 *POP3 Capabilities*

| CAPA Tag | Added Commands | Meaning |
|---|---|---|
| **TOP** | TOP | Indicates that the TOP command is supported. |
| **USER** | USER, PASS | Indicates that the USER and PASS commands are supported. |
| **SASL** | AUTH | The server returns a list of supported authentication mechanisms, i.e., it is SASL layer support, similar to the SMTP protocol. The client authenticates using the selected authentication method with the AUTH command. |
| **RESP-CODES** | | Indicates that any response text issued by this server which begins with an open square bracket ("[") is an extended response code. |
| **PIPELINING** | | Indicates that the server is capable of accepting multiple commands at a time; the client does not have to wait for the response to a command before issuing a subsequent command. |
| **EXPIRE** | | Indicates how many days the client can keep the email on the server after downloading it from the server. |
| **UIDL** | UIDL | Indicates that the UIDL command is supported. |
| **STLS** | STLS | Indicates that it supports SSL/TLS. |
| **UTF8** | UTF8 | International email support: The UTF8 command switches the session from the ASCII-only mode of POP3 to UTF-8 mode. |
| **LANG** | LANG | International email support: The LANG command without a parameter lists the supported languages. LANG with the selected supported language parameter switches the POP3 server responses to that language, i.e., texts after +OK (or -ERR) will be in this language. |

## 21.1 POP3 Capabilities

POP3 was published in 1998 [109]. It was soon expanded with new capabilities. RFC 2449 [110] introduced a mechanism by which a POP3 server can publish which new capabilities it supports. The mechanism is that the POP3 server implements the capa command, after which it returns a list of supported capabilities.

Some capabilities add new commands. For example, the USER capability introduces the USER and PASS commands. Others just provide information. For example, EXPIRE capability lists how long emails are left in the mailbox after they are downloaded:

- 0 means that emails are deleted after download.
- NEVER means that emails are never deleted.
- a non-zero number indicates how many days they will remain in the mailbox. If the keyword USER is specified, then the information refers to the logged-in user.

Table 21.2 *POP3 commands*

| State | Command | Meaning |
|---|---|---|
| **Authentication** | `USER` | Username. |
| | `PASS` | Password. |
| | `UTF-8` | Switches to UTF-8 mode. |
| | `LANG` | Choice of server response language. |
| | `AUTH` | Client authentication with help of SASL. |
| | `STLS` | Start SSL/TLS. |
| | `CAPA` | The server returns a list of supported capabilities. |
| | `QUIT` | Connection termination. |
| **Transaction** | `STAT` | Server lists the number of emails in the mailbox and the total size of the mailbox. |
| | `LIST` | Returns a list of individual messages in the mailbox (one line per message). For each message lists sequence number and its size. |
| | `RETR` | Transfers the email from the server to the client. The number of the transferred message is entered as a parameter. |
| | `DELE` | Deletes the messages in the mailbox. The number of deleted messages is entered as a parameter. |
| | `NOOP` | Empty command. |
| | `RSET` | Restores messages deleted during the current session. |
| | `TOP` | Prints the beginning of the message. |
| | `UIDL` | Displays a unique identification of the message or of all messages in the mailbox. |
| | `CAPA` | The server returns a list of supported capabilities. |
| **Update** | `QUIT` | Updates the mailbox and terminates the connection. |

## 21.2 POP3 commands

In practice, POP3 servers are often designed as very simple servers that allow a user to log in to their mailbox using the POP3 protocol only once at the same time. The principle is that after the user authentication, the user mailbox is copied on the server. The original mailbox will remain and other mails can arrive in it, for example, via the SMTP protocol. The POP3 protocol works with the mailbox copy. The UPDATE state is the moment when both mailboxes are merged back into one.

When a user logs in to the POP3 server, it is first tested to see if there is already a copy of the

mailbox, if so, then it is assumed that the user works with the POP3 protocol with his mailbox and such a login request is rejected. Therefore, if the user cannot log in to the POP3 server, we first check whether a copy of the mailbox from the user's last login was accidentally preserved. If we do not have such an option, then we have to wait a few minutes for the POP3 server expires the paused client session.

## 21.3 Authentication

The original authentication of the client to the POP3 server is authentication using a username and password. By implementing the SASL layer, the server can tell the client that it supports the list of authentication methods. The situation is the same as in the case of the SMTP protocol, i.e., in addition to original authentication by username and password, for example, OAuth, Kerberos, etc. protocols may be supported.

## 21.4 POP3 over TLS

Using the STLS command, communication between a client and an SSL/TLS server can be transmitted over a secure channel. The idea is as follows:

```
C: STLS
S: +OK Begin TLS negotiation
```

An SSL/TLS dialogue begins, at the end of which a secure SSL/TLS channel will have been created.

But the practice is usually different. The POP3 server directly supports SSL/TLS, i.e., immediately after the TCP connection has been established, an SSL/TLS dialogue automatically takes place, and in the created secure channel, communication takes place in exactly the same way as in the case of an unsecured server, for example, with username and password authentication.

# 22 IMAP4

**Internet Message Access Protocol version 4 (IMAP4)** [111] is more sophisticated protocol designed for working with mailboxes from a PC on a server in On-line (and Off-line) mode. Client can work with its mailboxes from multiple applications at a time. Some applications even establish two TCP connections with the IMAP4 server at the same time (e.g., MS Outlook). One for working with mailboxes and the other for working with individual items (mail messages).

While working with the mailbox (during an established TCP connection), another application can change the content of the mailbox (e.g., the SMTP server writes newly arrived mail to the mailbox). These events (such as adding new messages to the mailbox) are signaled by the server as a part of established connection.

**Note:** On today's servers, an IMAP4 server usually runs on port 993/tcp, that requires the client to create an SSL/TLS channel immediately after establishing a TCP connection (**IMAP4 over TLS**). The second option described below requires that the TLS handshake be initiated exclusively by the IMAP4 server (see section 22.1 Not Authenticated State).

If one application has a mailbox open for read-write and another application wants to open the same mailbox also for read-write, the first application must be changed to read-only. The first application must be immediately signaled that it still has the mailbox open as read-only, and must reopen the mailbox if it wants to make a change to the mailbox.

But first, we need to describe what IMAP4 commands look like. Commands are again entered in ASCII (as in the case of SMTP, POP3, etc. protocols). However, the commands are different from POP3 commands. The difference is not only in the names of the commands, but especially in the philosophy of the commands. In the IMAP4 protocol, a number of commands can be entered, and responses to individual commands can come from the server in any order. Therefore, the client numbers the commands and the server repeats in its response the number of the command to which it responds. It is up to the client to identify (number) the commands. The command identifier is generally a string (not necessarily a number). Unambiguity of identification is also a responsibility of the client.

After the client establishes a TCP connection with the server, the server introduces itself, for example:

C: *Open TCP connection*

```
S: * OK [CAPABILITY STARTTLS AUTH=SCRAM-SHA-256 LOGINDISABLED IMAP4rev2]
```

```
IMAP4rev2 Service Ready
```

The server returns the supported capabilities in [CAPABILITY …], the protocol version and the information that it is ready to receive commands.

We have now reached the so-called not authenticated state, where authorization is required.

## 22.1 Not Authenticated State

It is necessary to authenticate, but IMAP4rev2 server at first **requires communication to be secured by TLS**:

**`C: a000 starttls`**

`S: a000 OK Proceed with TLS negotiation`

*… TLS securing (TLS Handshake Protocol)*

The client tagged its request with command identifier. In our case, it is the string `a000` tag followed by its own command (in our case, the `starttls` command). A command can be followed by its parameters, but the `starttls` command has no parameters. Each request has its own unique tag.

The server responds with two types of responses:

1. Untagged responses that have an asterisk in place of the command/response tag. These untagged responses carry the information that the client requested.
2. Tagged answers (the tag corresponds to the tag of the client's request – in our case `a000`), which begin request tag and tell how the request turned out.

Now the authentication selected from CAPABILITY (e.g., SCRAM-SHA-256) takes place.

**`C: A001 AUTHENTICATE SCRAM-SHA-256`** …

…

`S:  A001 OK` …

This brought us to the so-called authenticated state.

## 22.2 Authenticated state

In the authenticated state, the client can work with mailboxes on the server like with files, i.e., it can create (`CREATE`), delete (`DELETE`), rename (`RENAME`), display a list of mailboxes (`LIST`), etc. If the mailboxes are implemented as files in the file system, then it is actually working with files.

But it is not enough to just create a mailbox, you also need to tell the server to register it as a mailbox with the `SUBSCRIBE` command. It can be unregistered with the `UNSUBSCRIBE` command. The status of the mailbox can be displayed with the `STATUS` command.

With the `SELECT` (or `EXAMINE`) command, the client selects (opens) a specific mailbox and switches to the "Selected" state, in which it can work with individual items of the select mailbox (e.g., transfer items from the server to the client, etc.):

```
C: a002 select inbox
S:   * 18 EXISTS
S:   * FLAGS (\Answered \Flagged \Deleted \Seen \Draft)
S:   * OK [UIDVALIDITY 3857529045] UIDs valid
S:   * LIST () "/" INBOX ("OLDNAME" ("inbox"))
S:   a002 OK [READ-WRITE] SELECT completed
```

The server's response is more complex – it consists of several line types:

- The line `18 EXISTS` communicates that the currently opened mailbox contains 18 messages.
- The line `OK [UIDVALIDITY...` communicates the unique identification of the mailbox.
- The `FLAGS...` line identifies the flags that are applicable for this mailbox:

  `\Seen` - message has been read.

  `\Answered` - message has been answered.

  `\Flagged` - message is "flagged" for urgent/special attention.

  `\Deleted` - message is "deleted" for removal by later `EXPUNGE`.

  `\Draft` - message has not completed composition (marked as a draft).
- The line `LIST...` indicates that the selected mailbox is called `INBOX`, but it is also possible to use `inbox` when searching for a mailbox.

## 22.3 Selected state

Each mailbox has its unique identification, which the server returns in the `UIDVALIDITY` parameter. This identification is, for example, independent of the name of the mailbox as a file.

Similarly, each message in the mailbox also has its own identification, which is also unambiguously guaranteed. Combining mailbox identification with message identification creates a unique identification of the message in the system.

Although this unique identification is guaranteed, it is not very practical. Messages within the mailbox are therefore numbered sequentially starting from one. Therefore, if, for example, a message in the mailbox is canceled with the `EXPUNGE` command, the messages in the mailbox must also be renumbered.

### 22.3.1 COPY

The COPY command copies messages from a selected mailbox to the mailbox specified as the second parameter. The first parameter is the number of the message to be copied. Instead of a single message, we can specify a message interval. For example, `3:6` is the interval of messages number 3, 4, 5 and 6. The following example copies messages from 3 to 6 to the mailbox mail/mailbox1:

```
C: 3 COPY 3:6 mail/mailbox1
S: 3 OK COPY completed
```

### 22.3.2 SEARCH

The `SEARCH` command can be used to search for messages in the mailbox. It is searched according to the search criteria that are specified as a parameter of the `SEARCH` command. If multiple search criteria are specified, messages that meet all the criteria at the same time (AND) are searched for. There is also an OR criterion that evaluates sub-criteria of OR operations. Similarly, there is also a criterion of NOT negation.

**Example:**

```
C: 789 SEARCH UNSEEN NOT FROM dostalek SINCE 4-Jan-2023
S: * SEARCH 7 8
S: 789 OK SEARCH completed
```

Messages 7 and 8 meet the specified criteria. In the following example, the criteria are used:

- `UNSEEN` – Messages that do not have the `\Seen` flag set.
- `NOT` – Messages that do not match the specified search key.
- `FROM` string – Messages that contain the specified string in the envelope structure's `FROM` field.
- `SINCE` date – Messages whose internal date (disregarding time and timezone) is within or later than the specified date.

### 22.3.3 FETCH

The FETCH command can be used to download a message or part of it from the server. The FETCH command has the following syntax:

```
FETCH messages "what"  ,
```

where `message` is either a message number or a message interval (e.g., `2:93`), `what` tells what part of the message to download.

**Example:**

- `BODY[section]` – the server returns the contents of the section of the message specified in square brackets. The sections are `HEADER` (header), `HEADER.FIELDS` (headers whose names are given in round brackets), `HEADER.FIELDS.NOT`, `MIME` and `TEXT` (message text).

  For example:

  ```
  C: 89 FETCH 9 BODY[HEADER.FIELDS (FROM DATE)]
  S: * 9 FETCH ENVELOPE (BODY[HEADER.FIELDS („FROM“ „DATE“)] {83}
  S: Date: Fri, 5 Jan 2001 15:28:39 +0100
  S: From: Libor Dostalek test user <dostalek>
  S:
  ```

```
S: )
S: 89 OK FETCH completed
```

The `FETCH` command returned the `Date` and `From` header fields from the message number 9. Data sent from the server are enclosed in round brackets (after the string `ENVELOPE`).

The first line of returned data (`BODY[HEADER.FIELDS ("FROM" "DATE")]`) repeats which data the server returns. Then string `83` is returned. This string represents the length of the following data, which did not fit on this line, meaning, that 83 bytes of data are sent and everything is terminated at the end of the data being sent with a right round bracket.

- `FLAGS` – returns the flags of the given messages:

```
C: 81 FETCH 9:10 FLAGS
S: * 9 FETCH (FLAGS (\Seen))
S: * 10 FETCH (FLAGS (\Seen))
S: 81 OK FETCH completed
```

- `RFC822.SIZE` – returns the length of the given messages:

```
C: 84 fetch 9 RFC822.SIZE
S: * 9 FETCH (RFC822.SIZE 277)
S: 84 OK FETCH completed
```

- UID – returns serial number of the given messages.

However, perhaps the most interesting part is how to fetch the full message. For example:

```
C: a003 fetch 12 full
```

### 22.3.4 STORE

The `STORE` command is used to change the flags of messages in the mailbox. It is, of course, possible to set, for example, the attribute `\Concept` for the item. However, probably the most interesting is the `\Delete` attribute, which prepares an item to be deleted. Then the following `EXPUNGE` command deletes the entry.

