Chair for Network Architectures and Services
Institute of Informatics
TU München – Prof. Carle

Network Security

Chapter 2 Cryptography

2.6 Cryptographic Protocols for Encryption, Authentication and Key Establishment

Acknowledgments

This course is based to a significant extend on slides provided by Günter Schäfer, author of the **book "Netzsicherheit - Algorithmische Grundlagen und Protokolle"**, available in German from **dpunkt Verlag**. The English version of the book is entitled “Security in Fixed and Wireless Networks: An Introduction to Securing Data Communications” and is published by Wiley is also available. We gratefully acknowledge his support.

The slides by Günter Schäfer have been partially reworked by Heiko Niedermayer, Ali Fessi, Ralph Holz and Georg Carle.

Overview

- Authentication and Key Establishment Protocols
  - Introduction
  - Key Distribution Centers (KDC)
  - Public Key Infrastructures (PKI)
  - Building Blocks of key exchange protocols

Problem Statement

- Goal
  - Run a key exchange protocol such that at the end of the protocol:
    - Alice and Bob have agreed on a shared „session key“ for a secure channel
    - Alice and Bob have agreed on the cryptographic algorithms to be used for the secure channel
    - Alice (Bob) must be able to verify that Bob (Alice) knows K and that he (she) is “alive”
    - Alice and Bob must know that K is newly generated

Entity Authentication or Key Establishment? (1)

- Many authentication protocols – as a side effect of the authentication exchange - do establish a secret session key for securing the session (to be used only for the current session).
- Some opinions about the relationship between authentication and key establishment:
  - „It is accepted that these topics should be considered jointly rather separately“ [Diff92]
  - „… authentication is rarely useful in the absence of an associated key distribution“ [Bell95]
  - „In our view there are situations when entity authentication by itself may useful, such as when using a physically secured communication channel.“ [Boyd03]

Entity Authentication or Key Establishment? (2)

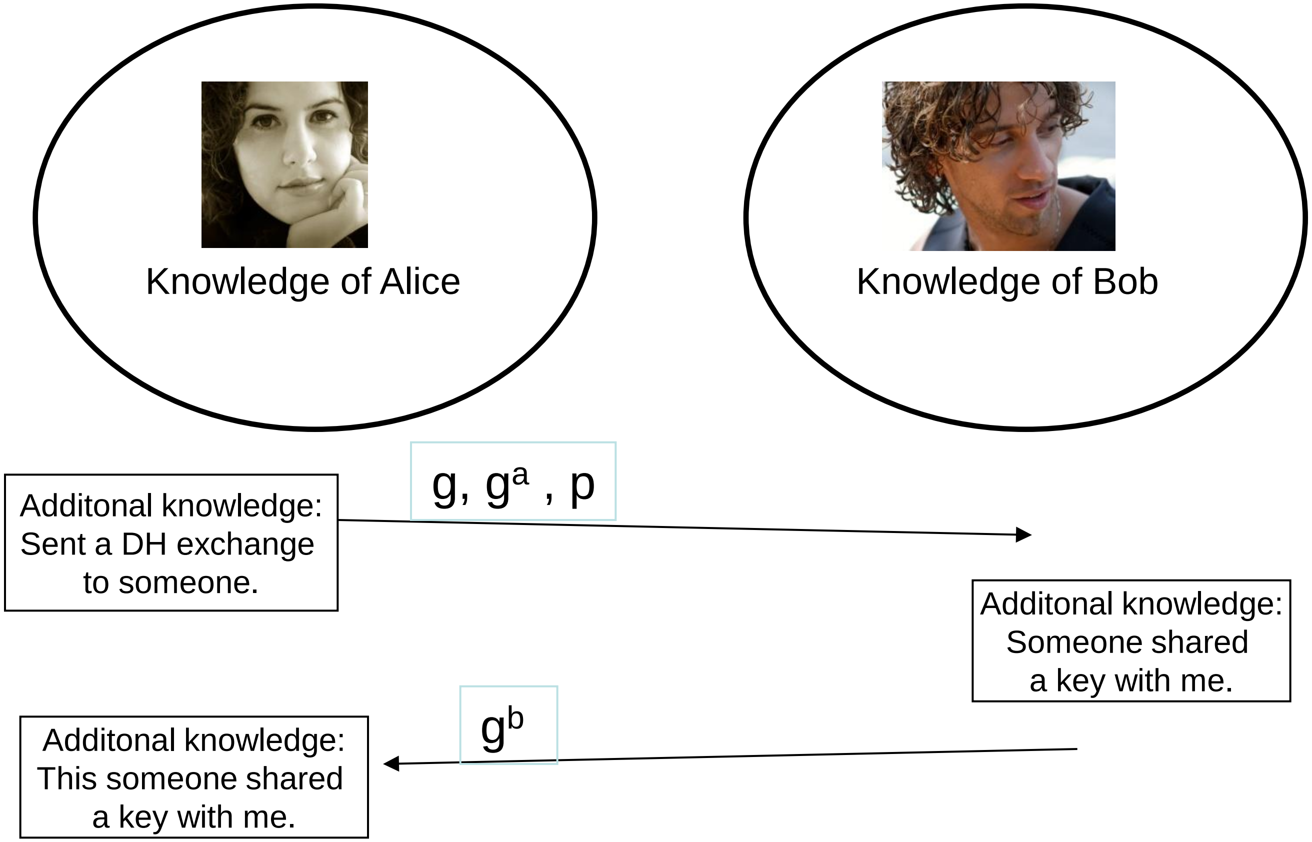
- Example
  - Alice wants to use the online banking service provided by her bank
  - Alice can perform an online banking session from any terminal using a (secure) Internet Browser
  - The Internet browser authenticates the web server based on the certificate (see below) which includes the public key of the web server.
  - Authentication of the web server:
    - as a consequence of this authentication mechanism, a shared session key $K_{A,B}$ is generated, which can be used for this session (it is important that this session key is correctly destroyed when the session is over)
  - Authentication of the client:
    - the web server authenticates Alice based on her PIN number. (As a consequence of the successful authentication of Alice, no additional secret key is established.)
  - This example shows that both cases are common:
    - Entity authentication with key establishment
    - Entity authentication without key establishment
  - The goals of a protocol have to be carefully set up for each application scenario

- Entity authentication
- Entity auth. with key establishment
- Mutual entity authentication
- Mutual entity auth. with key establishment

First Try: Key Establishment with Diffie-Hellman

- Assume Alice and Bob want to establish a secure channel with a shared secret $K_{A,B}$
- The Diffie-Hellman protocol introduced in Chapter 2.2 is our first example of a cryptographic protocol for key exchange. So what’s wrong with it?
- The problem with a “simple DH exchange” is that a man-in-the-middle attack is possible.
- Neither Alice nor Bob know after a protocol run with whom they actually have exchanged a key

Why Diffie-Hellman does not provide authentication.



Knowledge of Alice

Knowledge of Bob

g, g^a, p

Additonal knowledge:
Sent a DH exchange
to someone.

Additonal knowledge:
Someone shared
a key with me.

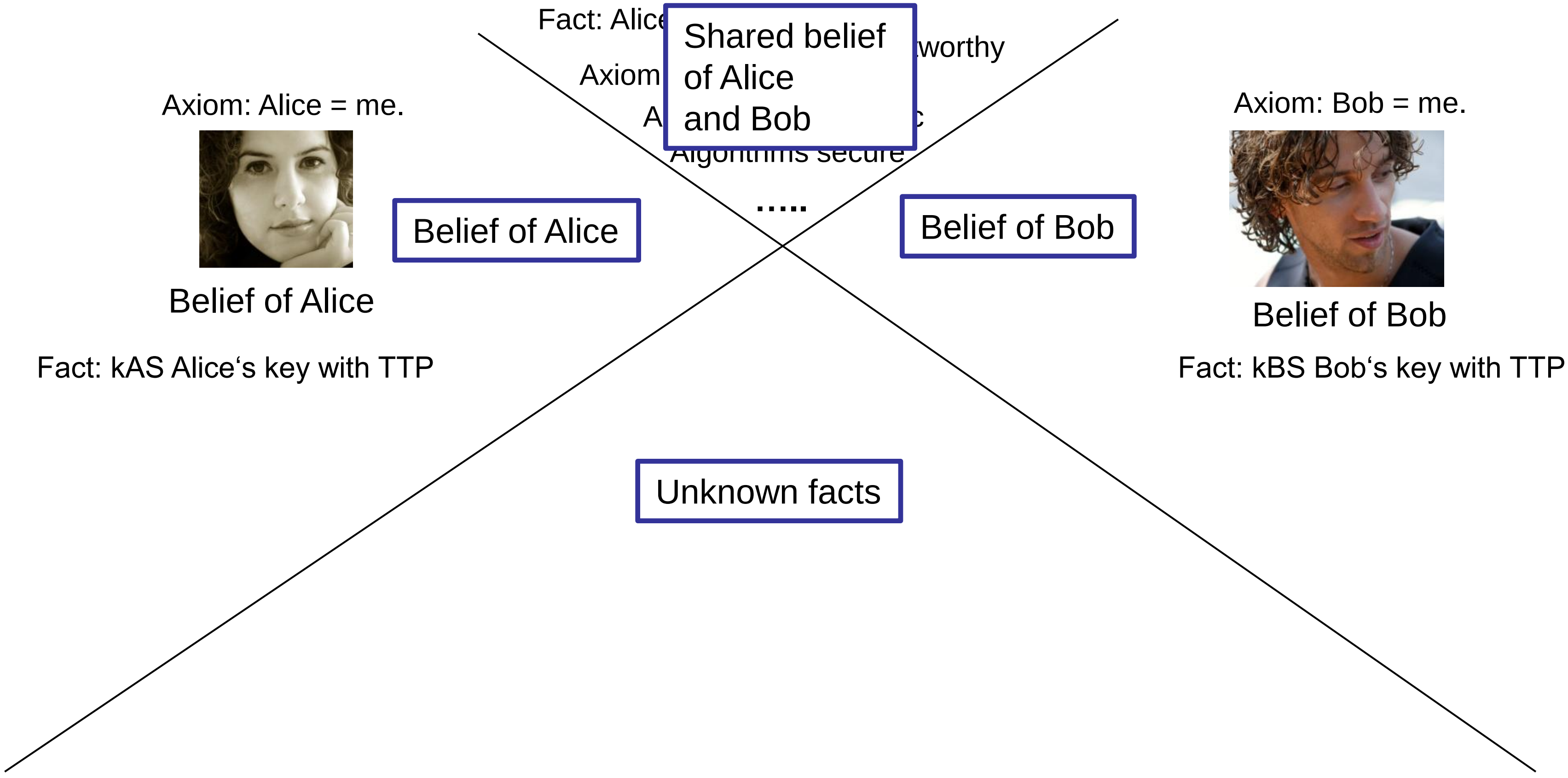
g^b

Additonal knowledge:
This someone shared
a key with me.

- Diffie-Hellman provides a key agreement, but without authentication.
- Without further security measures, neither Alice nor Bob know or proof with whom they shared the key. DH is a key agreement protocol!
- Knowing = it was proven given your knowledge and the protocol

Authentication = Proof in Logic

- Entities belief all facts that can be derived from their *axioms* and the *facts they learned*. (Axiom = basic fact believed without pre-condition)



Fact: Alice

Shared belief of Alice and Bob

worthy

Axiom

A

c

Algorithms secure

.....

