

---

# Network Security

## Chapter 7 Cryptographic Protocols

Prof. Dr.-Ing. Georg Carle

Chair for Computer Networks & Internet  
Wilhelm-Schickard-Institute for Computer Science  
University of Tübingen

<http://net.informatik.uni-tuebingen.de/>  
[carle@informatik.uni-tuebingen.de](mailto:carle@informatik.uni-tuebingen.de)

---

# Chapter 7 Cryptographic Protocols

## Introduction

---

- ❑ Definition:

A *cryptographic protocol* is defined as a series of steps and message exchanges between multiple entities in order to achieve a specific security objective

- ❑ Properties of a protocol (in general):

- ❑ Everyone involved in the protocol must know the protocol and all of the steps to follow in advance
- ❑ Everyone involved in the protocol must agree to follow it
- ❑ The protocol must be unambiguous, that is every step is well defined and there is no chance of misunderstanding
- ❑ The protocol must be complete, i.e. there is a specified action for every possible situation

- ❑ Additional property of a cryptographic protocol:

- ❑ It should not be possible to do or learn more than what is specified in the protocol

## Applications of Cryptographic Protocols

---

- ❑ Key exchange
  - ❑ Authentication
    - ❑ Data origin authentication
    - ❑ Entity authentication
  - ❑ Combined authentication and key exchange
- } treated in this course
- ❑ Secret splitting
  - ❑ Secret sharing
  - ❑ Time-stamping
  - ❑ Key escrow (ensuring that only an authorized entity can recover keys)
  - ❑ Zero-Knowledge proofs (proof of knowledge of an information without revealing the information)
  - ❑ Blind signatures (useful for privacy-preserving time-stamping services)
  - ❑ Secure elections
  - ❑ Electronic money

## Key Exchange

---

- ❑ The Diffie-Hellman protocol introduced in section 2.1.2 is our first example of a cryptographic protocol for key exchange
- ❑ Please note, that it does not realize any authentication:
  - ❑ Neither Alice nor Bob know after a protocol run, with whom they have exchanged a key
  - ❑ As this pure key exchange without authentication can not even guarantee privacy of a communication following the exchange, it has to be combined with authentication
- ❑ However, this separation of key exchange and authentication of the exchange has a big advantage, as it allows to guarantee the property of *perfect forward secrecy (PFS)*:
  - ❑ If a key exchange ensures PFS, then a compromise of one key in the future will not allow to compromise any data that has been protected with other keys exchanged before that compromise.
  - ❑ Example: imagine Alice and Bob both sign the data exchanged to compute *sk* with their private keys. Even the compromise of a private key in the future will not allow to decrypt recorded data that has been protected with *sk*.

## Data Origin Authentication

---

### Definition:

*Data origin authentication* is the security service that enables entities to verify that a message has been originated by a particular entity and that it has not been altered afterwards

A synonym for this service is *data integrity*

- ❑ The relation of data integrity to cryptographic protocols is twofold:
  - ❑ There are cryptographic protocols to ensure data integrity. As a rule they comprise just one protocol step and are, therefore, not very “exciting”:
    - Example 1: assume, that everybody knows Alice’s public RSA key and can be sure to know really Alice’s key, then Alice can insure data integrity of her messages by encrypting them with her private key.
    - Example 2: Alice can as well compute an MDC over her message and append the MDC encrypted with her private key to the message
  - ❑ Data integrity of messages exchanged is often an important property in cryptographic protocols, so data integrity is a building block to cryptographic protocols

## Entity Authentication (1)

---

### Definition:

*Entity authentication* is the security service, that enables communication partners to verify the identity of their peer entities.

- ❑ Entity authentication is the most fundamental security service, as all other security services build upon it
- ❑ In general it can be accomplished by various means:
  - ❑ *Knowledge*: e.g. passwords
  - ❑ *Possession*: e.g. physical keys or cards
  - ❑ *Immutable characteristic*: e.g. biometric properties like fingerprint, etc.
  - ❑ *Location*: evidence is presented that an entity is at a specific place (example: people check rarely the authenticity of agents in a bank)
  - ❑ *Delegation of authenticity*: the verifying entity accepts, that somebody who is trusted has already established authentication
- ❑ In communication networks, direct verification of the above means is difficult or insecure which motivates the need for cryptographic protocols

## Entity Authentication (2)

---

- ❑ The main reason, why entity authentication is more than an exchange of (data-origin-) authentic messages is *timeliness*:
  - ❑ Even if Bob receives authentic messages from Alice during a communication, he can not be sure, if:
    - Alice is actually participating in the communication *in this specific moment*, or if
    - Eve is *replaying* old messages from Alice
  - ❑ This is of specific significance, when authentication is only performed at connection-setup time:
    - Example: transmission of a (possibly encrypted) PIN when logging in
  - ❑ Two principle means to ensure timeliness in cryptographic protocols:
    - *Timestamps* (require more or less synchronized clocks)
    - *Random numbers* (challenge-response exchanges)
- ❑ Most authentication protocols do also establish a secret session key for securing the session following the authentication exchange