The STORE command has three parameters:

- The item number (or interval) in the mailbox for which the attribute should be changed.
- The second parameter is one of the keywords: `FLAGS`, `+FLAGS`, or `-FLAGS`. We use the +FLAGS keyword to add the attributes listed as the third parameter. Using the word -FLAGS we remove the parameters and using the word `FLAGS` we set the parameters of the item to the parameters listed as the third parameter.
- The last parameter is a list of attributes enclosed in round brackets.

**Example:**

```
C: 35 store 9:10 +FLAGS (\Deleted)
S: * 9 FETCH (FLAGS (\Seen \Deleted))
S: * 10 FETCH (FLAGS (\Seen \Deleted))
S: 35 OK STORE completed
```

### 22.3.5 EXPUNGE

The `EXPUNGE` command deletes items marked with the `\Delete` flag in the mailbox. In previous example are messages 9 and 10 marked with the `\Delete` flag. Then the `EXPUNGE` command causes:

```
C: 38 EXPUNGE
S: * 9 EXPUNGE
S: * 9 EXPUNGE
S: * 9 EXISTS
S: * 0 RECENT
S: 38 OK Expunged 2 messages
```

You probably think I made a mistake because messages 9 and 10 should have been deleted and not line 9 twice. But it works as follows: immediately after deleting message 9, the mailbox will be renumbered, because messages in the mailbox must be numbered without breaking the sequence of numbers. So the original message number 10 suddenly had the number 9. Subsequently, message number 9 (originally 10) will be deleted again.

### 22.3.6 CLOSE

The `CLOSE` command permanently removes all messages that have the `\Deleted` flag set from the currently selected mailbox, and it returns to the authenticated state from the selected state. No untagged `EXPUNGE` responses are sent.

```
C: 100 close
S: 100 OK CLOSE completed
```

### 22.3.7 LOGOUT

The `LOGOUT` command informs the server that the client is done with the connection. The server sends a `BYE` untagged response before the (tagged) `OK` response, and then closes the network connection.

```
C: a006 logout
S: * BYE IMAP4rev2 server terminating connection
S: a006 OK LOGOUT completed
```

# 23 MIME

The SMTP message format allows data to be transmitted only in ASCII (seven bits per byte). However, it soon became clear that this fact did not meet the requirements of many users who wanted to use email to send text of other character sets, images, sounds, binary files, etc. The **MIME (Multipurpose Internet Mail Extension)** [24] solves both the transfer of non-ASCII messages and the transfer of messages that contain multiple parts (e.g., attachments).

The problem of how to send an email message containing non-ASCII characters can be solved without MIME. This means encoding the message into "**seven-bits**" ASCII before sending, for example, using **Base64** or **Quoted-Printable encoding**. The recipient must first decode the message and then he can read it. The sender and recipient must agree on the type of encoding of the transmitted data to ASCII out of this communication.

This solution is unacceptable for regular users. They do not want to deal with coding, they want the mail client software to solve these problems for them. MIME came up with a solution by expanding the repertoire of mail header fields with header fields describing the type of data being transferred and the algorithm used to encode the data into ASCII before sending. This allows the entire process to be automated without the need for user intervention.

The recipient's software can automatically detect from the header fields how the data were encoded and automatically decode the message. In addition, the sender's software stores the type of transmitted data in the `Content-Type` header field, according to which the recipient's software automatically recognizes which program the recipient should use to view the data. Recipient's email client then launches the appropriate program (text editor, image graphic editor, sound player, etc.) according to the type of data being transferred.

This additional information about the transmitted data is carried by header fields starting with the string "Content-", which is specified by the MIME standard. MIME is a standard that completes email message header fields and ensures backward compatibility. MIME is designed so that messages containing diacritics, images, sounds, etc. can be sent through the existing mail system.

## 23.1 MIME header fields

MIME defines the following header fields:

- `MIME-Version` indicates that the message is in compliance with MIME standard, i.e., according to RFC2045 [24] and RFC2049 [112].

- `Content-Type` specifies the type and subtype of data sent in the message body (text, audio, video, virtual reality…).
- `Content-Transfer-Encoding` specifies the encoding by which the message is converted into a format compatible with the transfer mechanism (encoding to ASCII).
- `Content-ID` contains unique message identity (e.g., for a reference in a hypertext link).
- `Content-Description` contains text description of the content.
- `Content-Disposition` determines if the message data is intended for automatic display by the recipient (inline) or if it is not intended for ”manual” processing by the recipient (attachment).

### 23.1.1 Mime-Version

This mandatory header field is the only MIME header that does not begin with the word `Content`. It specifies the version of the MIME standard. The reason for introducing this header is to ensure compatibility—its presence allows client software to recognize that the message is MIME-encoded. The MIME extension defined in RFC2045 [24] and RFC2049 [112] is version 1.0.

The HTTP protocol also uses header fields based on the MIME concept, and many HTTP headers begin with the word `Content`, though they are adapted for HTTP-specific purposes. The fact that HTTP headers are not fully compatible with MIME is indicated by the absence of the `Mime-Version` header field in HTTP message headers.

Any message conforming to RFC2045 [24] and RFC2049 [112] is required to include this header field. Its format is as follows:

```
Mime-Version: 1.0
```

This header field must precede all other MIME headers.

### 23.1.2 Content-Transfer-Encoding

The data we want to send via email are often 8-bit or binary, which usually cannot be transmitted directly. Therefore, a conversion mechanism—called encoding—must be defined to convert the data into a form that uses only ASCII characters. In other words, the data are transformed into a 7-bit format.

The type of encoding used is specified in the `Content-Transfer-Encoding` header field. This header contains information about the algorithm used to encode the data into a 7-bit format. If the data are not encoded, the header indicates whether the content is in `7-bit`, `8-bit`, or `binary` form.

MIME defines two encoding algorithms for this purpose: Quoted-Printable and Base64 (see section 3.1 Base64).

The `Content-Transfer-Encoding` header field thus most often carries the following encoding methods:

- `Quoted-printable` encoding.
- `Base64` encoding.
- `7bit` – data are not encoded, are in lines, contain only US-ASCII characters.
- `8bit` – data are not encoded, are in lines, but non-US-ASCII characters may also occur.
- `Binary` – the data are not encoded, the bit stream is not divided into lines. The total number of bits transmitted must be divisible by eight.

The `Content-Transfer-Encoding` header applies to the entire message body. If the header appears in a subpart of the message, it applies to that subpart only.

**Example:**

```
Mime-Version: 1.0
Content-Type: text/plain; charset=ISO-8859-2
Content-Transfer-Encoding: base64
```

We interpret it as follows: the body of the message is a string of ASCII characters created by base64 encoding. The original data were in the ISO-8859-2 character set. They must therefore be converted to this set again when displayed to recipients.

### 23.1.3 Content-Type

The header field describes the type of data contained in the body of the message so that the client who receives this message can choose an appropriate way of presenting the content of the message. The header has the following form:

```
Content-Type: type/subtype; parameters
```

This header field indicates the need for a sufficiently complete description of the data contained in the body to enable the receiving user agent to select an appropriate agent or mechanism for presenting the data to the user, or otherwise deal with the data in an appropriate manner. The value in this field is called a **media type**. The media type is specified by type and subtype and optionally by using additional parameters. Parameters have the following form:

```
attribute=value; …
```

There can be more parameters – they are then separated by a semicolon and their order does not matter.

The **type** specifies what type of data it is, whether the body of the message contains text, an image or, for example, binary data (octet stream). The **subtype** then specifies the specific format of the image, text, etc.

**Examples:**

```
Content-Type: image/jpeg
```

Informs the recipient that the content of the message is a JPEG image.

Similarly, the following header field informs a recipient that the content is a document produced by the MS Word application:

`Content-Type: application/msword`

The types are of two kinds:

- Simple types, describing the type of transmitted data. These are, for example, types: `text, application, image, sound, video, model`, etc.
- Composite types, specifying that the message consists of several parts. These are, for example, types: `message, multipart, report`, etc.

So far, we have only dealt with simple messages, that is, messages consisting of one part. We will now consider the messages made up of multiple sub-messages (composite messages). Each individual sub-message can be also composite message or it can be already an simple message.

A composite message may contain in its body:

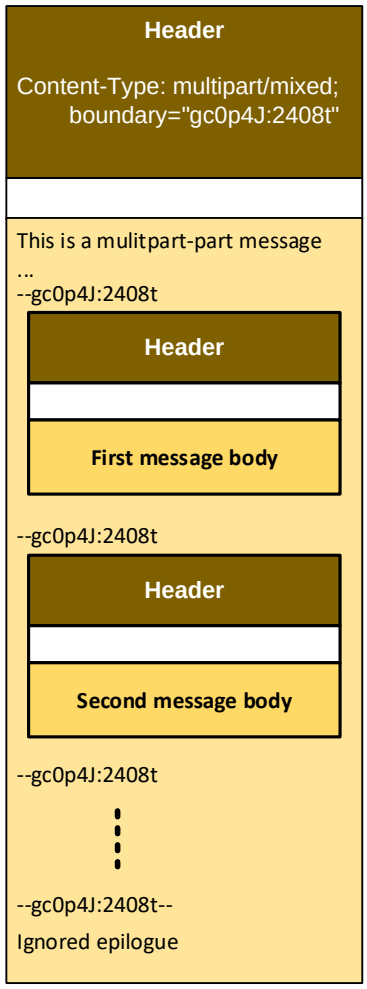
- Sub-messages, then the header field `Content-Type: multipart` is used.
- A long message can be transported as several shorter ones, then the `Content-Type: message` header field is used.

#### 23.1.3.1 Multipart

The body of a message of this type contains several different parts - several sub-messages. Each sub-message begins with an opening delimiter with prefix – –, followed by the headers of this part, an empty line and the body of the sub-message itself, followed by a delimiter again. The first sub-message can be followed by another sub-message again separated by delimiters, etc. The last sub-message is terminated by a trailing delimiter which suffix – –

Individual sub-messages are not interpreted during mail transfer. They may or may not contain header fields, but an empty line after the header must always be included. If header fields are not specified for the sub-message, the default header fields from the header of the message are applied. In practice, we most often come across emails that contain attachments. And it is the original message and the individual attachments that make up the individual sub-messages.

A sub-message delimiter is a special sequence of characters that must not occur anywhere inside message. The delimiter is defined in the message header field in the `Content-Type` header field in the `boundary` parameter. This parameter has the form `boundary=string`. The maximum delimiter length is 70 characters. For example in figure 23.1 the delimiter is gc0p4J:2408t.




```
From: Libor Dostalek
<dostalek@example.org>
To: bob@example.com
Subject: Mail with attachement
MIME-Version: 1.0
Content-type: multipart/mixed;
              boundary=„gc0p4J:2408t“
```

*This is the preamble that is ignored.*

```
--gc0p4J:2408t
Content-type: text/plain
```

*Email text*

```
--gc0p4J:2408t
Content-type: applicaton/msword
```

*Attachment in MS Word*

```
--gc0p4J:2408t
```

*Possible next attachments*

```
--gc0p4J:2408t--
```

*This is the epilogue which is also ignored*

Figure 23.1 *Multipart*

The `Multipart` type has, among others, the following subtypes:

- `Multipart/Mixed` – is the primary subtype. It is intended for messages that contain independent sub-messages that need to be bind in a specific order. A classic case of the multipart/mixed subtype is an email message containing one or more attachments.
- `Multipart/Alternative` – A message of this type contains several parts, all parts containing the same information, differing only in quality, for example, the same information is once written in ASCII, then spoken in audio format. The recipient's software must recognize which format it can display and select the highest quality format. Software creating a message must order the sub-messages in increasing quality. The quality is specified by the parameter q.
- `Multipart/Signed`, which is intended for a digitally signed message, which specifies a message consisting of two parts:

1. From the message.
2. From the digital signature of this message.

The `Multipart/Signed` subtype uses three parameters:

- `Micalg=alg`, where the value of `alg` indicates the algorithm for calculating the hash entering the digital signature. The hash is calculated from the entire first sub-message including MIME headers. (Micalg = *Message integrity check algorithm*).
- `Protocol=type/subtype` specifies the protocol by which the digital signature in the second sub-message is created. For example, the S/MIME protocol uses for the Protocol parameter value `Application/pkcs7-signature`. The electronic signature is calculated over the entire first sub-message, including all MIME headers. For example, if the first sub-message is Base64-encoded, the signature covers both the headers (including the `Content-Transfer-Encoding` field) and the Base64-encoded body. If an intermediate mail server were to decode the Base64 content en route to the recipient, it would break the electronic signature.
- `Boundary=value` contains value of delimiter that separates partial messages.

- `Multipart/Encrypted` specifies a message in an electronic envelope (encrypted message). It again consists of two parts:
  - Information about the encryption method used (e.g., encryption software version).
  - Encrypted message.

There are no subtypes defined for messages that are both digitally signed and encrypted. To handle such cases, the message must first be digitally signed, and then the entire result must be encrypted.

#### 23.1.3.2 Message

This subtype e.g. allows:

- Sending a mail message as the body of another mail message (`message/rfc822`).
- Sending a long message as several shorter ones (`message/partial`).
- Sending only information that the message is located on a server (`message/external-body`) instead of the message body.