Axiom: Alice = me.

Belief of Alice

Belief of Alice

Fact: kAS Alice‘s key with TTP

Belief of Bob

Axiom: Bob = me.

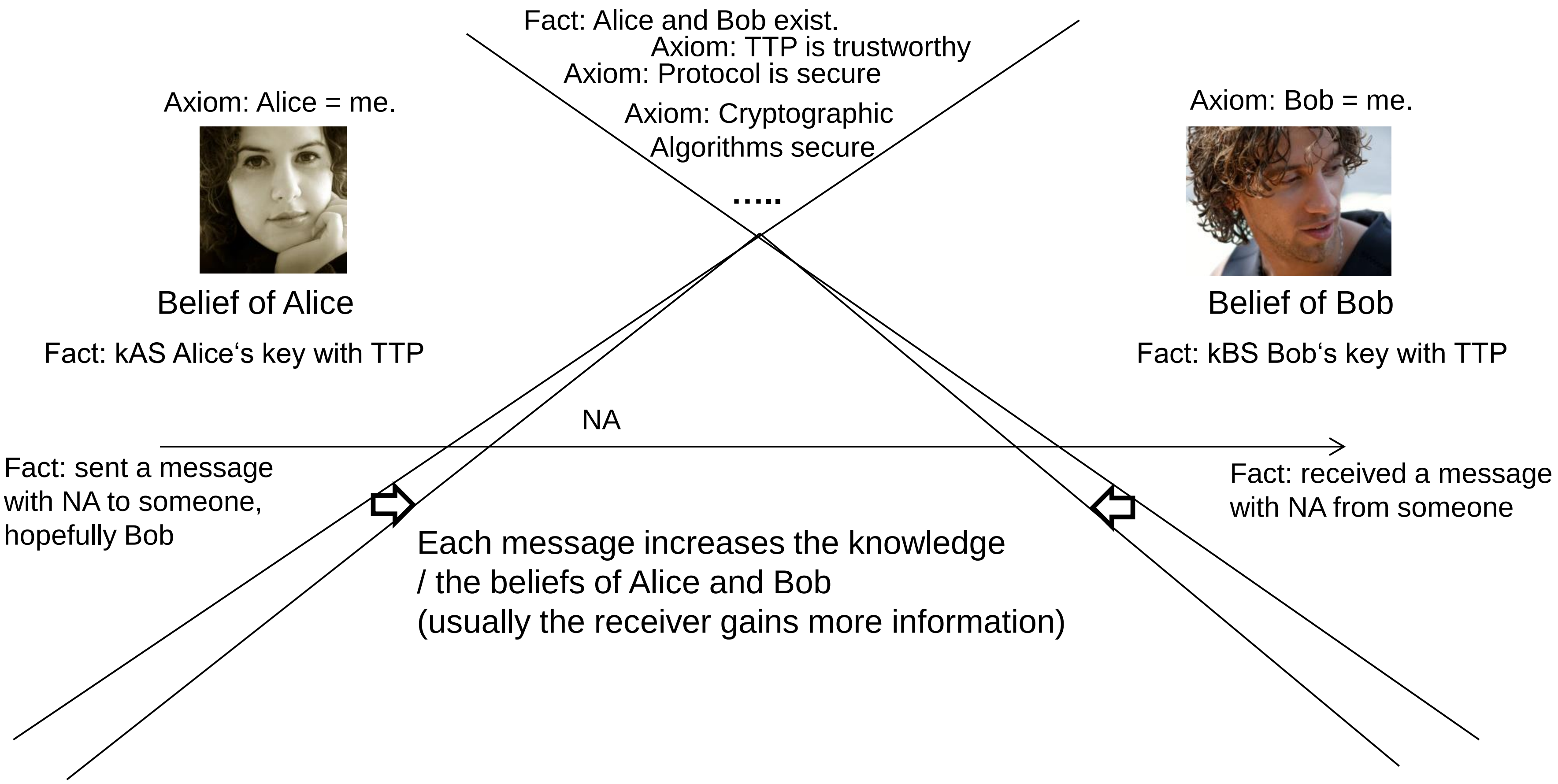
Belief of Bob

Fact: kBS Bob‘s key with TTP

Unknown facts

Authentication = Proof in Logic

- Entities belief all facts that can be derived from their *axioms* and the *facts they learned*. (Axiom = basic fact believed without pre-condition)



Fact: Alice and Bob exist.

Axiom: TTP is trustworthy

Axiom: Protocol is secure

Axiom: Cryptographic Algorithms secure

.....

Axiom: Alice = me.

Belief of Alice

Fact: kAS Alice‘s key with TTP

Axiom: Bob = me.

Belief of Bob

Fact: kBS Bob‘s key with TTP

NA

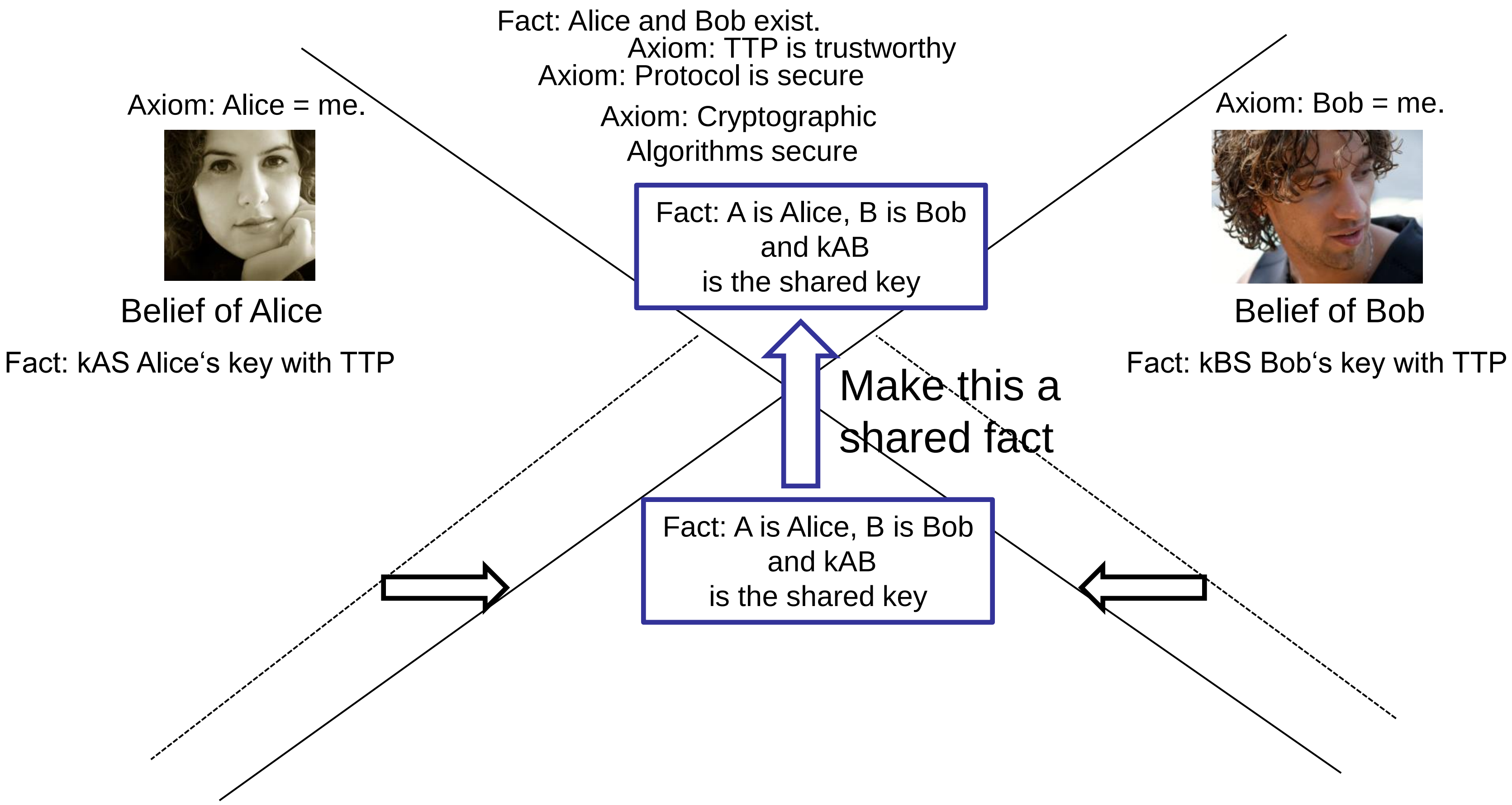
Fact: sent a message with NA to someone, hopefully Bob

Fact: received a message with NA from someone

Each message increases the knowledge / the beliefs of Alice and Bob (usually the receiver gains more information)

Authentication = Proof in Logic

- Goal: Both proove their identity and they establish a shared key and recognize each other with this key (for some time, their session)



Fact: Alice and Bob exist.

Axiom: TTP is trustworthy

Axiom: Protocol is secure

Axiom: Cryptographic Algorithms secure

Axiom: Alice = me.

Belief of Alice

Fact: kAS Alice‘s key with TTP

Axiom: Bob = me.

Belief of Bob

Fact: kBS Bob‘s key with TTP

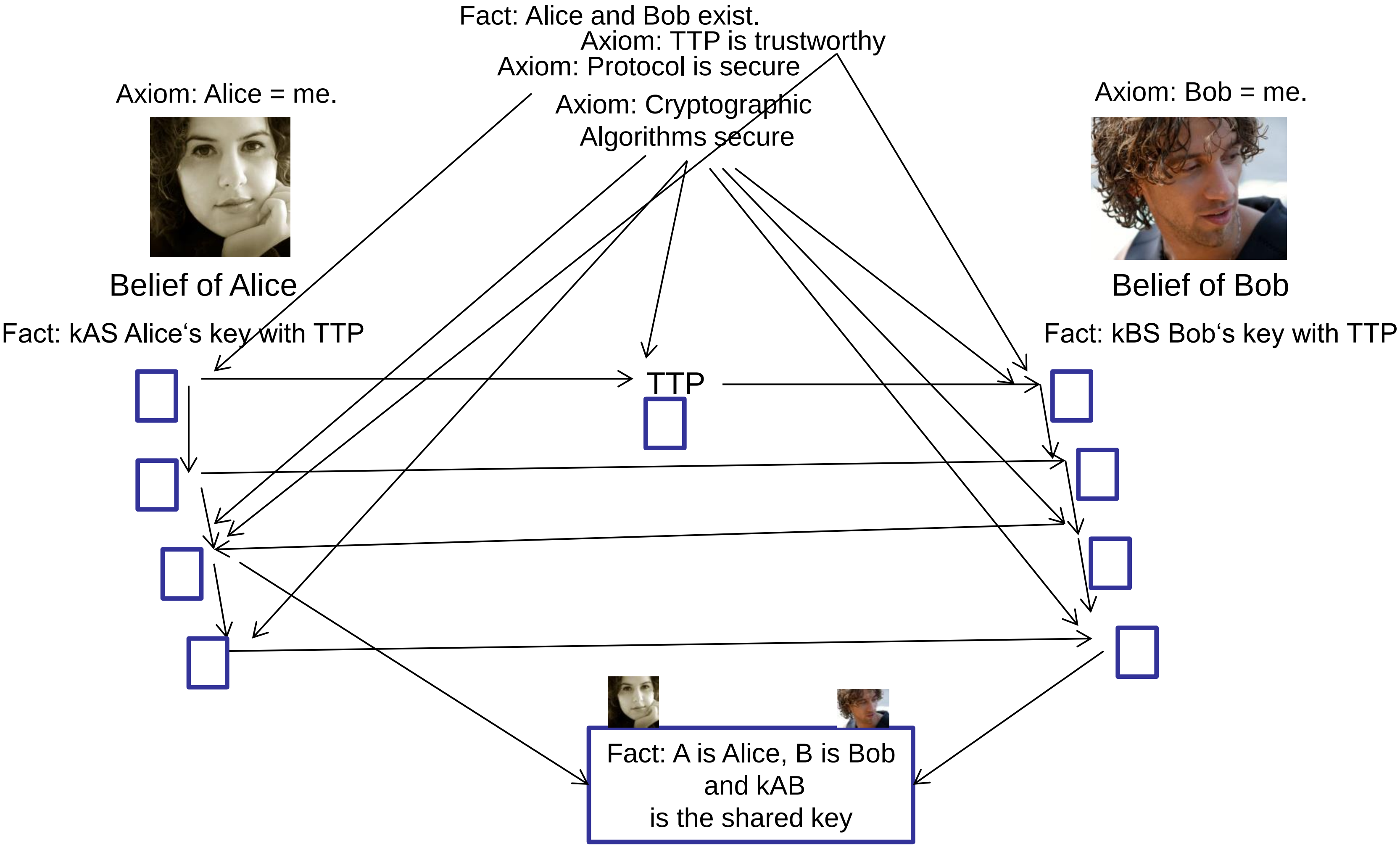
Fact: A is Alice, B is Bob and kAB is the shared key