## Entity Authentication (3)

- Two main categories of protocols for entity authentication:
  - *Arbitrated authentication*: an arbiter, also called *trusted third party (TTP)* is directly involved in every authentication exchange
    - Advantages:
      - This allows two parties A and B to authenticate to each other without knowing any pre-established secret
      - Even if A and B do not know each other, symmetric cryptography can be used
    - Drawbacks:
      - The TTP can become a bottleneck, availability of TTP is critical
      - The TTP can monitor all authentication activity
  - *Direct authentication*: A and B directly authenticate to each other
    - Advantages: no online participation of a third party is required and no possible performance bottleneck is introduced
    - Drawbacks: requires asymmetric cryptography or pre-established secret keys

## Some Notation...

### Notation of Cryptographic Protocols (1)

Notation	Meaning
$A$	Name of A, analogous for $B, E, TTP, CA$
$CA_A$	Certification Authority of A (explained later)
$r_A$	Random value chosen by A
$t_A$	Timestamp generated by A
$(m_1, \dots, m_n)$	Concatenation of messages $m_1, \dots, m_n$
$A \rightarrow B: m$	A sends message $m$ to B
$K_{A, B}$	Secret key, only known to A and B

## Some Notation...

### Notation of Cryptographic Protocols (2)

Notation	Meaning
$+K_A$	Public key of $A$
$-K_A$	Private key of $A$
$\{m\}_K$	Message $m$ encrypted with key $K$ , synonym for $E(K, m)$
$H(m)$	MDC over message $m$ , computed with function $H$
$A[m]$	Shorthand notation for $(m, \{H(m)\}_{-K_A})$
$Cert_{-CK_{CA}}(+K_A)$	Certificate of $CA$ for public key $+K_A$ of $A$ , signed with private certification key $-CK_{CA}$ (explained later)
$CA\langle\langle A \rangle\rangle$	Shorthand notation for $Cert_{-CK_{CA}}(+K_A)$

## The Needham-Schroeder Protocol (1)

- Invented in 1978 by Roger Needham and Michael Schroeder [Nee78a]
- The protocol relies on symmetric encryption and makes use of a *trusted third party (TTP)*
- Assume that  $TTP$  shares secret keys  $K_{A,TTP}$  and  $K_{B,TTP}$  with Alice and Bob, respectively:
  - $A$  generates a random number  $r_A$  and sends the following message:
    - 1.)  $A \rightarrow TTP: (A, B, r_A)$
  - $TTP$  generates a session key  $K_{A,B}$  for secure communication between  $A$  and  $B$  and answers to  $A$ :
    - 2.)  $TTP \rightarrow A: \{r_A, B, K_{A,B}, \{K_{A,B}, A\}_{K_{B,TTP}}\}_{K_{A,TTP}}$
  - $A$  decrypts the message and extracts  $K_{A,B}$ . She confirms that  $r_A$  is identical to the number generated by her in the first step, thus she knows the reply is a fresh reply from  $TTP$ . Then she sends to  $B$ :
    - 3.)  $A \rightarrow B: \{K_{A,B}, A\}_{K_{B,TTP}}$

## The Needham-Schroeder Protocol (2)

- Needham-Schroeder protocol definition (continued):
  - Bob decrypts the message and obtains  $K_{A,B}$ . He then generates a random number  $r_B$  and answers to Alice:
    - 4.)  $B \rightarrow A: \{r_B\}_{K_{A,B}}$
  - Alice decrypts the message, computes  $r_B - 1$  and answers with:
    - 5.)  $A \rightarrow B: \{r_B - 1\}_{K_{A,B}}$
  - Bob decrypts the message and verifies that it contains  $r_B - 1$
- Discussion:
  - The exchange of  $r_B$  and  $r_B - 1$  is supposed to ensure that an attacker, trying to impersonate Bob, can not perform a full protocol run with replayed messages
  - However, as old session keys  $K_{A,B}$  remain valid, an attacker, Eve, who manages to get to know a session key  $K_{A,B}$  can later use this to impersonate as Alice:
    - 1.)  $E \rightarrow B: \{K_{A,B}, A\}_{K_{B,TTP}}$
    - 2.)  $B \rightarrow A: \{r_B\}_{K_{A,B}}$  Eve has to intercept this message
    - 3.)  $E \rightarrow B: \{r_B - 1\}_{K_{A,B}}$So, even though she doesn't know  $K_{A,TTP}$  nor  $K_{B,TTP}$  Eve impersonates as Alice!