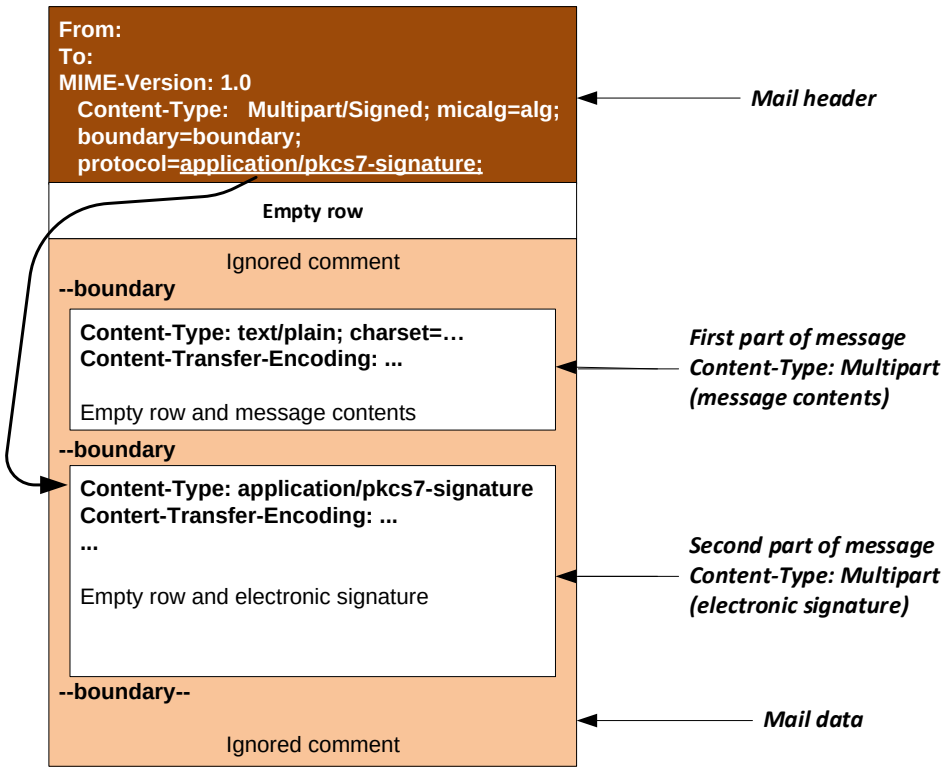



Figure 23.2 *Multipart - signed*

## 23.2 S/MIME

S/MIME is a protocol for end-to-end security of email messages. The message is secured by the sender before transmission. It then travels through the network to the recipient without the possibility of the security being compromised during transit. The message is therefore protected against attacks along the entire path from sender to recipient.

Several protocols for end-to-end email security have been proposed. Historically, these included the PEM and MOSS protocols, which were never widely adopted. While PGP initially gained broader acceptance, today S/MIME appears to be the dominant choice. The current version of S/MIME (version 4.0) is defined in RFC 8550 [113] and RFC 8851 [114].

Email messages, as well as JSON or XML , are typically free-form and may include arbitrary whitespace. However, if such texts are to be digitally signed, they must first be transformed into a unique, standardized representation. Even seemingly insignificant changes, such as adding whitespace, can invalidate a digital signature. This transformation process is known as **canonicalization**.

From the entering of the message in the sender's local environment to its sending via the SMTP protocol, the message goes through several steps:

1. The message is entered into the sender's local environment.

2. In the case of plain text, canonicalization involves replacing line ends with CR and LF character pairs. For other types of data, canonicalization can be significantly more complex.
3. Encoding to 7bit is done especially if the message is to be transmitted via email. Quoted printable or Base64 encoding is used.
4. Furthermore, the message is supplemented with MIME header fields, which, among other things, specify the data type (e.g., `Content-Type: text/plain`) and the data encoding method. The result is a MIME message consisting of header, empty lines and a message body. In addition to MIME headers, classic RFC 822 headers can also be added if these headers carry important information regarding the message (e.g., `Subject, To, From, Cc`).
5. In practice, the resulting MIME message is often not elementary, but composite. If the mail contains attachments, then the resulting message is of the `Multipart/mixed` type. In the case of some sub-messages, we may also encounter the `Multipart /alternative` type.
6. The resulting MIME message always enters the CMS machine as `id-data`. Even if there is, for example, a signed message inside, it is always wrapped in MIME headers!
7. Since the result from the CMS engine is binary data and the message is likely to be transmitted via email, it must be canonicalized and encoded to the 7bit.
8. The result of possible canonicalization and encoding is wrapped again with MIME headers that describe what data is involved (e.g., `Content-Type: application /pkcs7-mime`). Now it depends on whether the message should be further secured (e.g., signed). If it is further secured, then the recreated MIME message is sent to the CMS machine. If not, then it is sent further to be supplemented with RFC 822 headers.
9. The last step is to send the message using, for example, the SMTP, i.e., inserting the message into the SMTP envelope. Whether the SMTP protocol is somehow secured (e.g., SSL/TLS) is out of scope of this chapter.

For better understanding, lets show practical example of sent message:

*SMTP envelope:*

```
S: 220 prometheus.terms.cz ESMTP Sendmail …
C: HELO dostalek2
S: 250 prometheus.terms.cz, pleased to meet you
C: MAIL FROM: <libor@example.org>
S: 250 <libor@example.org>... Sender ok
C: RCPT TO: <marta@nextra.cz>
S: 250 <marta@nextra.cz>... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
```

*RFC-822 envelope:*

```
Message-ID: <002201c4fbcf$4f5689d0$7002a8c0@example.org>
From: =?iso-8859-2?Q?Libor_Dost=E1lek?= <libor@example.org>
To: <marta@nextra.cz>
Subject: Greeting
Date: Sun, 16 Jan 2005 14:28:43 +0100
\raggedleft \textit{extern MIME envelope:}
```

*MIME envelope:*

```
    MIME-Version: 1.0
    Content-Type: multipart/signed;
    protocol="application/pkcs7-signature";
    micalg=SHA1;
    boundary="----=_NextPart_000_001C_01C4FBD7.AEA7F600"

    This is a multi-part message in MIME format.
------=_NextPart_000_001C_01C4FBD7.AEA7F600
```

*first sub-message (text):*

```
      Content-Type: text/plain; charset="iso-8859-2"
      Content-Transfer-Encoding: quoted-printable

      Please answer me in encrypted form.
      D=EDk
      Libor
```

```
------=_NextPart_000_001C_01C4FBD7.AEA7F60
```

*second sub-message (signature):*

```
     Content-Type: application/pkcs7-signature; name="smime.p7s"
     Content-Transfer-Encoding: base64
     Content-Disposition: attachment; filename="smime.p7s"

     MIAGCSqGSIb3DQEHAqCAMIACAQExCzAJBgUrDgMCGgUAMIA  …
     AAAAAA==
```

```
------=_NextPart_000_001C_01C4FBD7.AEA7F600--
```

```
C: .
S: 250 2.0.0 j0GDSidj025250 Message accepted for delivery
C: QUIT
S: 221 2.0.0 prometheus.terms.cz closing connection
```

We intercepted an S/MIME message containing an external digital signature. First, we extract the message from the SMTP envelope. While the SMTP envelope was relevant in previous chapters, it is now considered redundant. We are only interested in the MIME envelope, which specifies the message format via the `Content-Type: multipart/signed` header field. As we know from the previous chapter, this is a composite type consisting of two parts: the first part contains the signed text, and the second part contains the external signature.

The first sub-message was composed, canonicalized, and encoded using quoted-printable. The digital signature included in the second part was calculated over the entire MIME message constructed in this manner. The second part was created by the CMS engine, then Base64-encoded and inserted into a MIME entity with `Content-Type: application/pkcs7-signature`. Both sub-messages were embedded in the final MIME message with `Content-Type: multipart/signed`.

### 23.2.1 CMS and S/MIME

S/MIME has chosen the CMS (see chapter 6 CMS) protocol to secure its messages. In doing so, it supports the following three message security methods:

- Digital signature (CMS uses `SignedData` type).
- Electronic envelope (uses CMS type `EnvelopedData`).
- Compressed message (uses CMS type `CompressedData`).

Any combination of individual security methods is possible. For example, a message can first be digitally signed, then wrapped in MIME headers, and finally placed in an electronic envelope (i.e., encrypted).

As stated before, any combination of security measures is possible. However, the resulting configuration should serve a practical purpose. For instance, if compression is used, it is most effective to apply it to the original message. This would typically be followed by a digital signature and then by encryption.

In some scenarios, it is appropriate to sign a message even after it has been placed in an electronic envelope. Verifying such messages can then be done automatically, which is beneficial in secure environments—such as secure mailing lists or conferences—where the message might be compressed, signed by the sender, encrypted with the recipient's key, and finally signed by the conference itself to distinguish it from spam.

When combining individual security mechanisms, each step wraps the MIME-formatted message. That is, every security operation is applied to a complete MIME message. This approach differs from the CMS model outside the S/MIME context, where MIME is not necessarily used between steps.

#### 23.2.1.1 SignedData

S/MIME requires the use of specific signed attributes within the SignedData CMS message. While user agents (UAs) may be lenient and allow users to view and verify messages that do not include these attributes, they should always generate messages that contain at least one instance of each signed attribute listed in the Table 23.1.

Table 23.1 *SMTP commands*

| CMS signed attribute | Standard | Meaning |
|---|---|---|
| contentType | CMS | Specifies the content type of the message being signed |
| messageDigest | CMS | Contains the hash from the signed message, more precisely the hash from **DTBS/R (Data To Be Signed / Representation)** |
| signingTime | CMS | Contains the time of signing the message (but the time is not guaranteed by an independent third party) |
| sMIME Capabilities | S/MIME | It includes signing algorithms, symmetric encryption algorithms, and other capabilities preferred by the sender's UA |
| sMIMEEncryptionKeyPreference | S/MIME | Allows the sender to specify a certificate with a preferred encryption key (for a possible response) |
| signingCertificate | ESS | (standard RFC2634 - Enhanced Security Services for S/MIME [115]) It allows also including the identification of the certificate in the signature for the verification of this signature. This prevents forgery of the sender's certificate. |

#### 23.2.1.2 EnvelopedData

To create an electronic envelope, we need to have information about the recipient's capabilities to create secure messages. These capabilities can be obtained from digitally signed emails that we have received from the recipient in the past. Therefore, it is convenient for our UA to archive information about the recipient's UA capabilities (from the signed `sMIMECapabilities` attribute) from the received digitally signed message, especially those that can serve to secure the response (from the signed `sMIMEEncryptionKeyPreference` attribute).

### 23.2.2 Certificate and S/MIME

The email address of both the sender and the recipient should be presented in the certificate used. The email address should be presented in the Subject Alternative Name extension of the certificate, where multiple email addresses can also be presented. The sender's UA should check:

- In the case of a digitally signed email message, it should check that the email address presented in the sender's certificate matches the address in the `From:` or `Sender:` header field.
- In the case of a message in an electronic envelope, the email address presented in the recipient's certificate matches one of the `To:`, `Cc:` or `Bcc:` header fields.

There is a bit of a problem in the case of S/MIME with support for certificate extension.

S/MIME is required to process only the following certificate extensions:

- Basic Constraints.
- Key Usage.
- Authority Key Identifier.
- Subject Key Identifier.
- Subject Alternative Names.
- In case of Extended Key Usage, only object identifiers for `emailProtection` or `anyExtendedKeyUsage` are checked.

Other extensions should not be marked as critical, as UA may not be able to process them. This applies, for example, to Constrains or Certification policies.

## 23.3 Application/pkcs7-mime

`Application/pkcs7-mime` is an simple type that is suitable for an electronic envelope (”encrypted mail”), but also for an internal electronic signature (”signed mail”). In the following example, the SMTP envelope content is no longer included:

*RFC-822 envelope:*

```
Message-ID: 000601c4fbdd$40c7da00$7002a8c0@example.org
From: =?iso-8859-2?Q?Libor_Dost=E1lek?= <libor@example.org>
To: <marta@nextra.cz>
Subject: Greeting
Date: Sun, 16 Jan 2005 16:08:36 +0100
```

*MIME envelope:*

```
MIME-Version: 1.0
Content-Type: application/pkcs7-mime;
smime-type=signed-data;
name="smime.p7m"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="smime.p7m"

MIAGCSqGSIb3....
XN4iSrInJ/rwFU0vAAAAAAAA
```

Meaning of parameters:

- The `smime-type` parameter provides information about the content of the message. Since this MIME type can be used to transmit various types of secured messages, the smime-type parameter specifies the exact type of security used (see Table 23.2).
- The `name` parameter is used to inform mail gateways so they can handle the message appropriately. It can be used to specify the file name. The file name is typically limited to 8 characters with a 3-character extension (see Table 23.2).

Table 23.2 *Application/pkcs7-mime*

| **Message extension** | `application/` | `smime-type` | **CMS type** | **File** |
|---|---|---|---|---|
| Signed *(internal signature)* | `pkcs7-mime` | `signed-data` | `SignedData` | `.p7m` |
| Signed *(external signature)* | `pkcs7-signature` | n/a | `SignedData` | `.p7s` |
| Certificates only | `pkcs7-mime` | `certs-only` | `SignedData` | `.p7c` |
| Enveloped | `pkcs7-mime` | `enveloped-data` | `EnvelopedData` | `.p7m` |
| Compressed | `pkcs7-mime` | `compressed-data` | `CompressedData` | `.p7z` |

# 24 DNS and DNSSEC

**DNS** (**Domain Name System**) [116] [117] is a hierarchical and distributed naming system used to obtain information about computers, services, and other resources. DNS is typically used to resolve a hostname to an IPv4 or IPv6 address, but it can also store additional information. For example, as shown in the SIP (chapter 15 SIP), it can include phone numbers.

The entire Internet is divided into **domains**, which are groups of names that logically belong together. These domains indicate whether the names are associated with a particular company, country, or other entity. Within a domain, it is possible to create **subdomains**, which represent lower levels in the DNS hierarchy.

A host's domain name reflects its position within this hierarchy. It is composed of a sequence of domain and subdomain labels. For example, the hostname `bob.example.com` refers to a host named `bob` in the `example` subdomain, which in turn is part of the top-level domain (TLD) `com`. Later, we will see that the TLD com itself is a subdomain of the root domain, represented by a dot (`.`).