Make this a shared fact

Fact: A is Alice, B is Bob and kAB is the shared key

Authentication = Proof in Logic

- Both entities locally proof the fact, they need to agree on it in the end.
- Formal definitions for this exist, yet we do not need them for the lecture.



Fact: Alice and Bob exist.

Axiom: TTP is trustworthy

Axiom: Protocol is secure

Axiom: Alice = me.

Axiom: Cryptographic Algorithms secure

Axiom: Bob = me.

Belief of Alice

Belief of Bob

Fact: kAS Alice‘s key with TTP

Fact: kBS Bob‘s key with TTP

TTP

Fact: A is Alice, B is Bob and kAB is the shared key

Second Try: Static Approach

- Static Approach for the negotiation of “session keys” and cryptographic algorithms
  - Keys are manually exchanged. Cryptographic algorithms are agreed on personally
- Pro’s
  - Simple,
  - session keys are automatically authenticated
- Con‘s
  - Manual process is required (either by a direct meeting or by phone call)
  - Does not scale for a large set of hosts

    $\frac{n*(n-1)}{2}$ symmetric keys would be needed for n entities
  - Renewing of keys or cryptographic algorithms require another manual process
  - If the key is compromised, all sessions can be compromised (also previous recorded sessions!)
  - Keys are not changed frequently

Example: Static Approach in GSM/UMTS Networks

- The user mobile phone share a long-term secret key with the home network.
- The secret key is stored in the SIM card that the user received from his provider.

- Note: in GSM/UMTS networks, the scalability issue is not severe
  - A mobile device does not communicate directly with other mobile devices.
  - Communication takes place between the mobile device and the network instead.
  - Only *n* symmetric keys are required (instead of $\frac{n*(n-1)}{2}$ keys).

Trusted Third Parties (TTP)

- Boyd‘s Theorem [Boyd03]
  - *„Assuming the absence of a secure channel, two entities cannot establish an authenticated session without the existence of an entity that can mediate between the two and which both parties trust and have a secure channel with“.*

- A TTP is a special entity which has to be trusted by its users
- A TTP can significantly reduce the key management complexity
- “Trusted” means that it is expected to always behave honestly.
- The TTP is assumed to always respond exactly according to the protocol specification, and, therefore, will never deliberately compromise the security of its clients.

Key Distribution Centers (KDC)

- A KDC is an option for providing authentication and key establishment.
- A KDC is a TTP that shares secrets with all entities (an entity may be a user or a host).
- Alice asks KDC for a secret to (securely) talk to Bob.
- KDC generates a secret $K_{A,B}$
- Example of KDCs:
  - The Kerberos protocol is based on a KDC.
  - In fact, a Kerberos server is often called a KDC.
- Drawbacks:
  - KDC can monitor all authentication and key establishment activities.
  - KDC knows the session key.
  - KDC needs to be online during the authentication and key establishment procedure.
  - KDC is a potential single-point-of-failure/ bottleneck.

Public Key Infrastructures (PKI)

- A Certificate Authority (CA) asserts the correctness of the certificate by signing it with her private key.
- CA is a trusted third party (TTP) that is trusted by all the entities.
- All entities know the public key of the CA.
- Since Alice knows CA's public key, she can verify the signature of Bob's certificate that was generated by CA.
- See later in this chapter for more details on PKIs.

Trusted Third Parties (TTP) – General Remarks

- The TTP is a very powerful entity in this topology. If an attacker manages to compromise TTP, he will be in control of the whole network!

- The TTP may directly be involved in the authentication procedure, which is the case for KDCs.

  ➔ Online TTP

- TTP may not be required for the authentication.
- In case a CA signs the public key of Alice, and Bob knows the public key of the CA, he will be able to verify the validity of Alice's certificate that is signed by CA without talking to CA.

  ➔ Offline TTP (provides more scalability)
  However, Certificate Revocation Lists (CRLs) are still required.

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
r_A	Random value chosen by A
t_A	Timestamp generated by A
$(m_1, ..., m_n)$	Concatenation of messages $m_1, ..., m_n$
$A \rightarrow B: m$	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
$K_{A\text{-}pub}$	Public key of *A*
$K_{A\text{-}priv}$	Private key of *A*
$\{m\}_K$	Message *m* encrypted with key *K*, synonym for $E(K, m)$ *(also integrity protection in case of shared key protocols)*
$H(m)$	Cryptographic hash value over message m, computed with function *H*
$A[m]$	Shorthand notation for $(m, \{H(m)\}_{K_{A\text{-}priv}})$
$Cert_{CK_{CA\text{-}priv}}(K_{A\text{-}pub})$	Certificate of *CA* for public key $K_{A\text{-}pub}$ of *A,* signed with private certification key $CK_{\text{CA-priv}}$ Shorthand notation for $Cert_{CK_{CA\text{-}priv}}(K_{A\text{-}pub})$

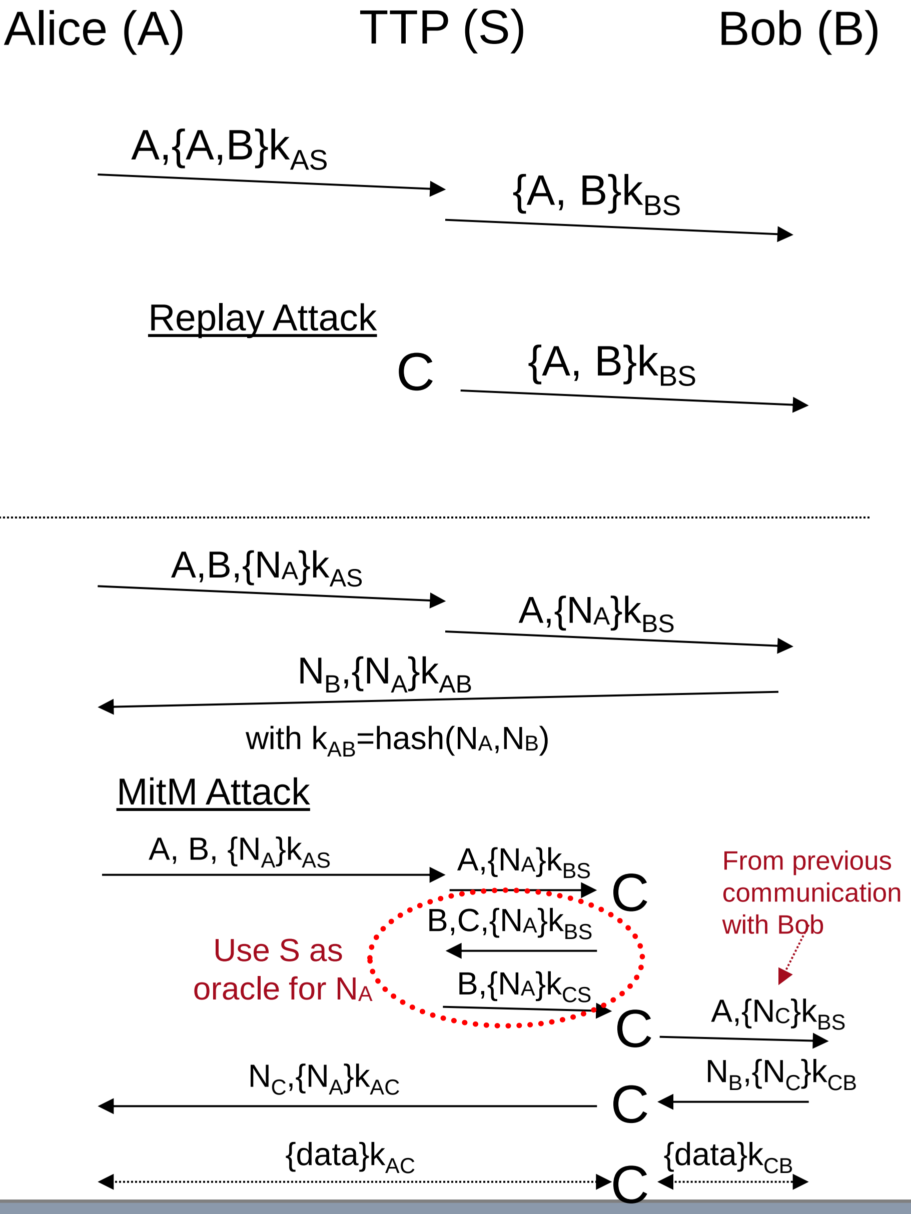
How do attacks against crypto protocols look like?

Replay Attack

- An attacker C can resend the second message.
- Bob cannot decide whether the message is fresh or not.
- Reacting to an old message can result in security compromise!

Man-in-the-Middle attack

- C positions itself between Bob and Alice, and between Bob and the TTP.
- In this example, we assume that C has once talked to Bob and seen the second message containing $\{N_C\}k_{BS}$.