## The Otway-Rees Protocol (1)

- The security problem described above as well as some others were addressed by Needham and Schroeder. Their solution [Nee87a] is essentially the same like the one proposed by Otway and Rees in the same journal [Otw87a]:
  - Alice generates a message containing an index number  $i_A$ , her name  $A$ , Bobs name  $B$ , and the same information plus an additional random number  $r_A$  encrypted with the key  $K_{A,TTP}$  she shares with TTP, and sends this message to Bob:
    - 1.)  $A \rightarrow B: (i_A, A, B, \{r_A, i_A, A, B\}_{K_{A,TTP}})$
  - Bob generates a random number  $r_B$ , encrypts it together with  $i_A, A$ , and  $B$  using the key  $K_{B,TTP}$  he shares with TTP and sends the message to TTP:
    - 2.)  $B \rightarrow TTP: (i_A, A, B, \{r_A, i_A, A, B\}_{K_{A,TTP}}, \{r_B, i_A, A, B\}_{K_{B,TTP}})$
  - TTP generates a new session key  $K_{A,B}$  and creates two encrypted messages, one for Alice and one for Bob, and sends them to Bob:
    - 3.)  $TTP \rightarrow B: (i_A, \{r_A, K_{A,B}\}_{K_{A,TTP}}, \{r_B, K_{A,B}\}_{K_{B,TTP}})$

## The Otway-Rees Protocol (2)

---

- ❑ Otway-Rees protocol definition (continued):
  - ❑ Bob decrypts his part of the message, verifies  $r_B$  and sends Alice's part of the message to her:
    - 4.)  $B \rightarrow A: (i_A, \{r_A, K_{A,B}\}_{K_{A,TTP}})$
  - ❑ Alice decrypts the message and checks if  $i_A$  and  $r_A$  have not changed during the exchange. If not, she can be sure that TTP has send her a fresh session key  $K_{A,B}$  for communication with Bob. If she now uses this key in an encrypted communication with Bob, she can be sure of his authenticity.
- ❑ Discussion:
  - ❑ The index number  $i_A$  prevents against replay attacks. However, this requires that TTP checks if  $i_A$  is bigger than the last  $i_A$  he received from Alice.
  - ❑ As TTP will just generate the two messages if both parts of the message he receives contain the same index number  $i_A$  and names  $A, B$ , Alice and Bob can be sure that both of them have authenticated to TTP during the protocol run.

## Kerberos (1)

---

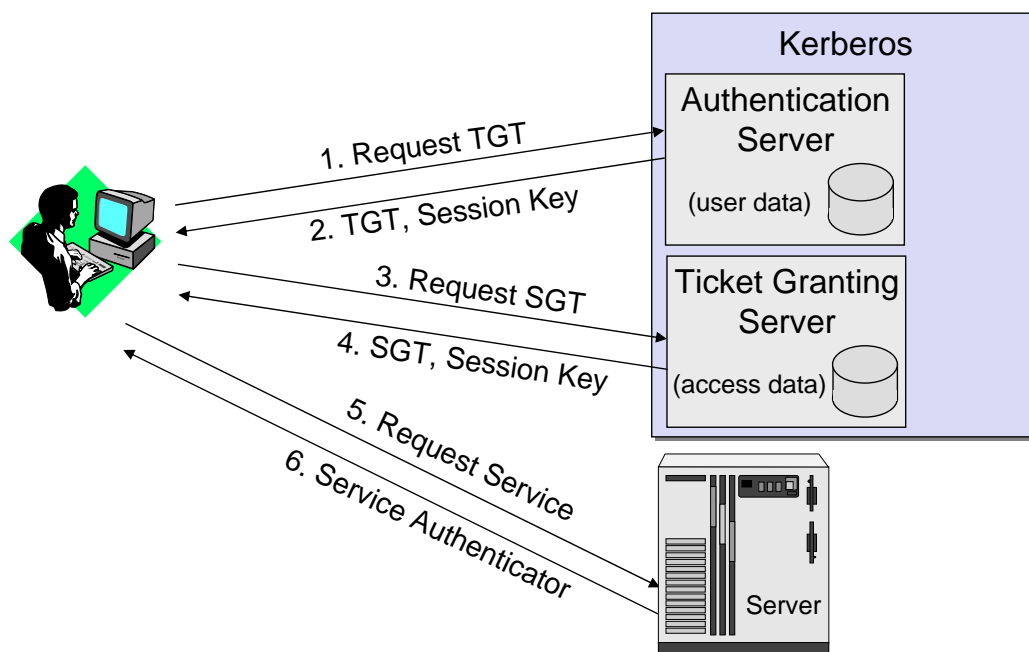
- ❑ Kerberos is an authentication and access control service for workstation clusters that was designed at the MIT during the late 1980s
- ❑ Design goals:
  - ❑ *Security*: eavesdroppers or active attackers should not be able to obtain the necessary information to impersonate a user when accessing a service
  - ❑ *Reliability*: as every use of a service requires prior authentication, Kerberos should be highly reliable and available
  - ❑ *Transparency*: the authentication process should be transparent to the user beyond the requirement to enter a password
  - ❑ *Scalability*: the system should be able to support a large number of clients and servers
- ❑ The underlying cryptographic primitive of Kerberos is symmetric encryption (Kerberos V. 4 uses DES, V. 5 allows other algorithms)
- ❑ A good tutorial on the reasoning beyond the Kerberos design is given in [Bry88a], that develops the protocol in a series of fictive dialogues