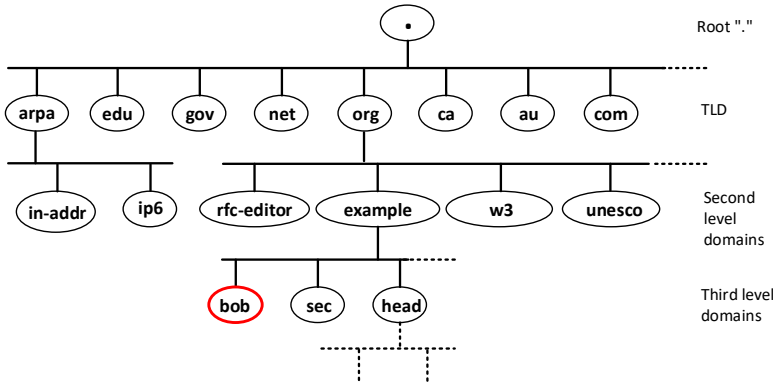



Figure 24.1 *Domain names tree hierarchy*

## 24.1 DNS name syntax

Names in DNS are listed in a dot notation (for example, `abc.head.company.com`). Names have the following general syntax:

`label.label.label .....label.` ,

where the first label is usually a computer name, followed by the name of the lowest domain, then the name of a higher domain, etc. For unambiguity, a dot expressing the root domain (.) is also listed at the end.

The entire name can have a maximum of 255 characters. The label is a string that can have a maximum of 63 characters. The string can consist of ASCII letters, numbers, and hyphens. A hyphen cannot be at the beginning or at the end of a string.

## 24.2 Internationalized domain name

For internationalized domain name see section 19.6 Internationalized email adress.

**Note:** Many domain administrators (hostmasters) restrict the use of internationalized domain names. This is because some characters in different alphabets look very similar. For example, the letter o in the Latin alphabet and the **o** in Cyrillic appear almost identical. An attacker can exploit this similarity to spoof a DNS name, such as `hоst.example.org`, by replacing the Latin o with a Cyrillic **о**, making the domain visually indistinguishable from the legitimate one.

## 24.3 Reverse domains

Some applications need to find a name for an IP address – in other words, find the reverse record. This process is the translation of an IP address into a domain name, which is often called reverse resolution.

As with domains, IP addresses also create a tree structure. Domains created by IP addresses are often called reverse domains. For the purpose of reverse translations the pseudo domains `IP6.arpa` (IPv6) and `in-addr.arpa` (IPv4) were created.

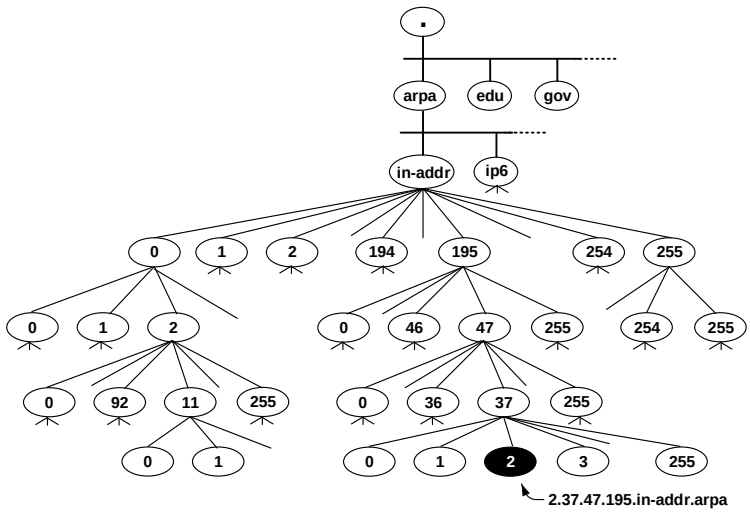



Figure 24.2 *Reverse domain to IP address 195.47.37.2*

Under the domain `in-addr.arpa`, there are subdomains with names corresponding to the first octet of the network IP address. For instance, the `in-addr.arpa` domain encompasses subdomains from 0 to 255. Each of these subdomains further contains subdomains ranging from 0 to 255. For example, for the network 195.47.37.0/24, its corresponding subdomain is `37.47.195.in-addr.arpa`. This subdomain, in turn, is part of the domain `47.195.in-addr.arpa`, and so on. It is worth noting that these domains are structured in reverse order compared to the network IP addresses.

For IPv6 addresses, the principle is similar, except that dot notation is used instead of colon notation. Example: IPv6 adders `2001:db8::567:89ab` in reverse notation:

```
b.a.9.8.7.6.5.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa
```

## 24.4 Zones

As previously mentioned, a domain is a collection of names that share a common suffix in their domain names. For instance, a domain may include a group of computers whose names end with `example.com`. However, the domain `example.com` is expandable—it can be further subdivided into subdomains such as `bob.example.com`, `sec.example.com`, `head.example.com`, and so on.

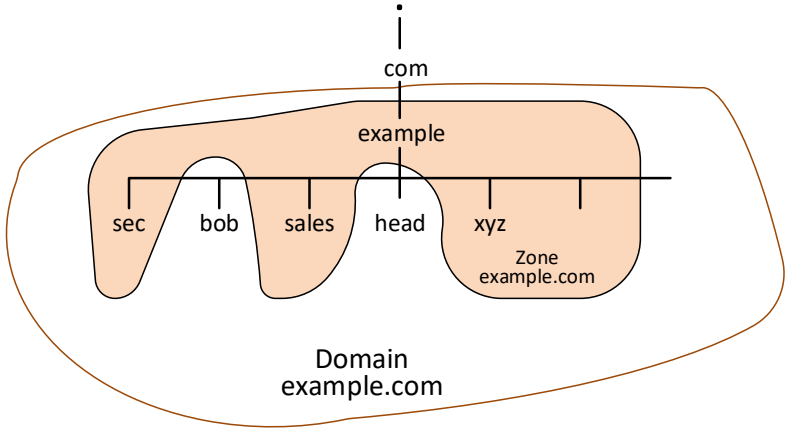



Figure 24.3 *Domain and zone*

The entire `example.com` domain can be managed by a single name server, or independent name servers can be designated for specific subdomains. In Figure 24.3, subordinate name servers have been set up for the subdomains `bob.example.com` and `head.example.com`.

The original name server continues to manage the `example.com` domain and its remaining subdomains, such as `sec.example.com`, `sales.example.com`, and `xyz.example.com`. In this context, the original name server administers the zone `example.com`.

A zone refers to a portion of the domain name space that is managed by a particular name server.

## 24.5 Name server

DNS is a distributed database that is distributed across name servers, which are DNS protocol servers.

Each **zone** has at least one **authoritative name server**, which acts as the original source of data for that zone. Only an authoritative name server can respond definitively that a requested DNS name does not exist.

An authoritative name server can be either a **master server** (also called a *primary name server*) or a **slave server** (also called a *secondary name server*). The database on the master server is typically configured manually by the **hostmaster**, while the database on the secondary server is replicated from the primary server using a **DNS zone transfer**.

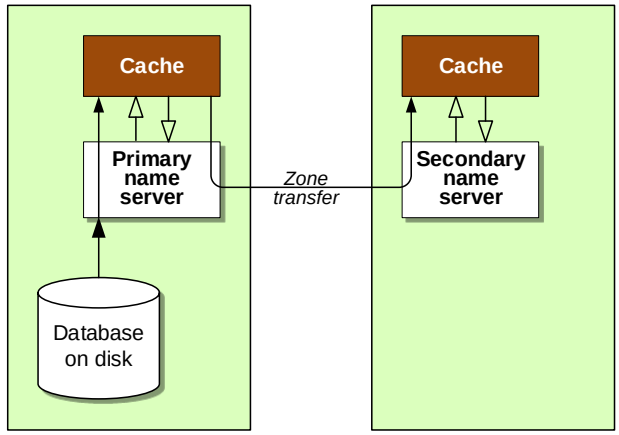



Figure 24.4 *Zone transfer*

A name server also maintains a **cache**, which stores both authoritative and non-authoritative data obtained from other name servers during DNS resolution.

## 24.6 Resolver

The client side of the DNS protocol is called a DNS resolver. A DNS resolver typically uses **recursive queries**, meaning it sends a request to a DNS server, which then queries other DNS servers on behalf of the requester as needed.

## 24.7 Recursive queries

A name server resolves queries using the data stored in its cache. If the required information is not found in the cache, the name server performs **iterative, non-recursive queries**. It begins the resolution process from the root domain (.), then proceeds to the top-level domain (TLD), and continues down the domain hierarchy until it reaches the authoritative server for the requested domain (Figure 24.5).

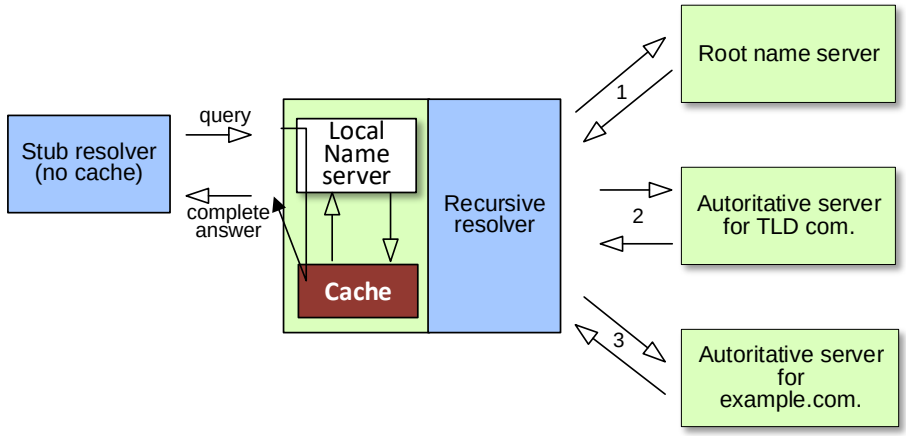



Figure 24.5 *Iterative resolution for host in zone* `example.com`

## 24.8 Resource records

Information about domain names and their corresponding IP addresses, as well as other data distributed via DNS, is stored in the memory (cache) of name servers in the form of **Resource Records (RRs)**. A name server populates its cache in several ways. Authoritative data is either read from configuration files on disk or obtained via a DNS zone transfer from another authoritative name server. As it processes individual DNS queries, the name server gradually acquires non-authoritative data from the caches of other servers.

When a DNS resolver needs to retrieve information, it requests RRs from a name server according to specific requirements. For example, a client can request an A record, which contains the IPv4 address of a given domain name. The client may be either a resolver or a name server that cannot resolve the query on its own.

RRs generally has the following formats:

| NAME | TTL | CLASS | TYPE | RDATA |
|---|---|---|---|---|

or

| NAME | CLASS | TTL | TYPE | RDATA |
|---|---|---|---|---|

**Example:**

```
www     3600    IN     AAAA  2001:db8::567:89ab
```

Where:

- `NAME` is the DNS name to be resolved by DNS. However, the official definition states that it is an **owner name**, i.e., the name of the node to which this resource record

pertains.

- `TYPE` is the type of RR.
- `CLASS` is usually `IN` (means internet).
- `TTL` (*Time-to-Live*) determines the lifetime (in seconds) of cached records by non-authoritative name servers. By setting a TTL value, administrators can control how long DNS information is stored on cache servers before it is updated from authoritative sources. This helps optimize DNS performance and ensures timely updates.
- `RDATA` describes the resource, it is TYPE specific.

**Some types of DNS records:**

- A `SOA` (*start of authority*) record is first record of zone which contains data about the zone.
- An `NS` (*name server*) record indicates the authoritative name server for a particular zone.
- An `A` (*address record*) record or address record associates a host name with an IPv4 address.
- An `AAAA` (*IPv6 address record*) record associates a host name with an IPv6 address. For example, a query for an AAAA record named K.ROOT-SERVERS.NET will return its IPv6 address as 2001:7fd::1.
- A `CNAME` (*canonical name record for an alias*) record maps one domain name (an alias) to another (canonical) name.
- An `MX` (*mail exchange*) record specifies the mail servers for a particular domain.
- A `TXT` (*text*) record is a special record that is displayed in text format and can contain any notes.
- A `PTR` (*pointer*) record contains information for reverse resolution.
- A `NAPTR` (*Name Authority Pointer*) record allows regular-expression-based rewriting of domain names which can then be used as URIs, further domain names to lookups (see subsection 15.9.1 NAPTR).
- A `DNSKEY` contains public part of Key Signing Key (KSK) or Zone Signing Key (ZSK) (see paragraph DNSSEC).
- A `DS` (*delegation signer*) record is used to identify the DNSSEC signing key of a delegated zone.
- A `RRSIG` record contains signature for a DNSSEC-secured record set.
- A `NSEC` (*next secure record*) record is used to prove an owner name does not exist (see DNSSEC).
- A `NSEC3` record allows one to prove the nonexistence of an owner name without permitting zonewalking.
- A `NSEC3PARAM` is a parameter record for use with NSEC3.

- An `OPT` record is a pseudo-RR utilized for exchanging parameters between the resolver and the nameserver; it should not be cached. Its functions include facilitating agreements on UDP datagrams longer than 512 bytes. Additionally, it indicates the DNSSEC.

**Example:** Easy example of `example.com` zone:

```
example.com.    3600 IN SOA ns.example.com hostmaster.example.com.(
                            123456
                            3600
                            300
                            3600000
                            3600
                            )
                3600 IN NS    ns.example.com.
                3600 IN MX    10 www.example.com.
ns              3600 IN AAAA  2001:db8::567:1
www             3600 IN AAAA  2001:db8::567:89ab
```

Round brackets allow the RR to continue on multiple lines. If the owner's name is not terminated by a dot, then the domain is supplemented from previous RRs.

We have two formats of RRs:

1. Text format is used by human administrators when entering zone data (see the example above).
2. The wire format is utilized by the resolver (DNS client) to communicate with the name server through the DNS protocol. The wire format is binary.

DNSSEC introduced the concept of a canonical form for Resource Records (RRs). This canonical form enables a unique and consistent ordering of RRs in the DNS cache, simplifying the determination of successor records. As a result, it becomes easier to perform searches within the list of **domain name owners**.

Moreover, the canonical ordering facilitates proof of non-existence for a given domain name. This is achieved using an NSEC record, which demonstrates that two existing RRs directly precede and follow the queried name in the sorted list—yet the queried name itself is absent. Because the canonical order guarantees a strict sequence, the absence of the queried name between the two proves it does not exist.

## 24.9 DNS protocol

The DNS protocol is a client–server application protocol that operates on a query–response basis. A client, called a resolver, sends a query to a server, called a name server, which responds with an answer. The DNS protocol supports several types of operations. The most commonly used operation is the DNS QUERY, which allows the retrieval of one or more Resource Records (RRs) from the DNS server. Other operations include DNS NOTIFY, DNS UPDATE, and others.