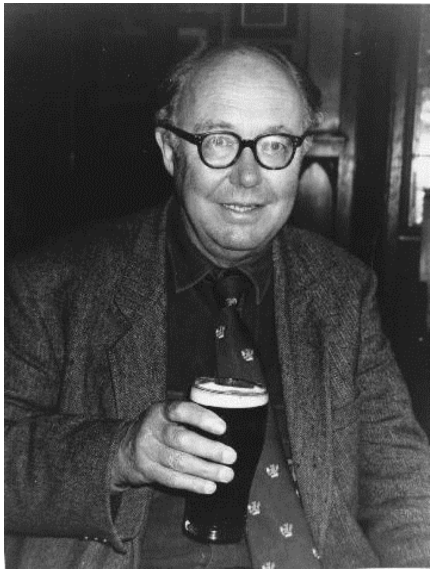


Overview

- ❑ Part I: Introduction
- ❑ Part II: The Secure Channel
- ❑ Part III: Authentication and Key Establishment Protocols
  - ▪ Introduction
  - ▪ Key Distribution Centers (KDC)
    - Needham-Schroeder Protocol
  - ▪ Public Key Infrastructures (PKI)
  - ▪ Building Blocks of a key exchange protocol

The Needham-Schroeder Protocol (1)

- Invented in 1978 by Roger Needham and Michael Schroeder [Nee78]



Roger Needham

Michael Schroeder

The Needham-Schroeder Protocol (2)

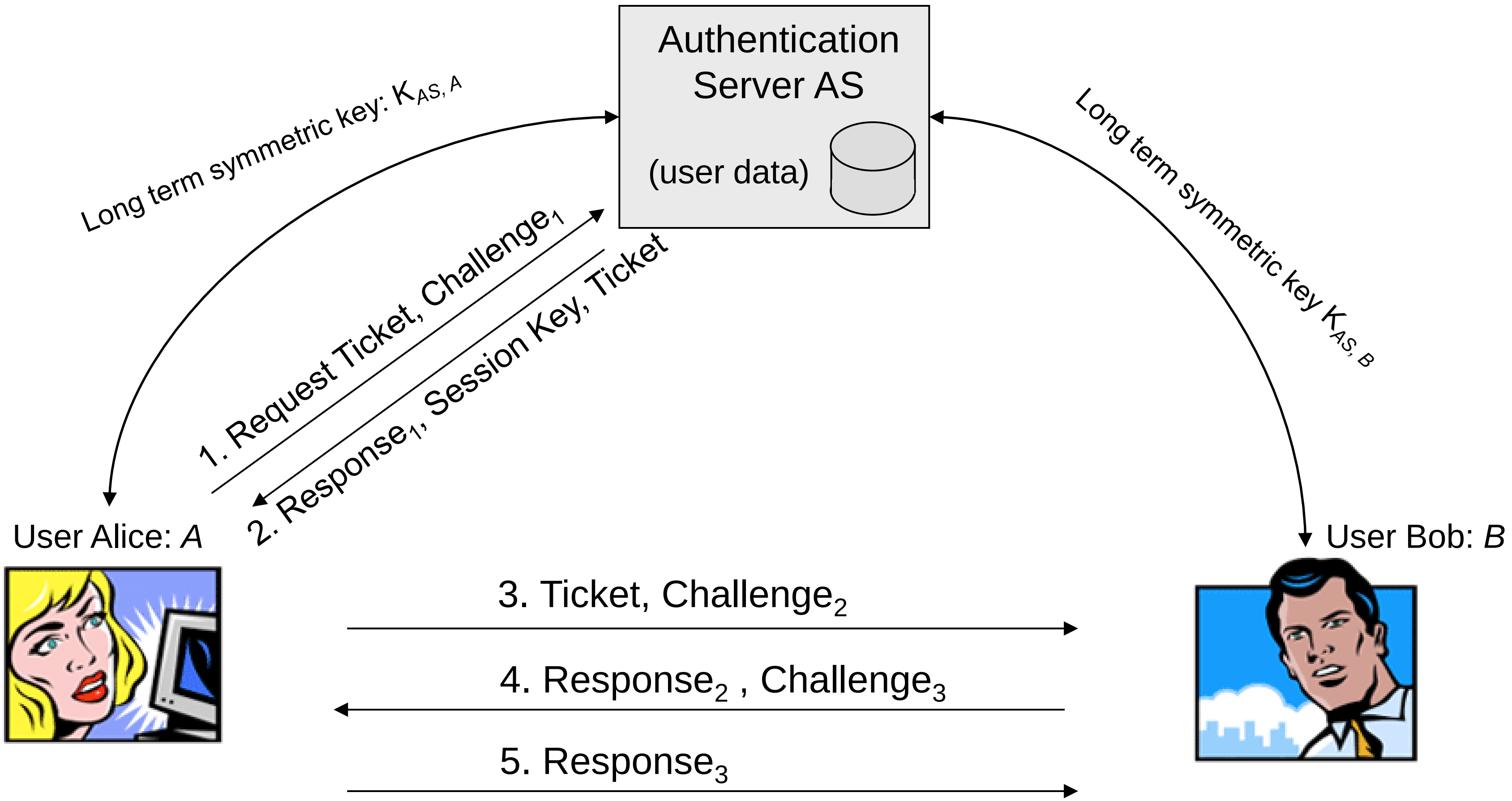
- The Needham-Schroeder Protocol is a protocol for mutual authentication and key establishment
- It aims to establish a session key between two users (or a user and an application server, e.g. email server) over an insecure network
- The protocol has 2 versions:
  - The *Needham Schroeder Symmetric Key Protocol*:
    - based on symmetric encryption
    - Forms the basis for the *Kerberos* protocol
  - The *Needham Schroeder Public Key Protocol*:
    - *Uses public key cryptography*
    - A flaw in this protocol was published by Gavin Lowe [Lowe95] 17 years later!
    - Lowe proposes also a way to fix the flaw in [Lowe95]

Gavin Lowe

Overview

- Part I: Introduction
- Part II: The Secure Channel
- Part III: Authentication and Key Establishment Protocols
  - Introduction
  - Key Distribution Centers (KDC)
    - Needham-Schroeder Protocol
      - Symmetric Version
      - Asymmetric Version
  - Public Key Infrastructures (PKI)
  - Building Blocks of a key exchange protocol

The Needham-Schroeder Symmetric Key Protocol (1)


Authentication Server AS
(user data)
Long term symmetric key: $K_{AS, A}$
Long term symmetric key $K_{AS, B}$
1. Request Ticket, Challenge$_1$
2. Response$_1$, Session Key, Ticket
User Alice: A
User Bob: B
3. Ticket, Challenge$_2$
4. Response$_2$, Challenge$_3$
5. Response$_3$

The Needham Schroeder Symmetric Key Protocol - Overview

The Needham-Schroeder Symmetric Key Protocol (2)

- *AS* shares symmetric keys with all users, in particular with Alice ($K_{AS,A}$) and Bob ($K_{AS,B}$)

1.) $A \rightarrow AS$: (A, B, r_1)

- Alice sends a message to *AS* with her name und Bob’s name, telling the server she wants to communicate with Bob.
- In other words, Alice asks the *KDC* to supply a session key and a “ticket” for secure communication with Bob.
- The freshly generated random number r_1 is used to authenticate *AS* and avoid that a man-in-the-middle is pretending to be *AS.*

2.) $AS \rightarrow A$: $\{r_1, K_{A,B}, B, Ticket_{A,B}\}_{K_{AS,A}}$ where $Ticket_{A,B} = \{K_{A,B}, A\}_{K_{AS,B}}$

- *AS* generates the session key $K_{A,B}$ and sends it to Alice encrypted with $K_{AS,A}$
- *AS* includes r_1 in the encrypted message, so Alice can confirms that r_1 is identical to the number generated by her in the first step, thus she knows the reply is a fresh reply from *AS*.
- Furthermore, *AS* includes a copy of the session key $K_{A,B}$ for Bob included in $Ticket_{A,B}$
- Note here that during this protocol run, *AS* does not communicate directly with Bob
- Since Alice may be requesting keys for several different people, the inclusion of Bob's name tells Alice who she is to share this key with.

The Needham-Schroeder Symmetric Key Protocol (3)

- Needham-Schroeder protocol definition (continued):

3.) $A \rightarrow B$: $(Ticket_{A,B})$

- Alice forwards the ticket to Bob.
- Bob can decrypt the ticket with $K_{AS,B}$ and get the session key $K_{A,B}$.
- Since Alice's name *A* is included in the ticket, Bob knows that this ticket was granted by *AS* for Alice.

4.) $B \rightarrow A$: $\{r_2\}_{K_{A,B}}$

- After decrypting message (3), Bob generates the new random number r_2 and includes it in message (4) which is encrypted with the freshly generated session key $K_{A,B}$.
- However, Bob still also needs to verify that Alice knows the session key $K_{A,B}$ and that she is alive (otherwise, an attacker could send an "old" ticket pretending to be Alice). Therefore, Bob challenges Alice with this new random number r_2

5.) $A \rightarrow B$: $\{r_2 - 1\}_{K_{A,B}}$

- Alice checks if message 4 was encrypted with the freshly generated session key $K_{A,B}$. Since Alice does not know r_2, she has to check the integrity of the message (or detect by similar means that Bob used key $K_{A,B}$).
- After decrypting Bob's message, Alice computes r_2 - 1 and answers with message (5)
- Bob decrypts the message and verifies that it contains r_2 – 1.

The Needham-Schroeder Symmetric Key Protocol (4)

- Needham-Schroeder also proposed a protocol variant where Alice reuses the Ticket from the server. Key $K_{A,B}$ is therefore not new anymore and it cannot be used to authenticate Bob. As a consequence Alice needs to include a challenge in message (3).
- Protocol variant with reuse of ticket and shared key:

1.)+ 2.) Not necessary, Alice reuses the ticket.

3.) $A \rightarrow B: (Ticket_{A,B}, \{r_2\}_{K_{A,B}})$

- Alice sends the ticket again to Bob.
- Bob either still knows the ticket or he can decrypt the ticket again with $K_{AS,B}$ and get the session key $K_{A,B}$.
- Since Alice's name A is included in the ticket, Bob knows that this ticket was granted by AS for Alice.
- As the session key is not fresh anymore, Alice cannot authenticate Bob with $K_{A,B}$. In order to verify that Bob is alive, receiving Alice's messages and still has the correct session key, Alice includes a challenge in message (3) which consists of a nonce random number r_2

4.) $B \rightarrow A: \{r_3, r_2 - 1\}_{K_{A,B}}$

- After decrypting message (3), Bob calculates ($r_2 – 1$) and includes it in message (4) which is encrypted with the freshly generated session key $K_{A,B}$
- However, Bob still also needs to verify that Alice really knows the session key $K_{A,B}$ and that she is alive (otherwise, an attacker could send an "old" ticket pretending to be Alice).
- Therefore, Bob must challenge Alice with a new random number r_3

5.) $A \rightarrow B: \{r_3 - 1\}_{K_{A,B}}$

- After decrypting Bob's message, Alice computes r_3 - 1 and answers with message (5)
- Bob decrypts the message and verifies that it contains r_3 – 1.

The Needham-Schroeder Symmetric Key Protocol (5)

- Discussion:
  - The *Needham-Schroeder Symmetric Key Protocol* can be considered as secure (no known attacks so far) if the session key $K_{A,B}$ can not be "brute-forced" or discovered by an attacker.
  - However, if an attacker, Eve, can manage to get to know a session key $K_{A,B}$, she can later use this to impersonate as Alice by *replaying* the message 3:

    3') $E \rightarrow B: (Ticket_{A,B}, r_2)$

    4') $B \rightarrow A: \{r_3 , r_2 - 1\}_{K_{A,B}}$, Eve has to intercept this message

  Since Eve knows $K_{A,B}$ knows she will be able to decrypt Bob's reply 4') and answers with

    5') $E \rightarrow B: \{r_3 - 1\}_{K_{A,B}}$

    So, if an attacker Eve is able to compromise ***one*** session key $K_{A,B}$, she will be able to impersonate Alice in the future even though she doesn't know $K_{A,TTP}$

  - This problem is solved in the Kerberos protocol with *timestamps*.