## Kerberos (2)

- ❑ The basic usage scenario of Kerberos is a user, *Alice*, who wants to access one or more different services, that are provided by different servers  $S_1, S_2, \dots$  connected over an insecure network
- ❑ Kerberos deals with the following security aspects of this scenario:
  - ❑ *Authentication*: Alice will authenticate to an *authentication server (AS)* who will provide a temporal permit to demand access for services. This permit is called *ticket-granting ticket ( $Ticket_{TGS}$ )* and is comparable to a temporal passport.
  - ❑ *Access control*: by presenting her  $Ticket_{TGS}$  Alice can demand a *ticket granting server (TGS)* to obtain access for a service provided by a specific server  $S_1$ . The TGS decides if the access will be permitted and answers with a service granting ticket  $Ticket_{S_1}$  for server  $S_1$ .
  - ❑ *Key exchange*: the authentication server provides a session key for communication between Alice and TGS and the TGS provides a session key for communication between Alice and  $S_1$ . The use of these session keys also serves for authentication purposes.

## Kerberos (3)



Accessing a Service with Kerberos Version 4 - Protocol Overview

## Kerberos (4)

- The user logs on his workstation and requests to access a service:
  - The workstation represents him in the Kerberos protocol and sends the first message to the authentication server  $AS$ , containing his name, the name of an appropriate ticket granting server  $TGS$  and a timestamp  $t_A$ :
    - 1.)  $A \rightarrow AS: (A, TGS, t_A)$
- The  $AS$  verifies, that  $A$  may authenticate itself to access services, generates the key  $K_A$  out of  $A$ 's password (which is known to him), extracts the workstation address  $Addr_A$  of the request, creates a ticket granting ticket  $Ticket_{TGS}$  and a session key  $K_{A,TGS}$ , and sends the following message to  $A$ :
  - 2.)  $AS \rightarrow A: \{K_{A,TGS}, TGS, t_{AS}, LifetimeTicket_{TGS}, Ticket_{TGS}\}_{K_A}$   
with  $Ticket_{TGS} = \{K_{A,TGS}, A, Addr_A, TGS, t_{AS}, LifetimeTicket_{TGS}\}_{K_{A,TGS}}$
- Upon receipt of this message, the workstation asks Alice to type in her password, computes the key  $K_A$  from it, and uses this key to decrypt the message. If Alice does not provide her "authentic" password, the extracted values will be "garbage" and the rest of the protocol will fail

## Kerberos (5)

- Alice creates a so-called *authenticator* and sends it together with the ticket-granting ticket and the name of server  $S1$  to  $TGS$ :
  - 3.)  $A \rightarrow TGS: (S1, Ticket_{TGS}, Authenticator_{A,TGS})$   
with  $Authenticator_{A,TGS} = \{A, Addr_A, t'_A\}_{K_{A,TGS}}$
- Upon receipt,  $TGS$  decrypts  $Ticket_{TGS}$ , extracts the key  $K_{A,TGS}$  from it and uses this key to decrypt  $Authenticator_{A,TGS}$ . If the name and address of the authenticator and the ticket are matching and the timestamp  $t'_A$  is still fresh, it checks if  $A$  may access the service  $S1$  and creates the following message:
  - 4.)  $TGS \rightarrow A: \{K_{A,S1}, S1, t_{TGS}, Ticket_{S1}\}_{K_{A,TGS}}$   
with  $Ticket_{S1} = \{K_{A,S1}, A, Addr_A, S1, t_{TGS}, LifetimeTicket_{S1}\}_{K_{A,TGS}}$
- Alice decrypts the message and does now hold a session key for secure communication with  $S1$ . She now sends a message to  $S1$  to show him her ticket and a new authenticator:
  - 5.)  $A \rightarrow S1: (Ticket_{S1}, Authenticator_{A,S1})$   
with  $Authenticator_{A,S1} = \{A, Addr_A, t''_A\}_{K_{A,S1}}$