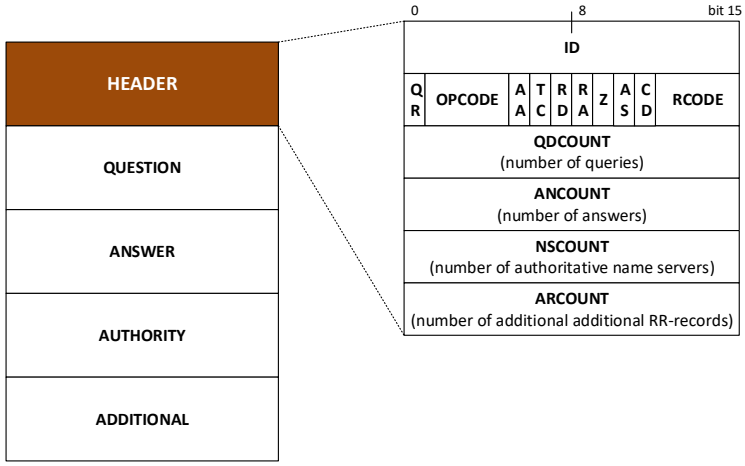



Figure 24.6 *Packet DNS QUERY*

A DNS QUERY packet (see Figure 24.6) consists of a header (see Table 24.1) and up to four sections that may carry RRs in wire format. Typically, a query includes only the QUESTION section, and sometimes the ADDITIONAL section. In contrast, a response can include all sections, including the QUESTION section (copied from the original query).

The Answer section contains Resource Records (RRs) that directly respond to the query. The Authority section includes RRs that identify the authoritative name servers for the queried domain. The Additional section provides supplementary RRs that are relevant to the query, such as IP addresses for the name servers listed in the Authority section, though they do not directly answer the original question.

Table 24.1 *Header items*

| Item | Meaning |
|---|---|
| QR | 0 – the message is a query; 1 – the message is an answer. |
| Opcode | The query type is the same in both the query and the answer:<br>0 – standard query (QUERY)<br>1 – inverse query (IQUERY)<br>2 – status query (STATUS)<br>4 – notify query (NOTIFY)<br>5 – update query (UPDATE)<br>6 – DNS Stateful Operations |
| AA | 0 – non-authoritative answer; 1 – authoritative answer. |
| TC | If set, the answer was truncated to 512 bytes. To obtain the full answer, the query must be resent via TCP. |
| RD | If set, a recursive resolution is requested (important for queries). |
| RA | If set, the server supports recursive resolution (important in responses). |
| Z | Reserved for future use. |
| AD | Set by the nameserver if all RRs in the Answer and Authority sections are considered authentic (DNSSEC). |
| CD | Indicates that the resolver is willing to accept unauthenticated data. Used in DNSSEC. Copied by nameservers in the response. |
| Rcode | Response code:<br>0 – No error (NoError)<br>1 – Format error (FormErr)<br>2 – Server failure (ServFail)<br>3 – Name error, i.e., domain does not exist (NXDomain, only from authoritative servers)<br>4 – Not implemented (NotImp)<br>5 – Query refused (Refused) |

## 24.10 DNSSEC

The need for implementing the DNSSEC extension arises from the lack of inherent protection in the DNS protocol. When the protocol was developed in the 1980s, security was not a primary consideration. As a result, no mechanisms were provided to verify the authenticity of responses from authoritative DNS servers. Currently, resolvers can only verify IP addresses, which is insufficient in today's threat landscape. Modern attack techniques enable the forging and manipulation of source IP addresses.

Attackers can exploit this vulnerability by impersonating authoritative servers, thereby redirecting users to malicious websites hosting prohibited or illegal content. DNSSEC (*Domain Name System Security Extensions*) [118] [119] [120] addresses this issue by providing reliable protection against server spoofing and DNS cache poisoning attacks.

DNSSEC is an extension of DNS designed to mitigate such attacks. It enhances DNS reliability through digital signatures based on public-key cryptography. These signatures verify the

authenticity of DNS responses. However, DNSSEC does not provide privacy for Resource Records (RRs) nor protect name servers themselves.

Public keys are published in Resource Records of type DNSKEY. Private keys are securely managed by the administrator of the primary name server.

In principle, a single pair of asymmetric keys would suffice for each DNSSEC-enabled zone. In practice, however, two separate key pairs are typically used:

- **Key Signing Key (KSK)**: This key is authenticated by the parent zone. The KSK is used to delegate security to child domains by inserting a DS-type RR (Delegation Signer) in the parent zone. This record contains a digest of the child zone's public KSK, which is then signed by the parent's Zone Signing Key (ZSK).
- **Zone Signing Key (ZSK)**: This key is used to sign the zone's RRs. The ZSK is signed with the KSK and also published in a DNSKEY record.

The DNSKEY record contains a `Flags` field (256 for ZSK, 257 for KSK), rather than the public key ID. The `Key Tag`—used to identify the corresponding public key—is calculated from the public key value.

Each signature is stored in a `RRSIG` Resource Record, which includes the Key Tag of the signing key. The signature is computed over a sequence of RRs in canonical order, all sharing the same `owner name`, `TTL`, `class`, and `type`.

**Example:** DNSSEC-Secured Zone:

```
example.com.      3600 IN SOA ns.example.com. hostmaster.example.com. (
                          123456 3600 300 3600000 3600 )
                  3600 IN RRSIG SOA signature using ZSK
                  3600 IN NS    ns.example.com.
                  3600 IN RRSIG NS  signature using ZSK
                  3600 IN MX    10 www.example.com.
                  3600 IN RRSIG MX  signature using ZSK
                  3600 IN NSEC  ns.example.com. AAAA RRSIG NSEC
                  3600 IN RRSIG NSEC signature using ZSK
                  3600 IN DNSKEY 256  ZSK public key
                  3600 IN DNSKEY 257  KSK public key
                  3600 IN RRSIG DNSKEY signature using ZSK
                  3600 IN RRSIG DNSKEY signature using KSK
ns.example.com.   3600 IN AAAA  2001:db8::567:1
                  3600 IN RRSIG AAAA signature using ZSK
                  3600 IN NSEC  www.example.com. AAAA RRSIG NSEC
                  3600 IN RRSIG NSEC signature using ZSK
www.example.com.  3600 IN AAAA  2001:db8::567:89ab
                  3600 IN RRSIG AAAA signature using ZSK
                  3600 IN NSEC  example.com. AAAA RRSIG NSEC
                  3600 IN RRSIG NSEC signature using ZSK
```

...

### 24.10.1 Zone Validation

Zone integrity checking is based on digital signatures. Every RR is accompanied by a corresponding RRSIG record, which can be verified using the public key in the DNSKEY record.

When a DNS client receives a response, it can validate the integrity of each record, whether from a cache or directly from the authoritative server. The validation for example above involves:

1. Retrieving the AAAA and RRSIG records via a DNS query.
2. Retrieving the public key ZSK via a DNSKEY query.
3. Verifying the signature using the ZSK.

To ensure the authenticity of the ZSK:

4. Retrieve the KSK using a DNSKEY query.
5. Retrieve the RRSIG over the ZSK using a DNSKEY query.
6. Verify the ZSK signature using the KSK.
7. Compute the digest of the KSK.
8. Retrieve the DS record from the parent zone.
9. Compare the computed digest with the one in the DS record—they must match.

This chain of trust can be validated recursively up to the DNS root. Each domain's KSK is referenced in the DS record of its parent zone. For the root zone, whose DS record is not stored elsewhere, the trust anchor must be obtained through a secure out-of-band channel.

### 24.10.2 Proof of Non-Existence

To prove the non-existence of a DNS name, DNSSEC uses the `NSEC` RR, which links a domain name to its immediate successor in canonical order. If the queried name logically falls between them, its absence is thus proven.

However, NSEC enables "**zone walking**"—enumerating the entire zone content. To mitigate this, `NSEC3` was introduced, which operates on hashes of names rather than the names themselves. These hashes are sorted and inserted into the cache in canonical order, making the content unreadable to humans—but tool-generated and machine-verified.

DNSSEC is exclusively utilized for signing and authenticating Resource Records (RRs) and does not encrypt network traffic. For securing DNS traffic, alternatives such as DNS over TLS (DoT) [121] or DNS over HTTPS (DoH) [122] should be considered.

### 24.10.3 DNS over TLS or HTTPS

DNS over TLS (DoT) [121] was initially developed to secure communication between stub resolvers and recursive name servers. However, its utility was expanded to include ensuring

secure zone transfers between authoritative name servers [123].

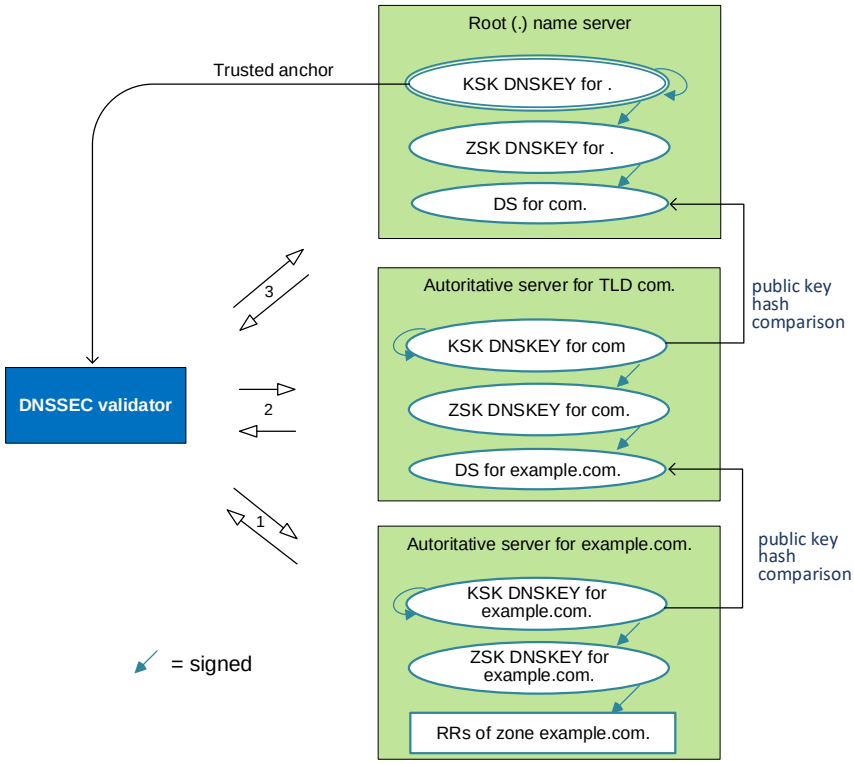



Figure 24.7 *DNSSEC Authentication Chain for zone example.com*

DNS over HTTPS (DoH) [122] can seem similar to DoT. The difference is that it operates over HTTPS with a HTTP header field:

```
content-type = application/dns-message
```

It enables DNS traffic to be tunneled over HTTPS, the same protocol is used for secure web browsing.

DoH prefers HTTP/2 or higher to utilize the PUSH method. DNS names (**owner names**) are encoded in Base64URL format.

Because internet browsers use DoH directly outside the operating system's services, concerns arise about its potential misuse. For example, Google operates DoH well known nameservers at IPv4 addresses 8.8.8.8 and 8.8.4.4 (for IPv6 it uses addresses 2001:4860:4860::8888 and 2001:4860:4860::8844), raising the possibility of connecting DNS resolution with third-party cookies. In such cases, it would be possible to aggregate a significant amount of information on these nameservers. Therefore, employers generally prohibit employees from using DoH on well-known nameservers.

### 24.10.4 Transport protocol and ports

Unlike most other application protocols, DNS uses both UDP and TCP as transport protocols. If the response does not fit into a UDP datagram, DNS packet is marked with the TC flag, and the resolver can retrieve the complete response by repeating the query using the TCP protocol. Initially, a UDP datagram could only be 512 bytes in size, but a larger UDP datagram can be negotiated using the pseudo-resource record OPT.

- The DNS protocol uses well-known ports 53/udp and 53/tcp.
- The DoT uses well-known port 853/tcp.
- The DoH uses well-known port 443/tcp (which directly incites to the use of 3rd party cookies).

### 24.10.5 DNSSEC deployment

As seen in example, DNSSEC significantly increases the requirements for the size of the DNS cache, as most Resource Records (RRs) are followed by an RSIG (Resource Record Signature) with a digital signature. For certain Top-Level Domains (TLDs), RRs may even reach several gigabytes in size. Implementing DNSSEC can be intricate and time-consuming, demanding meticulous planning and setup. It may also increase the load on DNS servers and enlarge the size of DNS responses, potentially impacting performance.

While DNSSEC is widely implemented in the root zone of the DNS system and in numerous top-level domains (TLDs), its deployment at lower levels of the DNS hierarchy, such as the domain name level, remains limited. This limitation can result in incomplete trust chains and potential security vulnerabilities.

DNS resolvers capable of verifying DNS data signatures to ensure authenticity can perform DNSSEC verification. However, not all DNS resolvers support DNSSEC, and some may conduct validation incorrectly or not at all. It is crucial to utilize a properly configured DNS resolver for DNSSEC validation (including trusted anchors).

### 24.10.6 DNSSEC Tools

There are several tools available for working with DNSSEC. Below is an overview of key utilities and systems that support DNSSEC functionality:

- **DNSSEC command-line utilities:** A set of command-line tools for managing DNSSEC. These include utilities for key generation, zone signing, and signature verification. They are compatible with most Unix-based systems.
- **OpenDNSSEC:** A DNSSEC management system that automates key generation, signing, and zone transfer processes. It includes a web-based management interface and supports most Unix-based systems.
- **BIND (Berkeley Internet Name Domain):** One of the most widely used DNS servers, BIND includes built-in support for DNSSEC. It supports both zone signing and verification and can be configured to work alongside other DNSSEC tools.
- **PowerDNS:** A DNS server that supports DNSSEC, DNS-over-TLS, and DNS-over-HTTPS.