The Needham-Schroeder Symmetric Key Protocol (6)

- Note:
  - The term „ticket“ was not used in the original description of the Needham-Schroeder Protocol. [Nee78]
  - However, it is used here to provide an analogy with the Kerberos protocol.
  - In the Kerberos protocol, the ticket includes more data than $K_{A,B}$ and *A.*

Overview

- ❑ Part I: Introduction
- ❑ Part II: The Secure Channel
- ❑ Part III: Authentication and Key Establishment Protocols
  - ▪ Introduction
  - ▪ Key Distribution Centers (KDC)
    - • Needham-Schroeder Protocol
      - – Symmetric Version
      - – Asymmetric Version
  - ▪ Public Key Infrastructures (PKI)
  - ▪ Building Blocks of a key exchange protocol

The Needham-Schroeder Public Key Protocol (1)

- The Needham-Schroeder Public Key Protocol
  - Protocol description
  - Attack published by Gavin Lowe in 1995

The Needham-Schroeder Public Key Protocol (2)

- Assumptions
  - *AS* is a trusted server.
  - *AS* knows the public keys of all users
  - All users know *AS*‘s public key

- Protocol run

1.) $A \rightarrow AS: (A, B)$

- Alice requests Bob’s public key from *AS.*

2.) $AS \rightarrow A: \{ K_{B\text{-}pub} , B \}_{K_{AS\text{-}priv}}$

- AS asserts that Bob’s public key is $K_{B\text{-}pub}$

3.) $A \rightarrow B: \{ r_A , A \}_{K_{B\text{-}pub}}$

- Alice generates a random number r_A and sends it to Bob together with her name, encrypted with Bob’s public key $K_{B\text{-}pub}$

4.) $\mathrm{B} \rightarrow AS: (B, A)$

- Bob requests Alice’s public key from *AS.*

The Needham-Schroeder Public Key Protocol (3)

- Needham-Schroeder public key protocol definition (continued):

5.) $AS \rightarrow B: \{ K_{A\text{-}pub} , A \}_{K_{AS\text{-}priv}}$

- AS asserts that Alice’s public key is $K_{A\text{-}pub}$

6.) $B \rightarrow A: \{ r_A , r_B \}_{K_{A\text{-}pub}}$

- Bob generates a random number r_B and sends it to Alice together with r_A encrypted with $K_{A\text{-}pub}$. Thus, Bob proves to Alice that he was able to decrypt message (3) successfully and therefore proving his identity to Alice. Here in message (6), Bob challenges also, whether she can decrypt the message and extracts r_B .

7.) $A \rightarrow B: \{ r_B \}_{K_{B\text{-}pub}}$

- Alice decrypts message (6) with her private key, extracts r_B and encrypts it with Bob’s public key.
- Upon receipt, Bob can verify that r_B is correct and thus verify that he is talking to Alice.

- At the end of the protocol run, Alice and Bob know each other‘s identities, know both r_A , r_B but r_A , r_B are not known to eavesdroppers. Therefore, a symmetric session key $K_{A,B}$ can be now easily derived on both sides: e.g. $K_{A,B} = H(r_A , r_B)$, where H is cryptographic hash function that has been agreed on a priori.

The Needham-Schroeder Public Key Protocol (4)

- Attack:
  - The *Needham-Schroeder Public Key Protocol* is vulnerable to a *man-in-the-middle attack.*
  - If an attacker *M* can persuade *A* to initiate a session with him, he can relay the messages to *B* and convince *B* that he is communicating with *A*.
  - For simplicity, we don't illustrate the communication with *AS* here, which remains unchanged.

    3') $A \rightarrow M\text{: } \{ r_A , A \}_{K_{M\text{-}pub}}$

    - A sends r_A to *M*, who decrypts the message with $K_{M\text{-}priv}$

    3'') $M \rightarrow B\text{: } \{ r_A , A \}_{K_{B\text{-}pub}}$

    - *M relays the message to B, pretending that A is communicating*

    6') $B \rightarrow M\text{: } \{ r_A , r_B \}_{K_{A\text{-}pub}}$

    - *B* sends r_B

    6'') $M \rightarrow A\text{: } \{ r_A , r_B \}_{K_{A\text{-}pub}}$

    - *M* relays it to *A*

The Needham-Schroeder Public Key Protocol (5)

- Attack on the Needham-Schroeder public key protocol (continued):

  7') $A \rightarrow M: \{ r_B \}_{K_{M\text{-}pub}}$

  - *A* decrypts r_B and confirms it to *M*, who learns it

  7'') $M \rightarrow B: \{ r_B \}_{K_{B\text{-}pub}}$

  - *M* re-encrypts r_B and convinces *B* that he has decrypted it.

- At the end of the attack, *B* falsely believes that *A* is communicating with him, and that r_A and r_B are known only to *A* and *B*.
- The attack was first described in 1995 by Gavin Lowe [Lowe95].
- The paper also describes a fixed version of the protocol, referred to as the *Needham-Schroeder-Lowe* protocol. The fix involves the modification of message (6)

  6.) $B \rightarrow A: \{ r_A , r_B \}_{K_{A\text{-}pub}}$

  which is replaced with the fixed version

  6.) $B \rightarrow A: \{ r_A , r_B , B \}_{K_{A\text{-}pub}}$

Overview

- Part I: Introduction
- Part II: The Secure Channel
- Part III: Authentication and Key Establishment Protocols
  - Introduction
  - Key Distribution Centers (KDC)
  - Public Key Infrastructures (PKI)
  - Building Blocks of key exchange protocols

Certificates ~ Passports in Network Security

Certificate

- Generated by Certificate Authority (CA) for an entity
- Purpose
  - The CA states that an entity and a public key correspond.
- A certificate contains
  - Cleartext
    - **Name of the entity (e.g. Bob)**
    - **Public Key of entity**
    - Name of the CA
    - (optionally) further data about the entity
      - E.g. is it also a CA?
    - (optionally) more data about CA
    - for all the cryptographic operations the algorithms that are used
  - **Signature by the CA**
    - Hash value of cleartext signed with private key of CA

Trusted Root Certificate
--- for ----
Name: GlobalCA
Public Key:
RSA 29302048934
….
--- by ---
CA: GlobalCA
--- Signature ---
4850300434040

Certificate
--- for ----
Name: Bob
Public Key:
RSA 47399844398
….
--- by ---
CA: GlobalCA
--- Signature ---
10493850405

Alice, Bob, and all other entities have stored this certificate on their device because they trust this authority.
➔ They know its public key!

PKI – Overview (more on PKI in a separate chapter)

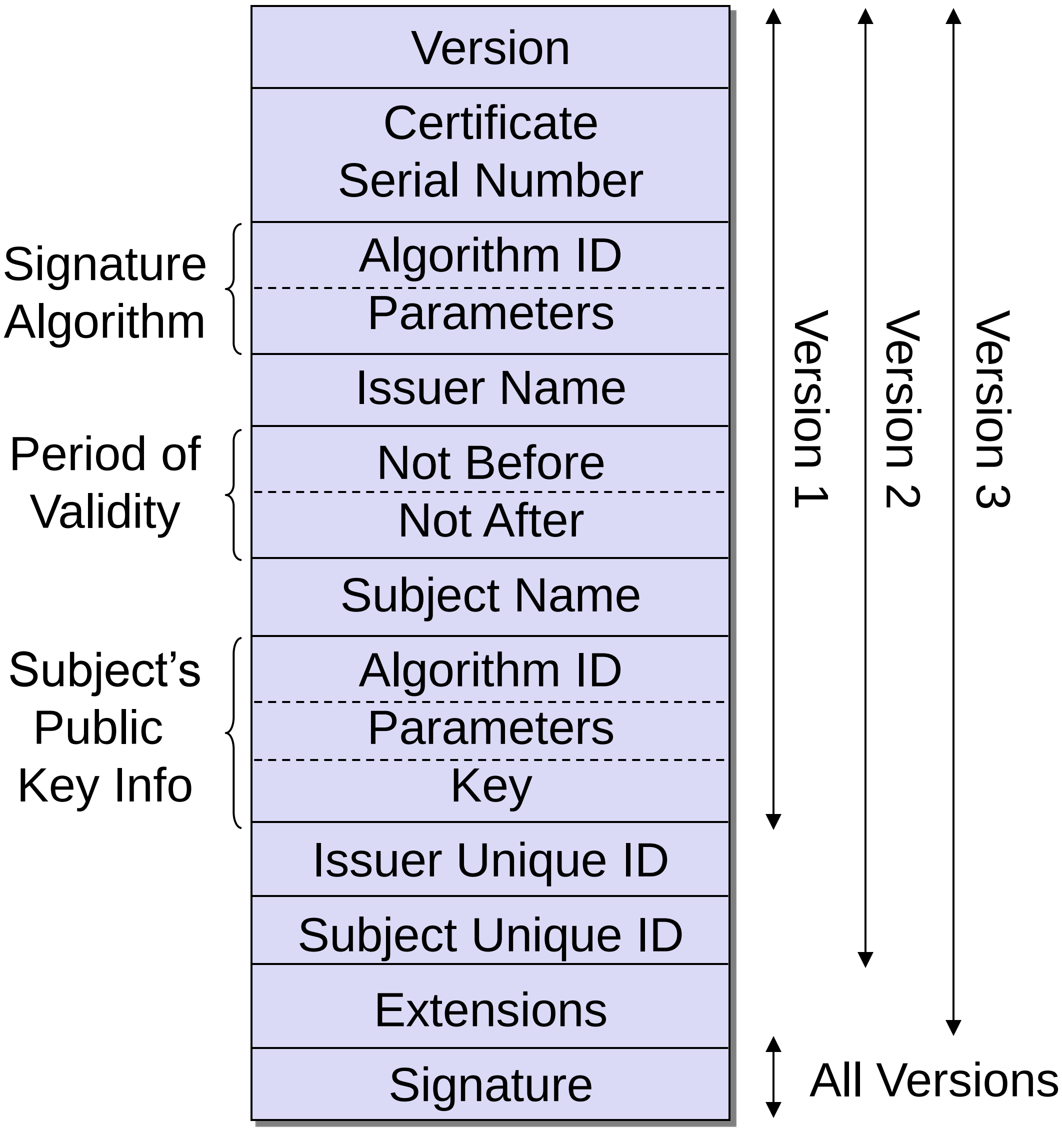
- Each entity has a public key/private key pair,

  e.g. RSA or ECC public/private keys
- Each entity has a „certificate“ that binds its „name“ to its public key
- Note: in a networking environment “names” could be
  - a user name (optionally with an email address)
  - But it could be also e.g. IP addresses, the DNS name of the node, etc.
- A Certificate Authority (CA) asserts the correctness of the certificate by signing it with her private key.
- CA is a trusted third party (TTP) that is trusted by all the entities.
- Furthermore, each entity knows the public key of CA
- When Alice wishes to communicate with Bob, she can receives Bob‘s certificate
  - E.g. from a directory service or from Bob himself at the beginning of the authentication procedure
- Since Alice knows CA’s public key, she can verify the signature of Bob’s certificate that was generated by CA