## Kerberos (6)

---

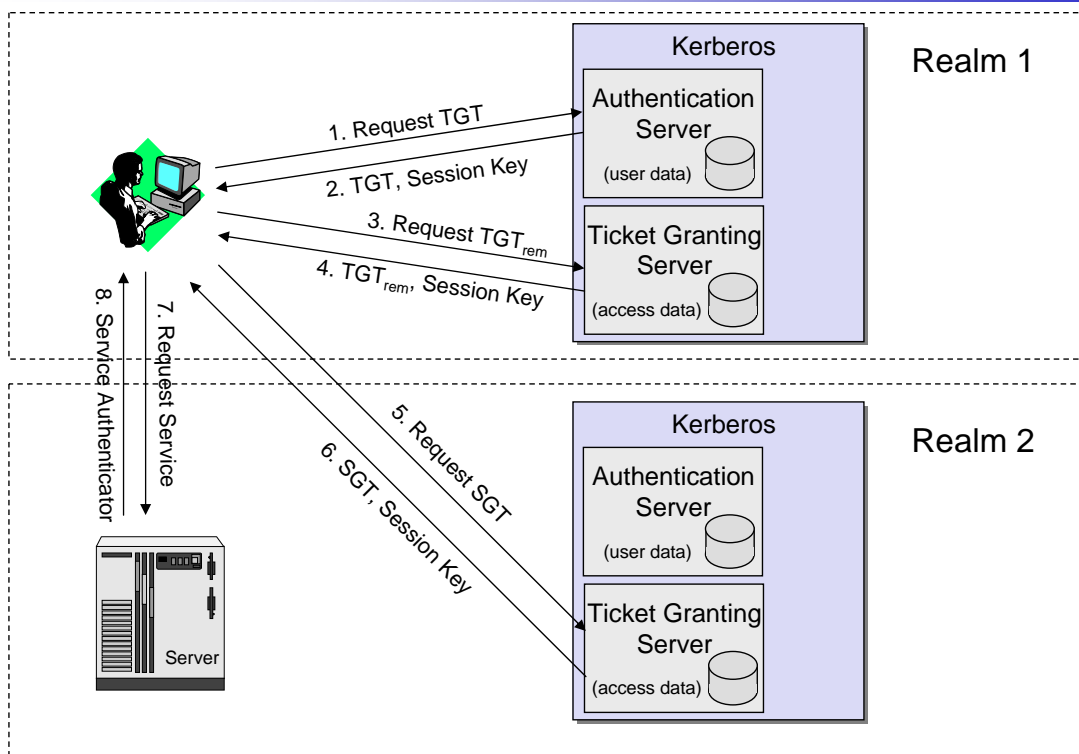
- Upon receipt,  $S1$  decrypts the ticket with the key  $K_{TGS,S1}$  he shares with  $TGS$  and obtains the session key  $K_{A,S1}$  for secure communication with  $A$ . Using this key he checks the authenticator and responds to  $A$ :
  - 6.)  $S1 \rightarrow A: \{t''_A + 1\}_{K_{A,S1}}$
- By decrypting this message and checking the contained value, Alice can verify, that she is really communicating with  $S1$ , as only he (besides  $TGS$ ) knows the key  $K_{TGS,S1}$  to decrypt  $Ticket_{S1}$  which contains the session key  $K_{A,S1}$ , and so only he is able to decrypt  $Authenticator_{A,S1}$  and to answer with  $t''_A + 1$  encrypted with  $K_{A,S1}$
- The protocol described above is the Kerberos Version 4 dialogue.
  - A number of deficiencies have been found in this protocol, so a new Version 5 of the protocol has been defined
  - Due to time constraints, this version will not be discussed in this class
  - For further information on V.5 please refer to [Koh94a, Sta98a]

## Multiple Domain Kerberos (1)

---

- Consider an organization with workstation clusters on two different sites, and imagine that user  $A$  of site 1 wants to use a server of site 2:
  - If both sites do use their own Kerberos servers and user databases (containing passwords) then there are in fact two different *domains*, also called *realms* in Kerberos terminology.
  - In order to avoid that user  $A$  has to be registered in both realms, Kerberos allows to perform an inter-realm authentication.
- Inter-realm authentication requires, that the ticket granting servers of both domains share a secret key  $K_{TGS1,TGS2}$ 
  - The basic idea is, that the TGS of another realm is viewed as a normal server for which the TGS of the local realm can hand out a ticket.
  - After obtaining the ticket for the remote realm, Alice requests a service granting ticket from the remote TGS
  - However, this implies that remote realm has to trust the Kerberos authentication service of the home domain of a “visiting” user!
  - Scalability problem:  $n$  realms require  $n \times (n - 1) / 2$  secret keys!