It includes built-in features for zone signing and validation and can be integrated with additional DNSSEC utilities.

- D**nssec-Trigger:** A stub resolver with DNSSEC support for Unix-based systems. It includes built-in support for DNS-over-TLS and DNS-over-HTTPS and helps detect and mitigate DNS-based attacks.
- **DNS node:** A DNS server capable of supporting DNSSEC, DNS-over-TLS, and DNS-over-HTTPS. It includes built-in support for zone signing and authentication and can operate in both authoritative and recursive modes.
- **Unbound:** A validating, recursive, and caching DNS resolver with full DNSSEC support. It also supports DNS-over-TLS and DNS-over-HTTPS, providing secure DNS query resolution.
- **ldns:** A DNS programming library that includes tools for working with DNSSEC. It offers functionality for key generation, zone signing, signature verification, and more.
- **DNSViz:** A web-based tool that provides visual representations of DNSSEC-signed domains. It can be used to verify the chain of trust and identify potential security issues in DNS configurations (see Figure 24.8).
- **dig:** A command-line tool included with most Unix systems. It can perform DNS queries and retrieve DNSSEC-related information, such as key details and signature data.

### 24.10.7 File /etc/hosts

The /etc/hosts file is a valuable tool for network administrators. It can be used to map hostnames to IP addresses, allowing for manual control over hostname resolution. This file is particularly useful for restricting or redirecting access to specific hosts and services on a local machine or server.

Located on each machine, the /etc/hosts file is part of the local operating system and does not provide detailed information about the OS version or running services (contrary to common misconceptions). Instead, it serves as a static lookup table used by the operating system to resolve hostnames before querying DNS. It is often used for:

- Testing and troubleshooting network configurations.
- Blocking access to certain domains.
- Redirecting traffic for development purposes.
- Defining hostnames in environments without a DNS server.

### 24.10.8 DNSSEC Security Considerations and Best Practices

For managing secure DNS infrastructure, tools that support DNSSEC (Domain Name System Security Extensions) are essential. DNSSEC ensures the integrity and authenticity of DNS data by using digital signatures. However, it can be complex to configure and maintain.

It is highly recommended to consult with DNSSEC experts or follow well-documented best practices and official guidelines when deploying a DNSSEC-enabled infrastructure. Proper

implementation helps protect against DNS spoofing and other attacks, but requires careful planning, especially in key management and trust chain validation.

DNSSEC relies on the secure management of cryptographic keys to ensure the authenticity and integrity of DNS data. If a key is compromised, an attacker can generate fraudulent signatures and deliver false DNS information. Effective key management is complex and requires careful planning and coordination among DNS operators. Best practices include using strong, unique keys, rotating keys regularly, and ensuring that keys are stored securely.

The reliability of cryptographic keys is a critical aspect of DNSSEC security. Compromised or stolen keys may allow attackers to spoof DNS data and redirect traffic to malicious sites. Keys that are too short are vulnerable to brute-force attacks, while overly long keys may degrade DNS resolution performance. Therefore, selecting an appropriate key length is essential to balance security and efficiency. Keys should be stored securely, access should be strictly controlled, and key usage should be audited regularly to detect potential compromises.

DNSSEC is a complex protocol that must be carefully configured to operate effectively. Misconfigurations can result in incomplete trust chains and can create security gaps. It's vital to follow established deployment guidelines and thoroughly test DNSSEC implementations to ensure correct functionality. While DNSSEC protects against unauthorized tampering with zone files, a compromised signing key would still allow an attacker to modify zone data and generate valid signatures. Regularly monitoring zone files for unauthorized changes and auditing signing keys for signs of compromise are essential security practices.

DNSSEC can also introduce performance and operational challenges. Because DNSSEC responses are typically larger than non-DNSSEC responses, there is a higher risk of Denial of Service (DoS) attacks. Attackers may exploit DNSSEC to overload DNS servers with excessive traffic. To mitigate this, organizations should monitor DNS traffic and implement DoS countermeasures such as traffic filtering and rate limiting.

Some cryptographic algorithms used in DNSSEC, such as RSA, have known vulnerabilities (e.g., susceptibility to factoring attacks). It is important to use strong, modern cryptographic algorithms and regularly update DNSSEC implementations to ensure they align with current security standards.

Another challenge is the limited adoption of DNSSEC. At the domain level, deployment remains relatively low, leading to incomplete trust chains and reduced overall effectiveness. Until wider adoption occurs, DNSSEC's benefits in providing end-to-end DNS security may be limited.

In addition, not all DNS resolvers and clients support DNSSEC, and even if they do, they may perform validation improperly. This can cause resolution failures and undermine the security benefits of DNSSEC. It is important to use resolvers and clients that are properly configured to support DNSSEC validation.

### 24.10.9 Conclusion

Overall, DNSSEC adds a valuable layer of security to DNS infrastructure, helping to prevent tampering, spoofing, and man-in-the-middle attacks. However, it must be carefully planned, implemented, and monitored. Key areas of concern include key management, key compro-

mise, zone file integrity, algorithm strength, DoS vulnerabilities, and client support.

By adhering to best practices—such as using strong cryptographic keys, enforcing secure key storage and access controls, regularly auditing DNSSEC implementations, and deploying complementary security measures—organizations can significantly reduce the risks associated with DNSSEC and improve the resilience of their DNS systems.

Despite its complexities and limitations, DNSSEC remains a crucial tool in the defense against DNS-based threats. With appropriate management and oversight, it can effectively enhance the security and trustworthiness of DNS operations.

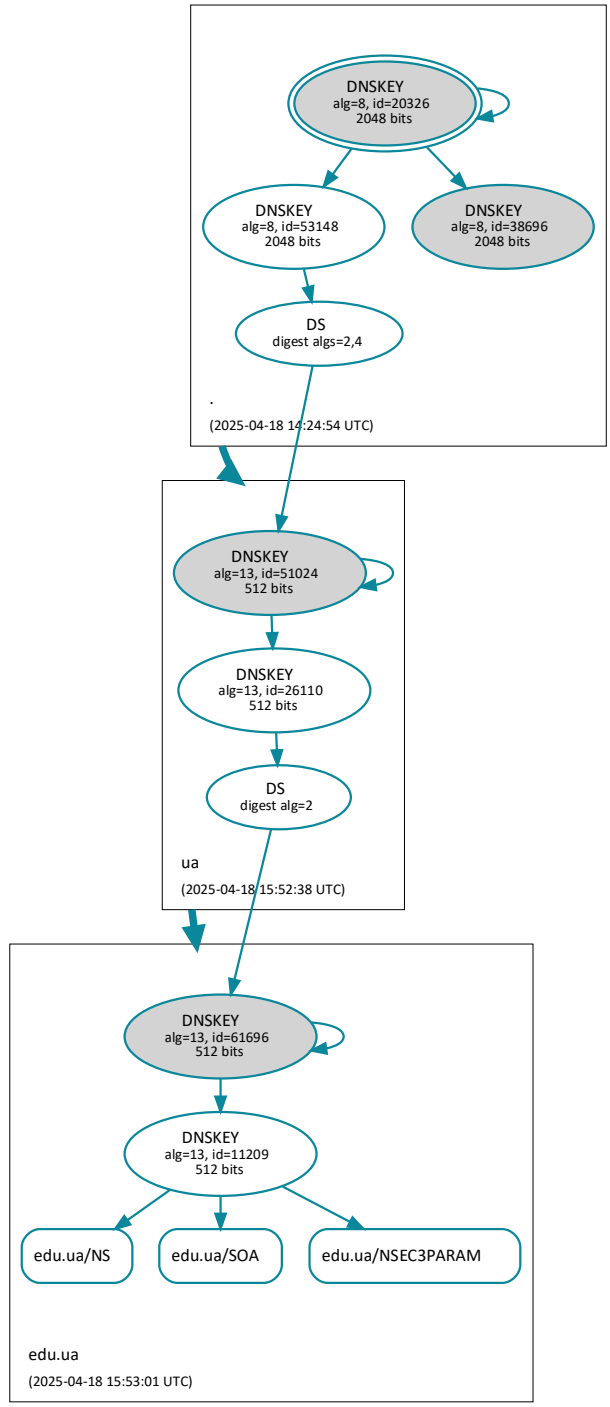



Figure 24.8 *DNSviz.net: DNSSEC Authentication Chain for zone edu.ua.*

# Bibliography


[1] J.-J. Quisquater and F. Koeune. *Side channel attacks*. Oct. 2002. URL: https://www.cryptrec.go.jp/exreport/cryptrec-ex-1047-2002.pdf.

[2] I.V. Vasyltsov. *Ataky spetsialnogo vydu na kryptoprystroi ta metody borotby z nymy*. 2009.

[3] I.V. Vasyltsov and L.O. Dubchak. *Metody zakhystu proty atak spetsialnogo vydu*. 2007.

[4] E. Toreini, B. Randell, and F. Hao. *An Acoustic Side Channel Attack on Enigma*. 2015.

[5] K. Gandolfi, C. Mourtel, and F. Olivier. *Electromagnetic analysis: Concrete results.* 2001.

[6] *FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION, Secure Hash Standard (SHS)*. Aug. 2015. URL: https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.180-4.pdf.

[7] H. Krawczyk, M. Bellare, and R. Canetti. *HMAC: Keyed-Hashing for Message Authentication*. Feb. 1997. URL: https://www.rfc-editor.org/rfc/rfc2104.html.

[8] 1QBit. *A Bit or Two About Qubits*. URL: https://1qbit.com/blog/quantum-computing/a-bit-or-two-about-qubits/.

[9] Andreas Huelsing et al. *RFC 8391 - XMSS: eXtended Merkle Signature Scheme*. May 2018. URL: https://www.rfc-editor.org/info/rfc8391.

[10] David McGrew, Michael Curcio, and Scott Fluhrer. *RFC 8554 - Leighton-Micali Hash-Based Signatures*. Apr. 2019. URL: https://www.rfc-editor.org/info/rfc8554.

[11] David Cooper et al. *NIST SP 800-208 - Recommendation for Stateful Hash-Based Signature Schemes*. Oct. 2020. URL: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-208.pdf.

[12] *FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION, Module-Lattice-Based Key-Encapsulation Mechanism Standard (ML-KEM)*. Aug. 2024. URL: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.203.pdf.

[13] *FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION, Module-Lattice-Based Digital Signature Standard (ML-DSA)*. Aug. 2024. URL: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.204.pdf.

[14] Matthew Green. *Hash-based Signatures: An illustrated Primer*. Apr. 2018. URL: https://blog.cryptographyengineering.com/2018/04/07/hash-based-signatures-an-illustrated-primer/.

[15] Dan Boneh and Victor Shoup. *Applied Cryptography*. 2021.

[16] *FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION, Stateless Hash-Based Digital Signature Standard (SLH-DSA)*. Aug. 2024. URL: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.205.pdf.

[17] *FALCON - Fast-Fourier Lattice-based Compact Signatures over NTRU*. URL: https://falcon-sign.info/.

[18] NIST. *NIST SP 800-56C Rev. 2 (Recommendation for Key-Derivation Methods in Key-Establishment Schemes)*. Aug. 2020. URL: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Cr2.pdf.

[19] Manuel Barbosa et al. "X-Wing: The Hybrid KEM You've Been Looking For". In: *Cryptology ePrint Archive* (2024).

[20] C. Tjhai et al. *RFC9370 - Multiple Key Exchanges in the Internet Key Exchange Protocol Version 2 (IKEv2)*. May 2023. URL: https://www.rfc-editor.org/info/rfc9370.

[21] ITU-T. *X.509 (Information technology – Open Systems Interconnection – The Directory: Public-key and attribute certificate frameworks)*. Oct. 2019. URL: https://www.itu.int/rec/T-REC-X.509-201910-I.

[22] Corey Bonnell et al. *Internet-Draft - A Mechanism for Encoding Differences in Paired Certificates*. Dec. 2024. URL: https://datatracker.ietf.org/doc/draft-bonnell-lamps-chameleon-certs/05.

[23] S. Josefsson. *RFC 4648, The Base16, Base32, and Base64 Data Encodings*. Oct. 2006.

[24] N. Freed, N. Borenstein, and N. Borenstein. *RFC 2045, Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*. Nov. 1996. URL: http://www.rfc-editor.org/rfc/rfc2045.txt.

[25] P. Karn and W. Simpson. *Photuris: Session-Key Management Protocol*. Mar. 1999. URL: https://www.rfc-editor.org/rfc/rfc2522.html.

[26] J. Arkko and H. Haverinen. *RFC 4187 - Extensible Authentication Protocol Method for 3rd Generation, Authentication and Key Agreement (EAP-AKA)*. Jan. 2006. URL: https://www.rfc-editor.org/rfc/rfc4187.html.

[27] J. Arkko and P. Eronen. *RFC 5448 - Improved Extensible Authentication Protocol Method for 3rd Generation Authentication and Key Agreement (EAP-AKA')*. May 2009. URL: https://www.rfc-editor.org/rfc/rfc5448.

[28] *5G; Security architecture and procedures for 5G System (3GPP TS 33.501 version 16.3.0 Release 16)*. Aug. 2020. URL: https://www.etsi.org/deliver/etsi_ts/133500_133599/133501/16.03.00_60/ts_133501v160300p.pdf.

[29] *3GPP TS 35.205 v14.0.0 - 3G Security; Specification of the MILENAGE algorithm set: An example algorithm set for the 3GPP authentication and key generation functions f1, f1*, f2, f3, f4, f5 and f5*; Document 1: General; 3GPP TS 35.205,* Mar. 2017. URL: http://www.3gpp.org.

[30] J. Arkko and H. Haverinen. *3GPP TS 35.206 v 14.0.0 - 3G Security; Specification of the MILENAGE algorithm set: An example algorithm set for the 3GPP authentication and key generation functions f1, f1*, f2, f3, f4, f5 and f5*; Document 2: Algorithm specification; 3GPP TS 35.206*. Mar. 2017. URL: http://www.3gpp.org.

[31] D. Cooper et al. *RFC 5280 - Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. May 2008. URL: https://www.rfc-editor.org/rfc/rfc5280.html.