X.509 PKI Authentication Services – 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

X.509 – Public Key Certificates (1)



Version

Certificate Serial Number

Signature Algorithm: Algorithm ID, Parameters

Issuer Name

Period of Validity: Not Before, Not After

Subject Name

Subject's Public Key Info: Algorithm ID, Parameters, Key

Issuer Unique ID

Subject Unique ID

Extensions

Signature

Version 1

Version 2

Version 3

All Versions

- 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 $K_{A\text{-}pub}$ to user A issued by certification authority CA using its private key $K_{CA\text{-}priv}$:
  - $Cert_{K_{CA\text{-}priv}}(K_{A\text{-}pub}) = CA[V, SN, AI, CA, T_{CA}, A, K_{A\text{-}pub}]$

    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
$K_{A\text{-}pub}$	= public key to be bound to a name

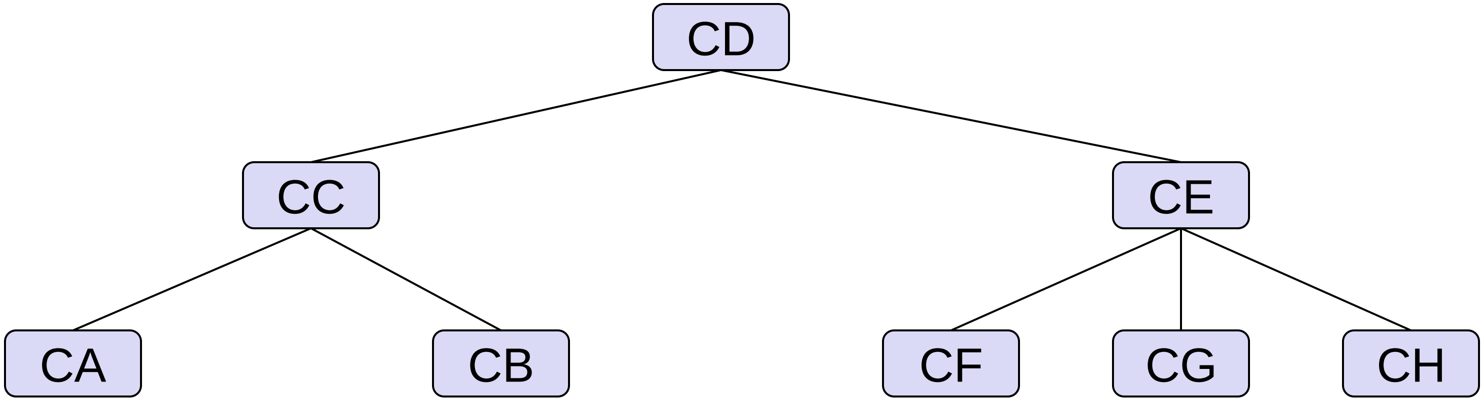
  - The shorthand notation $CA[m]$ stands for $(m, \{H(m)\}_{K_{CA\text{-}priv}})$
  - Another shorthand notation for $Cert_{K_{CA\text{-}priv}}(K_{A\text{-}pub})$ is CA<<A>>

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<<B>>* is useless to her, and the same applies in the other direction
- A solution to this problem is to construct *certificate chains*:
  - Imagine for a moment that *CA* and *CB* know and trust each other
    - A real world example of this concept is the mutual trust between countries considering their passport issuing authorities
  - If *CA* certifies *CB*'s public key with a certificate *CA<<CB>>*, then *A* can check *B's* certificate by checking a certificate chain:
    - Upon being presented *CB<<B>>* Alice tries to look up if there is a certificate *CA<<CB>>*
    - She then checks the chain: *CA<<CB>>, CB<<B>>*
  - In WWW (SSL/TLS) it is expected that B (= server) sends the complete chain to A. Assumption: a certain set of worldwide Root CAs is known by all participants.

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 $K_{G\text{-}priv}$ 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 (1)

- When a certificate is issued, it is expected to be in use for its entire validity period.
- However, various circumstances may cause a certificate to become invalid prior to the expiration of the validity period.
- Reasons for revocating a certificate:
  - The information in the certificate is not valid anymore.
  - The private key can not be used anymore, e.g. because
    - the physical medium where the private key was stored becomes defect, e.g. the hard disk, the USB stick or the smart card.
    - the physical medium where the private key is stored has been stolen.
    - the private is protected with a password and the password can not be recovered.
  - The private key is (partially) revealed or at least assumed to be revealed, e.g. a Trojan horse or a key logger has been discovered on the computer.
  - The parameters of the certificate become inadequate, e.g.
    - The cryptographic algorithm is broken.
    - The key length is considered as inappropriate.

X.509 – Certificate Revocation (2)

- An even worse situation occurs if 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
- Each CA issues a signed data structure periodically called a certificate revocation list (CRL).

→ Certificate revocation is a relatively slow and expensive operation

Online Certificate Status Protocol (OCSP)

- The CRL can be accessed with the *Online Certificate Status Protocol (OCSP)*
- An OCSP client issues a status request to an OCSP server and suspends acceptance of the certificate in question until the responder provides a response.
- CAs that support an OCSP service, either hosted locally or provided by an Authorized Responder, provide the necessary information for the online validation of the status of the certificate.
- OCSP just ports revocation status (OSCP does not do certificate verification).
- The certificate validation process is rather resource-consuming.
  - Therefore, in some environments, e.g. with cell phones, it would be desirable to fully off-load the certificate validation process to an external trusted entity.
  - The *Simple Certificate Validation Protocol (SCVP)* [RFC5055] offers this functionality.

PKI - Discussion

- PKIs assume a relationship between the CA and the entities, which is not always available:
  - There is no „global“ PKI
  - There is no worldwide CA. (But CAs might “cross-certify” each others)
- It remains questionable whether a CA executes its task faithfully, i.e., whether a CA verify the identity of the users thoroughly.
- In particular, if the CA certifies millions of users.
- Nevertheless, PKIs are very commonly used
  - They are integrated, e.g. in each Internet browser
  - Every Internet-Browser has a list of „root CAs“ that are considered as trusted.

Overview

- Part I: Introduction
- Part II: The Secure Channel
- Part III: Authentication and Key Establishment Protocols
  - Introduction
  - Key Distribution Centers (KDC)
  - Public Key Infrastructures (PKI)
  - Building Blocks of key exchange protocols

Problem Statement

(c.f. Niels Ferguson, Bruce Schneier: Practical Cryptography, Ch. 15, pp. 261ff)

- Assumption
  - Alice and Bob are able to authenticate messages to each other, e.g.
    - Using RSA signatures, if Alice and Bob know each other's public keys or using a PKI
    - Using a long term pre-shared secret key and a MAC function
- Goal
  - Run a key exchange protocol such as at the end of the protocol:
    1. Alice and Bob have agreed on a shared „session key“ for a secure channel
    2. Alice and Bob have agreed on the cryptographic algorithms to be used for the secure channel
    3. Alice (Bob) must be able to verify that Bob (Alice) knows *K* and that he (she) is “alive”
    4. Alice and Bob must know that *K* is newly generated
- Note: even if Alice and Bob possess a long term pre-shared secret key, it is recommended to perform a key exchange in order to derive a separate session key

Reasons for Separating Session Keys and Long-Term Keys

- Why do we need a session key if we already have a (long term) key?

- De-couple the session key from the long-term key
  1. If the session key is compromised, e.g. because of a flawed implementation of the secure channel, then the long-term shared secret should remain safe.
  2. If the long-term key is compromised *after* the key negotiation has been run, the attacker who learns the shared secret key still does not learn the session key negotiated by the protocol, i.e. yesterday‘s data is still protected if the long-term key is compromised today.
  - These properties are important and make the entire system more robust
  - The 2nd property is called „*Forward Secrecy*“

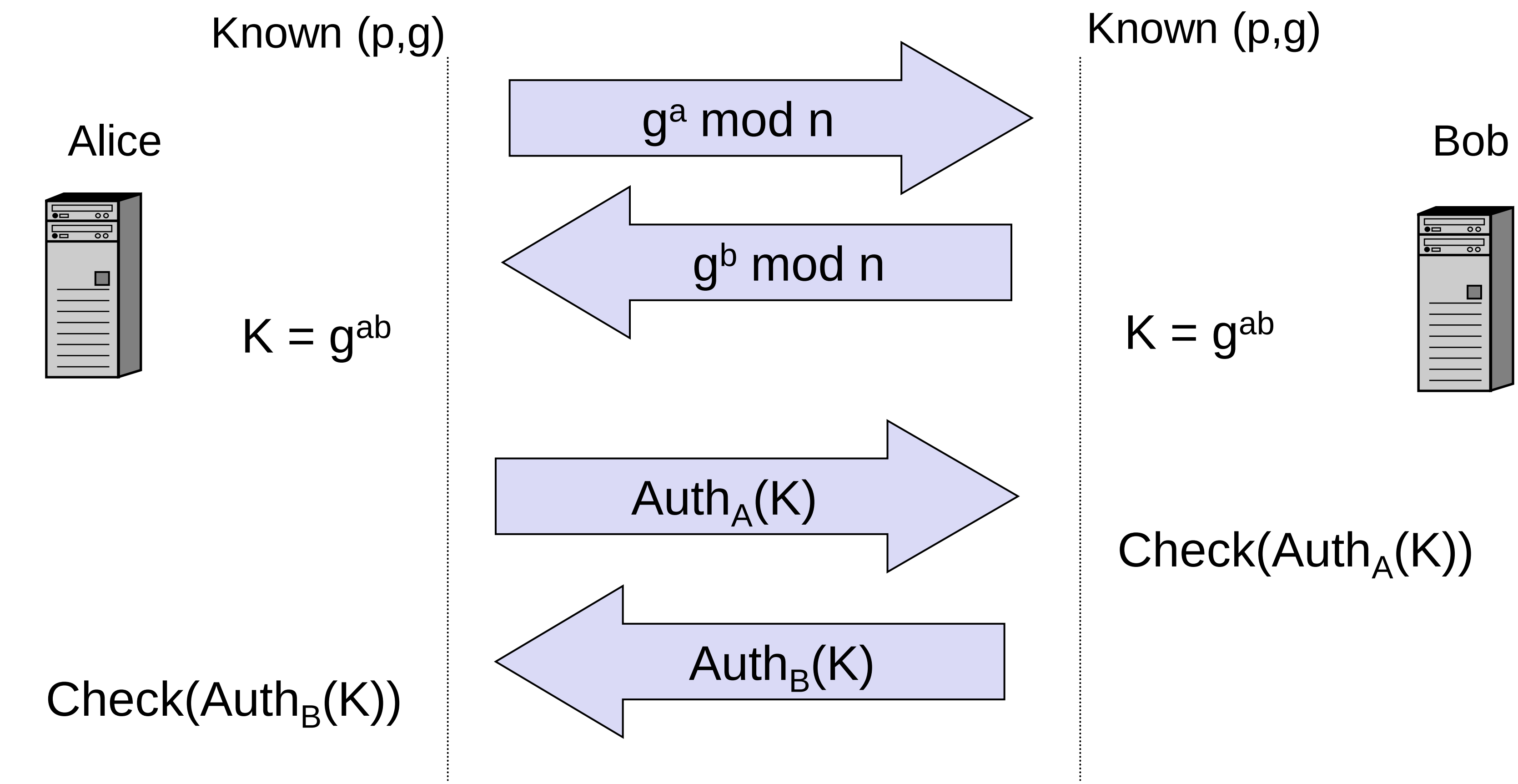
- Definition: Forward Secrecy [Boyd03]
  - A key establishment protocol provides *forward secrecy* if compromise of the long-term keys of a set of entities (private keys or symmetric keys) does not compromise the session keys established in previous protocol runs involving these entities.