## Multiple Domain Kerberos (2)



## Multiple Domain Kerberos (3)

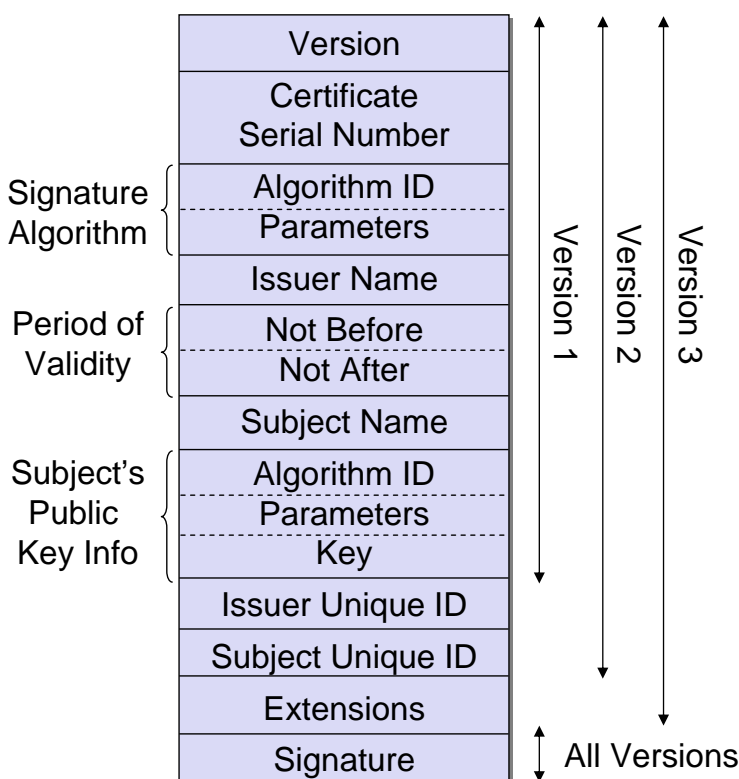
### □ Messages exchanged during a multiple domain protocol run:

- 1.)  $A \rightarrow AS1: (A, TGS1, t_A)$
- 2.)  $AS1 \rightarrow A: \{K_{A,TGS1}, TGS1, t_{AS}, LifetimeTicket_{TGS1}, Ticket_{TGS1}\}_{K_A}$   
with  $Ticket_{TGS1} = \{K_{A,TGS1}, A, Addr_A, TGS1, t_{AS}, LifetimeTicket_{TGS1}\}_{K_{A,TGS1}}$
- 3.)  $A \rightarrow TGS1: (TGS2, Ticket_{TGS1}, Authenticator_{A,TGS1})$   
with  $Authenticator_{A,TGS1} = \{A, Addr_A, t'_A\}_{K_{A,TGS1}}$
- 4.)  $TGS1 \rightarrow A: \{K_{A,TGS2}, TGS2, t_{TGS1}, Ticket_{TGS2}\}_{K_{A,TGS1}}$   
with  $Ticket_{TGS2} = \{K_{A,TGS2}, A, Addr_A, TGS2, t_{TGS1}, LifetimeTicket_{TGS2}\}_{K_{TGS1,TGS2}}$
- 5.)  $A \rightarrow TGS2: (S2, Ticket_{TGS2}, Authenticator_{A,TGS2})$   
with  $Authenticator_{A,TGS2} = \{A, Addr_A, t''_A\}_{K_{A,TGS2}}$
- 6.)  $TGS2 \rightarrow A: \{K_{A,S2}, S2, t_{TGS2}, Ticket_{S2}\}_{K_{A,TGS2}}$   
with  $Ticket_{S2} = \{K_{A,S2}, A, Addr_A, S2, t_{TGS2}, LifetimeTicket_{S2}\}_{K_{TGS2,S1}}$
- 7.)  $A \rightarrow S2: (Ticket_{S2}, Authenticator_{A,S2})$   
with  $Authenticator_{A,S2} = \{A, Addr_A, t'''_A\}_{K_{A,S2}}$
- 8.)  $S1 \rightarrow A: \{t'''_A + 1\}_{K_{A,S1}}$

## X.509 – Introduction

- ❑ X.509 is an international recommendation of ITU-T and is part of the X.500-series defining directory services:
  - ❑ The first version of X.509 was standardized in 1988
  - ❑ A second version standardized 1993 resolved some security concerns
  - ❑ A third version was drafted in 1995
- ❑ X.509 defines a framework for provision of authentication services, comprising:
  - ❑ Certification of public keys and certificate handling:
    - Certificate format
    - Certificate hierarchy
    - Certificate revocation lists
  - ❑ Three different dialogues for direct authentication:
    - One-way authentication, requires synchronized clocks
    - Two-way mutual authentication, still requires synchronized clocks
    - Three-way mutual authentication entirely based on random numbers

## X.509 – Public Key Certificates (1)



- ❑ A *public key certificate* is some sort of passport, certifying that a public key belongs to a specific name
- ❑ Certificates are issued by *certification authorities (CA)*
- ❑ If all users know for sure the public key of the CA, every user can check every certificate issued by this CA
- ❑ Certificates can avoid online-participation of a TTP
- ❑ The security of the private key of the CA is crucial to the security of all users!