[32] David Crocker. *RFC 822, ARPA INTERNET TEXT MESSAGES*. Aug. 1982. URL: https://www.rfc-editor.org/rfc/rfc822.html.

[33] S. Chokhani et al. *RFC 3647: Internet X.509 Public Key Infrastructure, Certificate Policy and Certification Practices Framework*. Accessed: 2025-05-06. Nov. 2003. URL: https://www.rfc-editor.org/rfc/rfc3647.html.

[34] John Linn. *RFC 989 :Privacy Enhancement for Internet Electronic Mail*. 1987. URL: https://www.rfc-editor.org/rfc/rfc989.html.

[35] J. Linn. *Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures*. Feb. 1993. URL: https://www.rfc-editor.org/rfc/rfc1421.html.

[36] B. Kaliski. *RFC 1424, Privacy Enhancement for Internet Electronic Mail: Part IV: Key Certification and Related Services*. Feb. 1993.

[37] B. Kaliski. *RFC 2314, PKCS #10: Certification Request Syntax, Version 1.5*. Mar. 1998. URL: https://www.rfc-editor.org/rfc/rfc2314.html.

[38] M. Nystrom and B. Kaliski. *RFC 2986, PKCS #10: Certification Request Syntax Specification, Version 1.7*. Nov. 2000. URL: https://www.rfc-editor.org/rfc/rfc2986.html.

[39] J. Schaad. *RFC 4211, Internet X.509 Public Key Infrastructure, Certificate Request Message Format (CRMF)*. Sept. 2005. URL: https://www.rfc-editor.org/rfc/rfc4211.html.

[40] J. Schaad and M. Myers. *RFC 5272: Certificate Management over CMS (CMC)*. 2008. URL: https://www.rfc-editor.org/rfc/rfc5272.html.

[41] C. Adams et al. *RFC 4210, Internet X.509 Public Key Infrastructure, Certificate Management Protocol (CMP)*. Sept. 2005. URL: https://www.rfc-editor.org/rfc/rfc4210.html.

[42] European Union Agency for Railways (ERA). *SUBSET-146, ERTMS End-to-End Security*. 2023. URL: https://www.era.europa.eu/system/files/2023-09/index010d_-_SUBSET-146_v400.pdf.

[43] S. Santesson et al. *RFC 6960, X.509 Internet Public Key Infrastructure, Online Certificate Status Protocol - OCSP*. June 2013. URL: https://www.rfc-editor.org/rfc/rfc6960.html.

[44] R. Barnes et al. *RFC 8555, Automatic Certificate Management Environment (ACME)*. Mar. 2019. URL: https://www.rfc-editor.org/rfc/rfc8555.html.

[45] R. Housley. *RFC 5652: Cryptographic Message Syntax (CMS)*. 2009. URL: https://www.rfc-editor.org/rfc/inline-errata/rfc5652.html.

[46] D. Mills et al. *RFC 5905, Network Time Protocol Version 4: Protocol and Algorithms Specification*. June 2010. URL: https://www.rfc-editor.org/rfc/rfc5905.txt.

[47] C. Adams et al. *RFC 3161, Internet X.509 Public Key Infrastructure, Time-Stamp Protocol (TSP)*. Aug. 2001. URL: https://www.rfc-editor.org/rfc/rfc3161.html.

[48] D. Pinkas, N. Pope, and J. Ross. *Policy Requirements for Time-Stamping Authorities (TSAs)*. Nov. 2003. URL: https://www.rfc-editor.org/rfc/rfc3628.html.

[49] R. Fielding, M. Nottingham, and J. Reschke. *RFC 9112, HTTP/1.1*. June 2022. URL: https://www.rfc-editor.org/rfc/rfc9112.html.

[50] M. Thomson and C. Benfield. *RFC 9113, HTTP/2*. June 2022. URL: https://www.rfc-editor.org/rfc/rfc9113.html.

[51] M. Bishop. *RFC 9114, HTTP/3*. June 2022. URL: https://www.rfc-editor.org/rfc/rfc9114.html.

[52] R. Fielding, M. Nottingham, and J. Reschke. *RFC 9110, HTTP Semantics*. June 2022. URL: https://www.rfc-editor.org/rfc/rfc9110.html.

[53] R. Peon and H. Ruellan. *RFC 7541*. May 2015. URL: https://www.rfc-editor.org/rfc/rfc7541.html.

[54] C. Krasic, M. Bishop, and A. Frindell. *RFC 9204, QPACK: Field Compression for HTTP/3*. June 2022. URL: https://www.rfc-editor.org/rfc/rfc9204.html.

[55] P. Resnick. *RFC 5322: Internet Message Format*. 2008. URL: https://www.rfc-editor.org/rfc/rfc5322.html.

[56] T. Berners-Lee, R. Fielding, and L. Masinter. *RFC 3986, Uniform Resource Identifier (URI): Generic Syntax*. Jan. 2005. URL: https://www.rfc-editor.org/rfc/rfc3986.html.

[57] A. Phillips and M. Davis. *RFC 5646: Tags for Identifying Languages*. 2009. URL: https://www.rfc-editor.org/rfc/rfc5646.html.

[58] K. Jaganathan, L. Zhu, and J. Brezak. *RFC 4559: SPNEGO-based Kerberos and NTLM HTTP Authentication in Microsoft Windows*. 2006. URL: https://www.rfc-editor.org/rfc/rfc4559.html.

[59] R. Fielding, M. Nottingham, and J. Reschke. *RFC 9111, HTTP Caching*. June 2022. URL: https://www.rfc-editor.org/rfc/rfc9111.html.

[60] A. Barth. *RFC 6265, HTTP State Management Mechanism*. Apr. 2011. URL: https://www.rfc-editor.org/rfc/rfc6265.html.

[61] C. Neuman, S. Hartman, and K. Raeburn. *RFC 4120: The Kerberos Network Authentication Service (V5)*. 2005. URL: https://www.rfc-editor.org/rfc/rfc4120.html.

[62] L. Zhu and B. Tung. *RFC 4556, Public Key Cryptography for Initial Authentication in Kerberos (PKINIT)*. June 2006.

[63] K. Moriarty, B. Kaliski, and A. Rusch. *RFC 8018, PKCS #5: Password-Based Cryptography Specification*. Jan. 2017.

[64] M. Jenkins, M. Peck, and K. Burgin. *RFC 8009, AES Encryption with HMAC-SHA2 for Kerberos 5*. Oct. 2016.

[65] K. Jaganathan, L. Zhu, and J. Brezak. *RFC 4559, SPNEGO-based Kerberos and NTLM HTTP Authentication in Microsoft Windows*. June 2006. URL: https://www.rfc-editor.org/rfc/rfc4559.html.

[66] M. Jones, J. Bradley, and N. Sakimura. *RFC 7519, JSON Web Token (JWT)*. May 2015. URL: https://www.rfc-editor.org/rfc/rfc7519.html.

[67] M. Jones, J. Bradley, and N. Sakimura. *RFC 7515: JSON Web Signature (JWS)*. 2015. URL: https://datatracker.ietf.org/doc/html/rfc7515.

[68] V. Bertocci. *RFC 9068, JSON Web Token (JWT) Profile for OAuth 2.0 Access Tokens*. Oct. 2021. URL: https://www.rfc-editor.org/rfc/rfc9068.html.

[69] J. Iyengar and M. Thomson. *RFC 9000, QUIC: A UDP-Based Multiplexed and Secure Transport*. May 2021. URL: https://www.rfc-editor.org/rfc/rfc9000.html.

[70] J. Postel. *Transmission Control Protocol*. Sept. 1981. URL: http://www.rfc-editor.org/rfc/rfc793.txt.

[71] W. Eddy. *RFC 9293, Transmission Control Protocol (TCP)*. Aug. 2022. URL: https://www.rfc-editor.org/rfc/rfc9293.html.

[72] R. Braden. *RFC 1379, Extending TCP for Transactions – Concepts*. Nov. 1992. URL: https://www.rfc-editor.org/rfc/rfc1379.html.

[73] R: Stewart, M. T□xen, and K: Nielsen. *RFC 9260, Stream Control Transmission Protocol*. June 2022. URL: https://www.rfc-editor.org/rfc/rfc9260.html.

[74] *Introduction to CCITT Signaling system No. 7*. Mar. 1993. URL: http://www.itu.int.

[75] M. Thomson and S. Turner. *RFC 9001, Using TLS to Secure QUIC*. May 2021. URL: https://www.rfc-editor.org/rfc/rfc9001.html.

[76] D. McGrew. *RFC 5116, An Interface and Algorithms for Authenticated Encryption*. Jan. 2008. URL: https://www.rfc-editor.org/rfc/rfc5116.html.

[77] Y. Nir and A. Langley. *RFC 8439, ChaCha20 and Poly1305 for IETF Protocols*. June 2018. URL: https://www.rfc-editor.org/rfc/rfc8439.html.

[78] D. McGrew and D. Bailey. *RFC 6655, AES-CCM Cipher Suites for Transport Layer Security (TLS)*. July 2012. URL: https://www.rfc-editor.org/rfc/rfc6655.html.

[79] E. Rescorla, H. Tschofenig, and N. Modadugu. *RFC 9147, The Datagram Transport Layer Security (DTLS) Protocol, Version 1.3*. Apr. 2022. URL: https://www.rfc-editor.org/rfc/rfc9147.html.

[80] R. Stewart, M. T□xen, and K Nielsen. *RFC 9260, Stream Control Transmission Protocol*. June 2022. URL: https://www.rfc-editor.org/rfc/rfc9260.html.

[81] J. Postel. *User Datagram Protocol*. Aug. 1980. URL: http://www.rfc-editor.org/rfc/rfc768.txt.

[82] J. Rosenberg et al. *RFC 3261, SIP: Session Initiation Protocol*. June 2002. URL: http://www.rfc-editor.org/rfc/rfc3261.txt.

[83] H. Schulzrinne. *RFC 5031, A Uniform Resource Name (URN) for Emergency and Other Well-Known Services*. Jan. 2008. URL: https://www.rfc-editor.org/rfc/rfc5031.html.

[84] P. Resnick. *RFC 5322, Internet Message Format*. Oct. 2008. URL: https://www.rfc-editor.org/rfc/rfc5322.html.

[85] F. Andreasen, M. Baugher, and D. Wing. *Session Description Protocol (SDP), Security Descriptions for Media Streams*. July 2006. URL: http://www.rfc-editor.org/rfc/rfc4568.txt.

[86] P. Zimmermann, A. Johnston, and J. Callas. *ZRTP: Media Path Key Agreement for Unicast Secure RTP*. Apr. 2011. URL: http://www.rfc-editor.org/rfc/rfc6189.txt.

[87] J. Arkko et al. *MIKEY: Multimedia Internet KEYing*. Aug. 2004. URL: http://www.rfc-editor.org/rfc/rfc3830.txt.

[88] I. Fette and A. Melnikov. *RFC 6455, The WebSocket Protocol*. Dec. 2011. URL: https://www.rfc-editor.org/rfc/rfc6455.html.

[89] J. Peterson. *RFC 3764: enumservice registration for Session Initiation Protocol (SIP), Addresses-of-Record*. 2004. URL: https://datatracker.ietf.org/doc/html/rfc3764.

[90] *3GPP TS 26.114: IP Multimedia Subsystem (IMS); Multimedia Telephony; Media handling and interaction*. Sept. 2023. URL: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=1404.

[91] *Lawful interception architecture and functions*. Dec. 2015. URL: http://www.3gpp.org.

[92] *Infrastructure of audiovisual services – Communication procedures, Gateway control protocol: Version 3*. Mar. 2013. URL: Http://www.itu.int/rec/T-REC-H.248.1.

[93] M. Handley, V. Jacobson, and C. Perkins. *SDP: Session Description Protocol*. Jan. 2021. URL: https://www.rfc-editor.org/rfc/rfc8866.html.

[94] H. Schulzrinne et al. *Real Time Streaming Protocol Version 2.0*. Jan. 2016. URL: https://www.rfc-editor.org/rfc/rfc7826.txt.

[95] H. Schulzrinne et al. *RTP: A Transport Protocol for Real-Time Applications*. July 2003. URL: http://www.rfc-editor.org/rfc/rfc3550.txt.

[96] M. Baugher et al. *RFC 3711: The Secure Real-time Transport Protocol (SRTP)*. Mar. 2004. URL: https://www.rfc-editor.org/rfc/rfc3711.html.

[97] Jonathan Postel. *RFC 821, SIMPLE MAIL TRANSFER PROTOCOL*. Aug. 1982. URL: https://www.rfc-editor.org/rfc/rfc821.html.

[98] J. Klensin. *RFC 5321, Simple Mail Transfer Protocol*. Oct. 2008. URL: https://www.rfc-editor.org/rfc/rfc5321.html.

[99] P. Resnick. *RFC 5322, Internet Message Format*. Oct. 2008. URL: https://www.rfc-editor.org/rfc/rfc5322.html.

[100] J. Klensin and Y. Ko. *Overview and Framework for Internationalized Email*. Feb. 2012. URL: https://www.rfc-editor.org/rfc/rfc6530.txt.

[101] J. Yao and W. Mao. *SMTP Extension for Internationalized Email*. Feb. 2012. URL: https://www.rfc-editor.org/rfc/rfc6531.txt.

[102] A. Yang, S. Steele, and N. Freed. *Internationalized Email Headers*. Feb. 2012. URL: https://www.rfc-editor.org/rfc/rfc6532.txt.

[103] T. Hasen, C. Newman, and A. Melnikov. *Internationalized Delivery Status and Disposition Notifications*. Feb. 2012. URL: https://www.rfc-editor.org/rfc/rfc6533.txt.

[104] K. Moore. *RFC 2047, MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text*. Nov. 1996. URL: https://www.rfc-editor.org/rfc/rfc2047.html.

[105] A. Costello. *RFC 3492, Punycode: A Bootstring encoding of Unicode for Internationalized Domain Names in Applications (IDNA)*. Mar. 2003. URL: https://www.rfc-editor.org/rfc/rfc3492.txt.