Other Reasons for Separating Session Keys and Long-Term Keys

- Sometimes the long-term key is weak, e.g. passwords
  - Users do not want to memorize a 30-letters password
  - They tend to choose much simpler ones
- In some cases, the session key needs to be changed before the session is over (re-keying)
  - This is, e.g., the case if the message sequence numbers overflow and need to be reset
  - This would be problematic if the session key is equal to the long-term key

First Try

- Alice and Bob perform a Diffie-Hellman key exchange and then authenticate the obtained key *k*



Problems with “First Try”

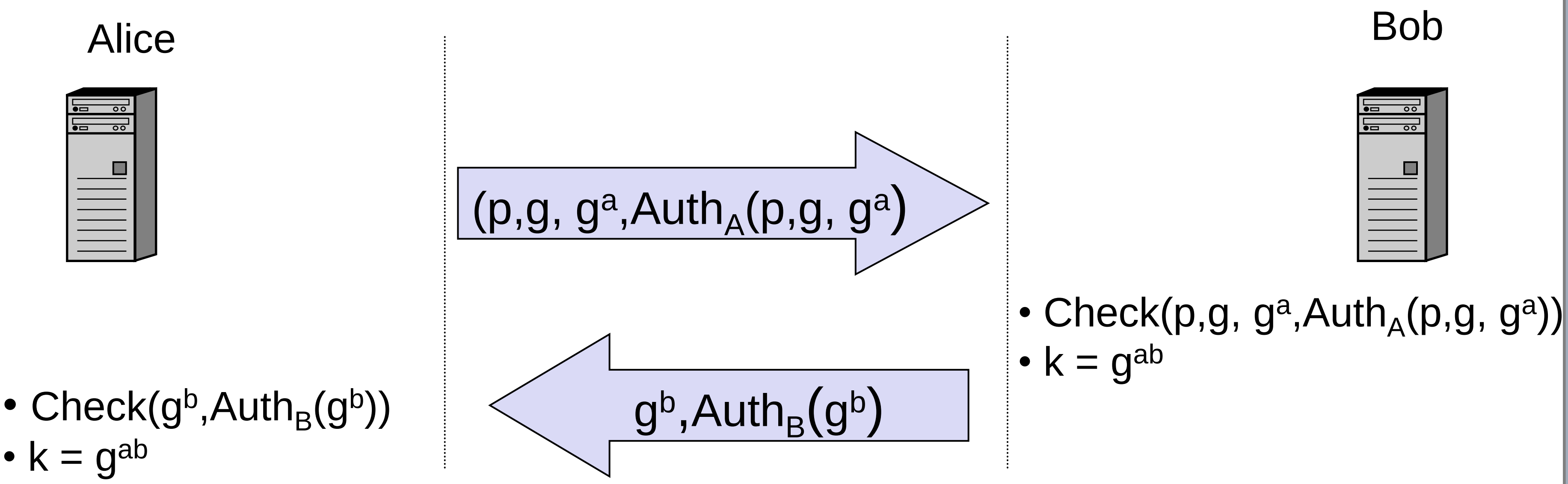
- Alice and Bob use constant DH parameters *p* and *g*
  - This is a bad design, since
    - *p* and *g* might be considered as insecure after a while
    - Protocols live for a long time. Using the same constants raises interoperability issues
- The exchange uses 4 messages, whereas it is possible to achieve the goal using 3 messages
- *K* is used as input for the authentication function *Auth*
  - This would be fine, if *Auth* is a strong function
  - But if *Auth*(*K*) leaks some knowledge about *K* this would require a new analysis of the entire protocol
  - A rule of thumb: “*Secrets should be used only for a single purpose*”.
- The authentication messages are too similar
  - If *Auth* is a MAC function, then $Auth_B(K) = Auth_A(K)$

  ➔ Bob can just send the authentication value that he received from Alice.

  ➔ At the end of the protocol run, Alice can not be sure that Bob has the correct key

Second Attempt

- Alice chooses the DH parameters *p* and *g*
  - Bob verifies that he supports *p* and *g*
- The protocol exchange is reduced to 2 messages



Alice

Bob

$(p,g, g^a,Auth_A(p,g, g^a))$

- $Check(p,g, g^a,Auth_A(p,g, g^a))$
- $k = g^{ab}$

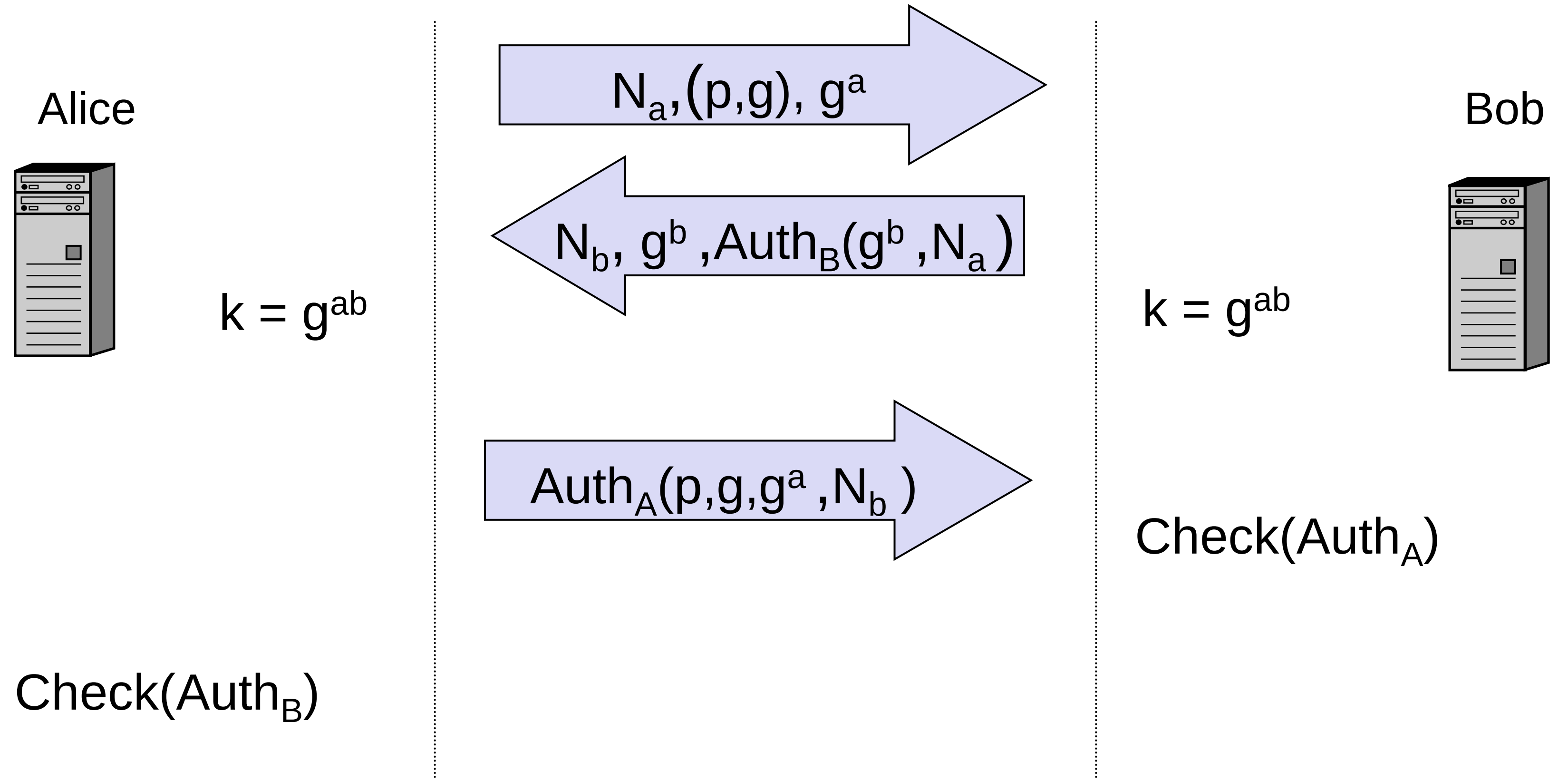
$g^b,Auth_B(g^b)$

- $Check(g^b,Auth_B(g^b))$
- $k = g^{ab}$

Second Attempt, Evaluation

- DH parameters are chosen dynamically
  - If *p* is not large enough, Bob can send an error message to Alice with the minimal supported length for *p* and abort the protocol run
- The protocol run requires only 2 messages
- The key g^{ab} is not used anymore for the authentication of messages
- Strings that are being authenticated are not the same
- However, a replay attack is possible
  - Bob can not be sure that he is actually talking to Alice
  - Anybody can record the first message that Alice sends and then later send it to Bob
  - Bob verifies $Auth_A$ and finishes the protocol thinking that he has just shared a session key *k* with Alice
- This problem is called *the lack of liveliness*
  - Bob can not be sure that Alice is „alive“, and he is not talking to a replaying attacker
  - The typical way to solve this problem is to make sure that Alice‘s authenticator $Auth_A$ covers a random value that has been chosen by Bob

Third Attempt



Alice

Bob

$N_a, (p,g), g^a$

$N_b, g^b, Auth_B(g^b, N_a)$

$k = g^{ab}$

$k = g^{ab}$

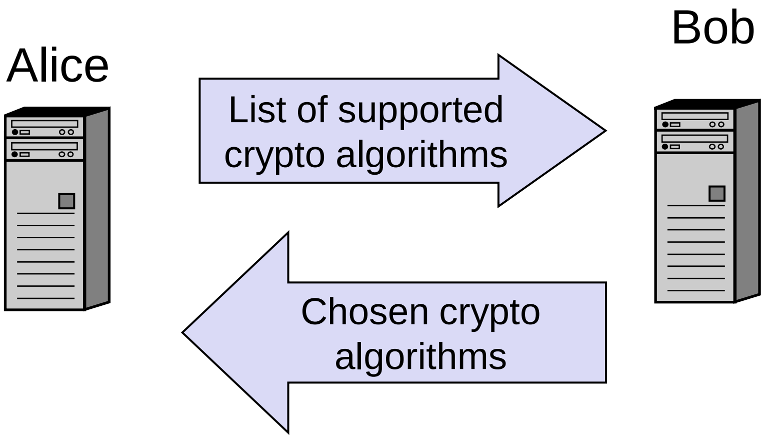
$Auth_A(p,g,g^a, N_b)$

$Check(Auth_A)$

$Check(Auth_B)$

Further Design Issues: Dynamic Negotiation of Crypto Algorithms

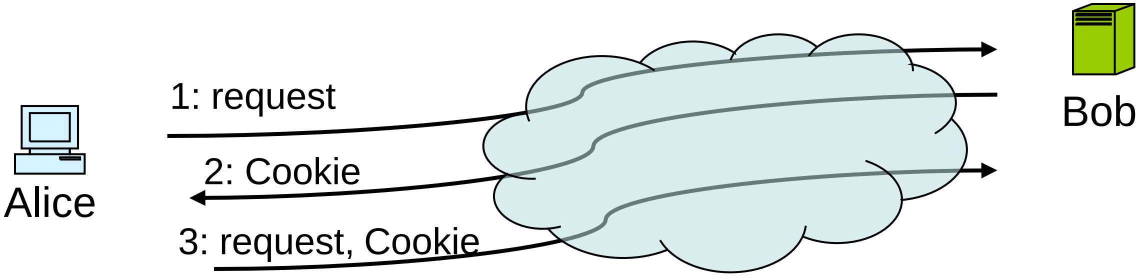
- Alice and Bob need to agree on the cryptographic algorithms to be used for encryption and data integrity
  - Facilitates the support of new stronger cryptographic algorithms
  - Deprecated cryptographic algorithms can be removed easily
  - Upgrades do not require an additional standardization process