## X.509 – Public Key Certificates (2)

---

- Notation of a certificate binding a public key  $+K_A$  to user  $A$  issued by certification authority  $CA$  using its private key  $-CK_{CA}$ :
  - $Cert_{-CK_{CA}}(+K_A) = CA[V, SN, AI, CA, T_{CA}, A, +K_A]$ 
    - with:  $V$  = version number
    - $SN$  = serial number
    - $AI$  = algorithm identifier of signature algorithm used
    - $CA$  = name of certification authority
    - $T_{CA}$  = period of validity of this certificate
    - $A$  = name to which the public key in this certificate is bound
    - $+K_A$  = public to be bound to a name
  - The shorthand notation  $CA[m]$  stands for  $(m, \{H(m)\}_{-CK_{CA}})$
  - Another shorthand notation for  $Cert_{-CK_{CA}}(+K_A)$  is  $CA\langle\langle A \rangle\rangle$

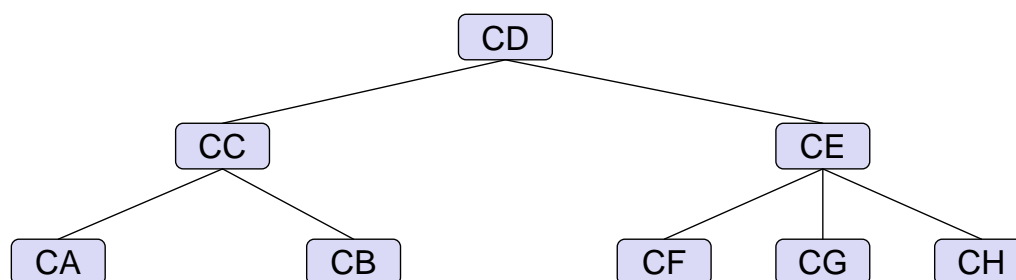
## X.509 – Certificate Chains & Certificate Hierarchy (1)

---

- Consider now two users Alice and Bob, living in different countries, who want to communicate securely:
  - Chances are quite high, that their public keys are certified by different CAs
  - Let's call Alice's certification authority  $CA$  and Bob's  $CB$
  - If Alice does not trust or even know  $CB$  then Bob's certificate  $CB\langle\langle B \rangle\rangle$  is useless to her, the same applies in the other direction
- A solution to this problem is constructing *certificate chains*:
  - Imagine for a moment that  $CA$  and  $CB$  know and trust each other
    - A real world example for this concept is the mutual trust between countries considering their passport issuing authorities
  - If  $CA$  certifies  $CB$ 's public key with a certificate  $CA\langle\langle CB \rangle\rangle$  and  $CB$  certifies  $CA$ 's public key with a certificate  $CB\langle\langle CA \rangle\rangle$ , then  $A$  and  $B$  can check their certificates by checking a certificate chain:
    - Upon being presented  $CB\langle\langle B \rangle\rangle$  Alice tries to look up if there is a certificate  $CA\langle\langle CB \rangle\rangle$
    - She then checks the chain:  $CA\langle\langle CB \rangle\rangle, CB\langle\langle B \rangle\rangle$

## X.509 – Certificate Chains & Certification Hierarchy (2)

- Certificate chains need not to be limited to a length of two certificates:
  - CA<<CC>>, CC<<CD>>, CD<<CE>>, CE<<CG>>, CG<<G>> would permit Alice to check the certificate of user G issued by CG even if she just knows and trusts her own certification authority CA
  - In fact, A's trust in the key  $+K_G$  is established by a *chain of trust* between certification authorities
  - However, if Alice is presented CG<<G>>, it is not obvious which certificates she needs for checking it
- X.509 therefore suggests that authorities are arranged in a *certification hierarchy*, so that navigation is straightforward:



## X.509 – Certificate Revocation

- Consider now that the private key of Alice is compromised, e.g. because Eve broke into her computer read her private key from a file and cracked the password she used to protect the private key:
    - If Alice detects the compromise of her private key, she would definitely like to ask for *revocation* of the corresponding public key certificate
    - If the certificate is not revoked, then Eve could continue to impersonate Alice up to the end of the certificate's validity period
  - An even worse situation occurs, when the private key of a certification authority is compromised:
    - This implies, that all certificates signed with this key have to be revoked!
  - Certificate revocation is realized by maintaining *certificate revocation lists (CRL)*:
    - CRLs are stored in the X.500 directory
    - When checking a certificate, it has also to be checked that the certificate has not yet been revoked (search for the certificate in the CRL)
- Certificate revocation is a relatively slow and expensive operation