[106] J. Klensin. *RFC 5890, Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework*. Aug. 2010. URL: https://www.rfc-editor.org/rfc/rfc5890.txt.

[107] R. Siemborski and A. Melnikov. *RFC 4954, SMTP Service Extension for Authentication*. July 2007. URL: https://www.rfc-editor.org/rfc/rfc4954.txt.

[108] J. Sermersheim. *RFC 4511, Lightweight Directory Access Protocol (LDAP): The Protocol*. June 2006. URL: https://www.rfc-editor.org/rfc/rfc4511.txt.

[109] J. Myers and M. Rose. *RFC 1939, Post Office Protocol - Version 3*. May 1996. URL: https://www.rfc-editor.org/rfc/rfc1939.html.

[110] R. Gellens, C. Newman, and L. Lundblade. *RFC 2449: POP3 Extension Mechanism*. Nov. 1998. URL: https://www.rfc-editor.org/rfc/rfc2449.html.

[111] A. Melnikov and B. Leiba. *RFC 9051, Internet Message Access Protocol (IMAP) - Version 4rev2*. Aug. 2021. URL: https://www.rfc-editor.org/rfc/rfc9051.html.

[112] N. Freed and N. Borenstein. *RFC 2049, Multipurpose Internet Mail Extensions (MIME) Part Five: Conformance Criteria and Examples*. Nov. 1996. URL: https://www.rfc-editor.org/rfc/rfc2049.html.

[113] J. Schaad, B. Ramsdell, and S. Turner. *RFC 8550, Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 4.0, Certificate Handling*. Apr. 2019. URL: https://www.rfc-editor.org/rfc/rfc8550.txt.

[114] J. Schaad, B. Ramsdell, and S. Turner. *RFC 8551, Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 4.0, Message Specification*. Apr. 2019. URL: https://www.rfc-editor.org/rfc/rfc8551.txt.

[115] P.. Hoffman. *RFC 2634, Enhanced Security Services for S/MIME*. June 1999. URL: https://www.rfc-editor.org/rfc/rfc2634.txt.

[116] P. Mockapetris. *RFC 1034: Domain names - concepts and facilities*. Nov. 1987. URL: https://www.rfc-editor.org/rfc/rfc1034.html.

[117] P. Mockapetris. *RFC1035: Domain names - implementation and specification*. Nov. 1987. URL: https://www.rfc-editor.org/rfc/rfc1035.html.

[118] R. Arends et al. *RFC 4033: DNS Security Introduction and Requirements*. 2005. URL: https://www.rfc-editor.org/rfc/rfc4033.txt.

[119] R. Arends et al. *RFC 4034: Resource Records for the DNS Security Extensions*. URL: https://www.rfc-editor.org/rfc/rfc4034.txt.

[120] R. Arends et al. *RFC 4035: Protocol Modifications for the DNS Security Extensions*. URL: https://www.rfc-editor.org/rfc/rfc4035.txt.

[121] Z. Hu et al. *RFC 7858: Specification for DNS over Transport Layer Security*. May 2016. URL: https://www.rfc-editor.org/info/rfc7858.

[122] P. Hoffman and P. McManus. *RFC 8484: DNS Queries over HTTPS (DoH*. Oct. 2018. URL: https://www.rfc-editor.org/info/rfc8484.

[123] W. Toorop et al. *RFC 9103: DNS Zone Transfer over TLS*. Aug. 2021. URL: https://www.rfc-editor.org/rfc/rfc9103.html.

[124] R. Atkinson. *RFC 1825 - Security Architecture for the Internet Protocol*. 1995. URL: https://www.rfc-editor.org/rfc/rfc1825.html.

[125] S. Kent and K. Seo. *RFC 4301 - Security Architecture for the Internet Protocol*. 2005. URL: https://www.rfc-editor.org/rfc/rfc4301.html.

[126] C. Kaufman et al. *RFC 7296 - Internet Key Exchange Protocol Version 2 (IKEv2)*. 2005. URL: https://www.rfc-editor.org/rfc/rfc7296.html.

[127] S. Kent. *RFC 4303: IP Encapsulating Security Payload (ESP)*. Dec. 2005. URL: https://www.rfc-editor.org/rfc/rfc4303.html.

[128] C. Kaufman et al. *RFC 7296: Internet Key Exchange Protocol Version 2 (IKEv2)*. Oct. 2014. URL: https://www.rfc-editor.org/rfc/rfc7296.html.

# Index


ACME *(Automatic Certificate Management Environment)*, 86
  external account binding, 86
AEAD, 22
algorithm
  (EC)DSA, 27
  AKA *(Authentication and Key Agreement)*, 34
  Base64, 30
  Base64/MIME, 31
  Base64/URL, 32
  DH *(Diffie-Hellman)*, 24
  ECDH, 25
  ECDSA, 28
  EdDSA, 28
  key excange, 24
  Photuris, 32
  RSA, 24, 26
  Shamir's secret sharing, 32
AS *(Authentication server)*, 137
authentication
  by smartcard (PKINIT), 141
  pre-authentication, 140
authentication tag, 22, 23, 28
Authorization Server, 153
availability, 2
CA, 67
  fake, 122
  tree, 59
CA *(certification authority)* , 40
CBC, 20
certificate, 43
  chain, 59, 60
  domain validation (DV), 86
  extended validation (EV), 86
  extension, 48
  lifecycle, 63
  of public key, 40
  qualified, 58
  qualified for web servers, 86
  rekey, 64
  renew, 64
  request, 68
  revocation, 77
  root, 59
  self-issued, 42
  self-signed, 42
  self-signed , 70
  verifying validity , 61
certification
  public key, 40
cipher
  AEAD, 20, 22
  asymmetric, 23
  block, 18
  hybrid encryption, 25
  mode of operation, 19
  mode of operation CBC, 20
  mode of operation ECB, 19
  mode of operation GCM, 22
  RSA, 24
  stream, 21
  symmetric, 18
CMAC, 20, 28
CMS *(Cryptographic Message Syntax)* , 91
CMS Signed Data, 29
codec, 225
confidentiality, 1
Control Plane, 219
CRL *(certificate revocation list)*, 78, 79
  delta (incremental), 78
  direct, 78
  entry extension, 81
  extension, 81
  full, 78
  indirect, 78
  reason code, 81
  remove from, 78

CRMF, 73
CSR *(certificate signing request)* , 41, 70
digest, 14
distinguished name, 45
DNS
  RR *(resource records)*, 275
  DoH *(DNS over HTTPS)*, 282
  domain, 271
  DoT *(DNS over TLS)*, 281
  ENUM, 207
  internationalized domain name, 272
  name, 271
  name server, 274
  recursive queries, 274
  resolver, 274
  reverse domains, 272
  RR A, 276
  RR AAAA, 276
  RR CNAME, 276
  RR DNSKEY, 276
  RR DS, 276
  RR MX, 243, 276
  RR NAPTR, 206, 276
  RR NS, 276
  RR NSEC, 276, 281
  RR NSEC3, 276
  RR NSEC3PARAM, 276
  RR OPT, 277
  RR PTR, 276
  RR RRSIG, 276
  RR SOA, 276
  RR TXT, 276
  subdomain, 271
  zone, 273
DNS *(Domain Name System)*, 271
DNSSEC
  Proof of Non-Existence, 281
  Zone Validation, 281
DTBS/R *(Data To Be Signed / Representation)*, 268
DTLS
  CID *(Connection ID)*, 188
  epoch, 187
ECB *(Electronic Codebook)*, 19
ECC *(Elliptic Curve Cryptography)*, 25
ephemeris second, 100
GCM *(Galois/Counter Mode)*, 22
GHASH, 23
GMAC, 23, 28
GMT *(Greenwich Mean Time)*, 100
H.248
  Context, 220
  Termination, 219
hash, 13
HMAC, 16, 28
Home network, 214
HSM *(Hardware Security Module)*, 67, 103
HTTP
  authentication, 129
  cache, 130
  cookie, 132
  Gateway, 122
  Intermediaries, 119
  methods, 125
  Proxy, 120
  Reverse proxy, 123
  status codes, 125
  Tunnel, 124
Identity Federation, 152
IdM *(Identity Management)*, 160
IdM *(Identity management)*, 161
IdP *(Identity Provider)*, 152
IMAP4
  Authenticated state, 253
  Not Authenticated State, 253
  over TLS, 235, 252
  Selected state, 254
  TLS handshake initiated by server, 253
imprint, 14, 98
IMS *(IP Multimedia Subsystem)*, 211
integrity, 2
Internet Message Format, 232
jitter, 225
JSON *(JavaScript Object Notation)*, 153
JWS *(JSON Web Signature)*, 153
JWT *(JSON Web Token)*, 153
KDC *(Key Distribution Center)*, 137
Kerberos
  forwarding, 143
  proxy, 142
  ticket, 147
  timers, 147
  trust between realms, 145
Lawful Interception, 217
lawful interception, 6
Mail, 232
  address, 237
  adress internationalized, 239
  INBOX, 233
  MTA *(Message Transfer Agent)*, 233, 243
mean solar time, 100
Media Plane, 219

message imprint, 98, 106
MG *(Media Gateway)*, 210, 219
MGC *(Media Gateway Controller)*, 210, 219
MIME:Content-Transfer-Encoding, 259
MIME:Content-Type, 260
MIME:Mime-Version, 259
non-repudiation, 2
nonce, 16
OAuth
    Authorization endpoint, 156
    Authorization server, 155
    Client, 155
    Resource owner, 155
    Resource server, 155
    Token Endpoint, 156
OAuth *(short for open authorization)*, 155
OCSP *(Online Certificate Status Protocol)*, 84
OIDC
    ID Token, 158
    UserInfo endpoint, 158
OIDC *(Open ID Connect)*, 158
one-way function, 13
padding, 16
PEM *(Privacy Enhancement for Internet Electronic Mail)*, 71
PFS *(Perfect Forward Secrecy)*, 176
PKCS#10, 72
PKCS#7, 91
PKI *(Public key Infrastructure)*, 60
POP3
    authentication, 251
    Capabilities, 249
    commands, 250
    over TLS, 235, 251
proof
    of private key ownership, 39, 42
protocol
    CMC *(Certificate Management over CMS)*, 73
    CMP (*Certificate Management Protocol*) , 73
    DNS, 277
    DNSSEC, 279
    DTLS *(Datagram Transport Layer Security)*, 187
    ESMTP *(extended SMTP)*, 240
    H.248, 219
    HTTP *(Hypertext Transfer Protocol)*, 110
    HTTP/1.1, 111
    HTTP/2, 111
    HTTP/3, 114
    IMAP4, 240, 245, 252
    Kerberos, 137
    LDAP *(Lightweight Directory Access Protocol)*, 245
    MIME, 258
    NTP *(Network Time Protocol)*, 101
    Photuris, 174
    POP3, 240, 245, 247
    QUIC, 163, 176
    RTCP *(Real-Time Control Protocol)*, 227
    RTP *(Real-time Transport Protocol)*, 195, 225
    S/MIME, 264
    SCTP *(Stream Control Transmission Protocol)*, 190
    SDP *(Session Description Protocol)*, 195, 222
    SIP *(Session Initiation Protocol)*, 195
    SMTP *(Simple Mail Transfer Protocol)*, 240, 247
    SPNEGO *(Simple And Protected Negotiate)*, 150
    SRTCP *(Secure RTCP)*, 230
    SRTP *(Secure Real-time Transport Protocol)*, 228
    TLS, 175
    TSP *(Timestamp Protocol)*, 103
QUIC
    0-RTT packet, 164, 169
    1-RTT packet, 169
    client, 163
    connection ID, 163, 165, 170
    early data, 164
    endpoint, 163
    frame, 170
    Handshake packets, 169
    Initial packets, 168
    Key Phase bit, 170
    packet, 166
    server, 163
    Spin bit, 169
    stream, 163, 165
RA *(Registration Authorities)*, 68
RBAC *(Role-Based Access Control)*, 160
RFC 822 message, 232
RSA-PKCS#1 v1.5, 27
RSA-PSS, 27
SAML *(Security Assertion Markup Language)*, 151
SAML assertion, 152
SASL *(Simple Authentication and Security*

*Layer)*, 245
SCTP
association, 191
chunk, 192
connection, 190
cookie, 191
multi-homing, 194
peer, 190
stream, 193
signature
based on (EC)DSA, 27
based on RSA, 26
composite, 28
digital, 26
electronic, 26
timestamping, 99
SIP
agent, 195
B2BUA *(Back-to-Back User Agent)*, 196
Gateway, 196
message, 199
Outbound Proxy, 196
peer, 197
Proxy, 196
Redirect Server, 196
Registrar, 196, 200
registration, 200, 216
SBC *(Session Border Controller)*, 210
trunk, 211
UA *(User Agent)*, 196
UAC *(User Agent Client)*, 196
UAS *(User Agent Server)*, 196
SMTP
Authentication, 243
over TLS, 235, 245
SPN *(Service Principal Name)*, 137
SS7 (Signaling System No. 7), 219
TAI *(Temps Atomique International)*, 100
TCXO *(Temperature-Compensated Crystal Oscillator)*, 102
TGS *(Ticket Granting Service)*, 138
Timestamp, 98
TLS
AP *(Alert Protocol)*, 183
CCS *(Change Cipher Specification)*, 184
Handshake Secret, 180
HP *(Handshake Protocol)*, 177
KDF *(Key Derivation Function)*, 180
Post Handshake authentication, 180
PSK *(Pre-shared Secret Key)*, 179
RLP *(Record Layer Protocol)*, 183
transcoding, 225
trusted anchor, 60
TSA *(Timestamping Authority)*, 98
TSU *(Timestamp Unit)*, 103
UPN *(User Principal Name)*, 137
URI
HTTP URI, 118
SIP URI, 198
TEL URI, 198
URI *(Uniform Resource Identifier)*, 118
URN, 199
UT *(Universal Time)*, 100
UTC *(Universal Time Coordinated)*, 101
Visited network, 214
VoLTE *(Voice over LTE)*, 214
VoNG *(Voice over 5G)*, 214
webmail, 234
XO *(Crystal Oscillator)*, 102
XOR, 17