## X.509 – Authentication Protocols (1)

---

- One-way authentication:
  - If only Alice wants to authenticate herself to Bob she sends the following message to Bob:
    - 1.)  $(A[t_A, r_A, B, \text{sgnData}_A, \{K_{A,B}\}_{+K_B}], CA\langle\langle A \rangle\rangle)$   
with  $\text{sgnData}_A$  representing optional data to be signed by A,  
 $\{K_{A,B}\}_{+K_B}$  being an optional session key encrypted with Bob's public key,  
and  $CA\langle\langle A \rangle\rangle$  being optional as well
  - Upon reception of this message, Bob verifies with  $+K_{CA}$  the contained certificate, extracts Alice's public key, checks Alice's signature of the message and the timeliness of the message ( $t_A$ ), and optionally decrypts the contained session key  $K_{A,B}$  Alice has proposed
- Two-way authentication:
  - If mutual authentication is desired, then Bob creates a similar message:
    - 2.)  $(B[t_B, r_B, A, r_A, \text{sgnData}_B, \{K_{B,A}\}_{+K_A}], CA\langle\langle B \rangle\rangle)$   
the contained timestamp  $t_B$  is not really required, as Alice can check if the signed message contains the random number  $r_A$

## X.509 – Authentication Protocols (2)

---

- Three-way authentication:
  - If Alice and Bob are not sure if they have synchronous clocks, Alice sends the following message to Bob:
    - 3.)  $A[r_B]$
  - So, the timeliness of Alice's participation in the authentication dialogue is proven by signing the "fresh" random number  $r_B$
- Note on the signature algorithm:
  - As obvious from the use of certificates, X.509 suggests to sign the authentication messages using asymmetric cryptography
  - However, the authentication protocol itself can also be deployed using symmetric cryptography:
    - In this case, A and B need to have agreed upon a secret *authentication key*  $AK_{A,B}$  prior to any protocol run, and
    - the messages are signed by appending a MAC computed with that key.



## Formal Validation of Cryptographic Protocols (1)

---

- ❑ As we have seen from the the Needham-Schroeder protocol, the security of a cryptographic protocol is not obvious to assess:
  - ❑ There are many more examples of *protocol flaws* in cryptographic protocols, which sometimes were discovered not until years after the publication of the protocol
    - An early version of the X.509 standard contained a flaw which was similar to the flaw in the Needham-Schroeder protocol
  - ❑ This motivates the need for *formal methods* for analyzing the properties of cryptographic protocols
- ❑ Categories of formal validation methods for cryptographic protocols:
  - ❑ *General approaches for analysis of specific protocol properties:*
    - Examples: finite-state-machine based approaches, first-order predicate calculus, general purpose specification languages
    - Main Drawback: security differs significantly from correctness as for the later one does not need to assume malicious manipulation

## Formal Validation of Cryptographic Protocols (2)

---

- ❑ Categories of formal validation methods for cryptographic protocols:
  - ❑ *Expert system based approaches:*
    - Knowledge of human experts is formalized into deductive rules that can be used by a protocol designer to investigate different scenarios
    - Main drawback: not well suited to find flaws in cryptographic protocols that are based on unknown attacking techniques
  - ❑ *Algebraic approaches:*
    - Cryptographic protocols are specified as algebraic systems
    - Analysis is conducted by examining algebraic term-rewriting properties of the model and inspecting if the model can attain certain desirable or undesirable states
  - ❑ *Specific logic based approaches:*
    - Approaches of this class define a set of predicates and a mapping of messages exchanged during a protocol run into to a set of formula
    - A generic set of rules allows then to analyze the *knowledge* and *belief* that is obtained by the peer entities of a cryptographic protocol during a protocol run (most successful approach: GNY logics [GNY90a])

## Additional References

---

- [Bry88a] R. Bryant. *Designing an Authentication System: A Dialogue in Four Scenes*. Project Athena, Massachusetts Institute of Technology, Cambridge, USA, 1988.
- [GNY90a] L. Gong, R. Needham, R. Yahalom. *Reasoning About Belief in Cryptographic Protocols*. Proceedings of the IEEE Symposium on Security and Privacy, pp. 234 - 248, Oakland, California, USA, 1990.
- [Koh94a] J. Kohl, C. Neuman, T. T'so, *The Evolution of the Kerberos Authentication System*. In Distributed Open Systems, pages 78-94. IEEE Computer Society Press, 1994.
- [Nee78a] R. Needham, M. Schroeder. *Using Encryption for Authentication in Large Networks of Computers*. Communications of the ACM, Vol. 21, No. 12, 1978.
- [Nee87a] R. Needham, M. Schroeder. *Authentication Revisited*. Operating Systems Review, Vol. 21, No. 1, 1987.
- [Otw87a] D. Otway, O. Rees. *Efficient and Timely Mutual Authentication*. Operating Systems Review, Vol. 21, No. 1, 1987.