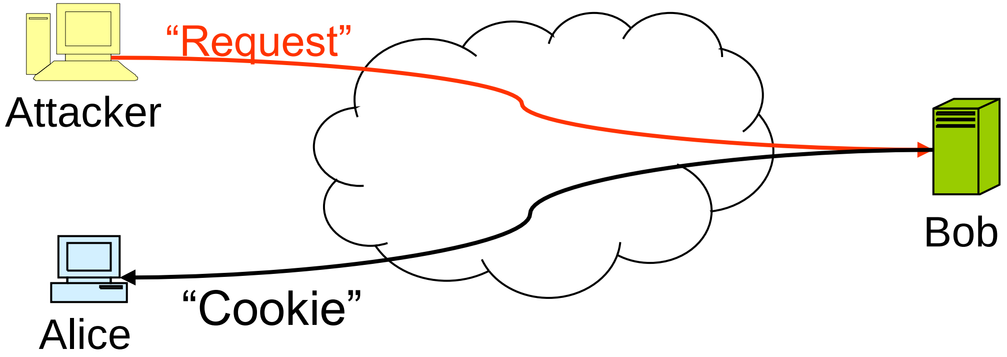
Further Design Issues: Denial-of-Service Protection (1)

- Bob may be flooded with a large number of requests for establishing a secure channel from a large number of attackers
- This phenomena is called *Distributed Denial-of-Service attacks (DDoS)*
- Since Bob needs to store state and perform computation for each request, a DoS attack would exhaust Bob‘s resources, such as CPU and memory
- Possible Countermeasures:
  - Before processing a new request, verify if the “initiator” can receive messages sent to the claimed source of the request (see next slide)

Denial-of-Service Protection with Cookies (1)



- Upon receiving a request from Alice, Bob calculates a Cookie and sends it to Bob.
- Alice will receive the Cookie and resend the request with the Cookie together.
- Bob verifies that the Cookie is correct and then starts to process Alice‘s request.
- An attacker that is sending requests with a spoofed (i.e. falsified) source address will not be able to send the Cookie.



Denial-of-Service Protection with Cookies (2)

- Requirements:
  - An attacker that is not on the path between Alice and Bob must not be able to guess the correct value of the Cookie
  - Bob must be able to generate the Cookie after receiving message 1 with minimal processing (CPU friendly)
  - Bob must be able to verify that the Cookie is correct upon receipt of message 3, without necessarily storing any information after message 1 (memory friendly)

    ➔ Bob must be able to re-calculate the Cookie sent in message 2 and verify that the received Cookie from Alice in message 3 is correct
- One possible way to compute the cookie could be as follow:

  $Cookie = \text{Hash}(N_a \mid Address_{Alice} \mid <secret>)$

where

  - N_a is the nonce sent by Alice (as above)
  - $<secret>$ is randomly generated secret known only to Bob
  - Hash is a cryptographic hash function.
- Only a legitimate initiator (Alice) or a host on the path can read the "cookie" and can send the cookie back to the responder (Bob)

Denial-of-Service Protection with Cookies (3)

- Additional requirement:
  - <*secret*> needs to be changed regularly. Otherwise, it can be brute-forced successfully after a while

➔ Another possible way to compute the cookie could be as follow:

$Cookie$ = <$Version\ ID\ of\ Secret$> | Hash(N_a | IP_a | <$secret$>)

where

- <*Version ID of Secret*> is changed whenever <*secret*> is regenerated.

- Cookies discussion:
  - Advantage: allows to counter simple address spoofing attacks
  - Drawbacks:
    - requires one additional message roundtrip.

Further Design Issues: Reuse of the DH Values

- The calculation if the DH values g^a and g^b is computationally expensive
- Alice and Bob may re-use the values g^a and g^b
- However, Alice and Bob must ensure that the key has been freshly generated
- ➔ The random numbers N_a and N_b can be included in the computation of the shared key
- ➔ One possible way to compute the session key:
  - $K = H\ (N_a \mid N_b \mid g^{ab})$ where H is a cryptographic hash function
- However, the re-use of the DH values affects the property of (perfect) forward secrecy (see next slide).

Forward Secrecy (1)

- The DH exchange is not only used to gain a shared secret g^{ab} (that needs to be authenticated).
- The DH exchange offers also the property of "forward secrecy“
  - If any long term keys,
    - the long-term pre-shared secret key between Alice and Bob
    - or Alice/Bob private key

    is compromised, an attacker that has recorded previous protocol runs, would need to compromise the DH exchange as well in order to gain the session keys for these previous sessions.
- Forward Secrecy was originally called “Perfect Forward Secrecy” (PFS)
  - Many cryptographers did not agree with the term “perfect”, so it is usually skipped.
- PFS requires that when a session is closed, each endpoint forgets
  - all the keying material used for this session
  - any information that could be used to recompute those keys
  - In particular, it needs to forget the secrets used in the DH calculation and the state of a pseudo-random number generator that could be used to re-compute the DH secrets.

Forward Secrecy (2)

- Note
  - By running a key exchange protocol, PFS is usually provided with the DH exchange
  - Many protocols do not provide PFS, since DH is computationally intensive
  - Examples
    - IPSec IKEv (Version 1 and Version 2): yes,
    - TLS: PFS optionally provided with ephemeral (temporary) DH
    - WLAN: WEP, WPA: no
    - GSM/UMTS Authentication and Key Exchange (AKA): no

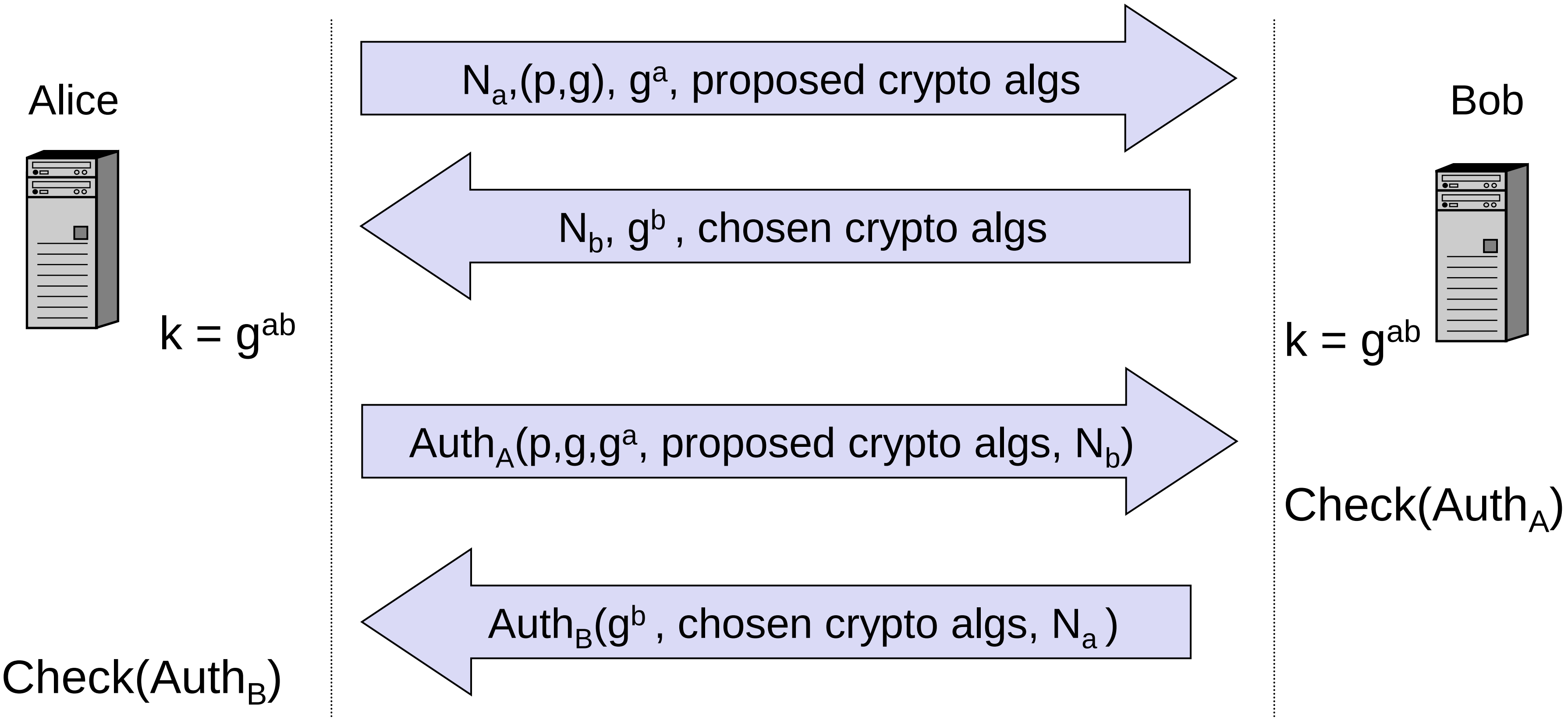
      (although some commercial products do already support PFS for GSM networks. But both mobile phones need to support it)

Further Design Issues: Simplicity

- „A more complex system loses on all fronts. It contains more weaknesses to start with, it is much harder to analyze, and it is much harder to implement without introducing security-critical errors in the implementation.“ [Fer00]

- An important design criterion for a new protocol is that the protocol state machine should be as simple as possible.
- Especially for security protocols, the simpler the state machine is the easier the security analysis of the protocol can be.
- Remember that an attacker can send any type of message at any time to any participant in the protocol.
- One way to reduce the complexity is to design the protocol such as it consists of pairs of messages:
  - a request
  - and a response.
- Every request requires a response.

Final protocol design attempt



Alice

Bob

N_a,(p,g), g^a, proposed crypto algs

N_b, g^b , chosen crypto algs

$k = g^{ab}$

$k = g^{ab}$

$Auth_A(p,g,g^a$, proposed crypto algs, $N_b)$

Check($Auth_A$)

$Auth_B(g^b$, chosen crypto algs, N_a)

Check($Auth_B$)

Online TTP not always a KDC (knows session key)

- In the lecture slides, we only look at Key Distribution Centers (KDC) in case of symmetric encryption
  - Key Transport Protocol instead of Key Agreement Protocol
- Key Transport
  - One party generates key and „*transports*“ it to the other parties.
- Key Agreement
  - The key is generated by the interaction of multiple parties. In the end, they agree on the same key.

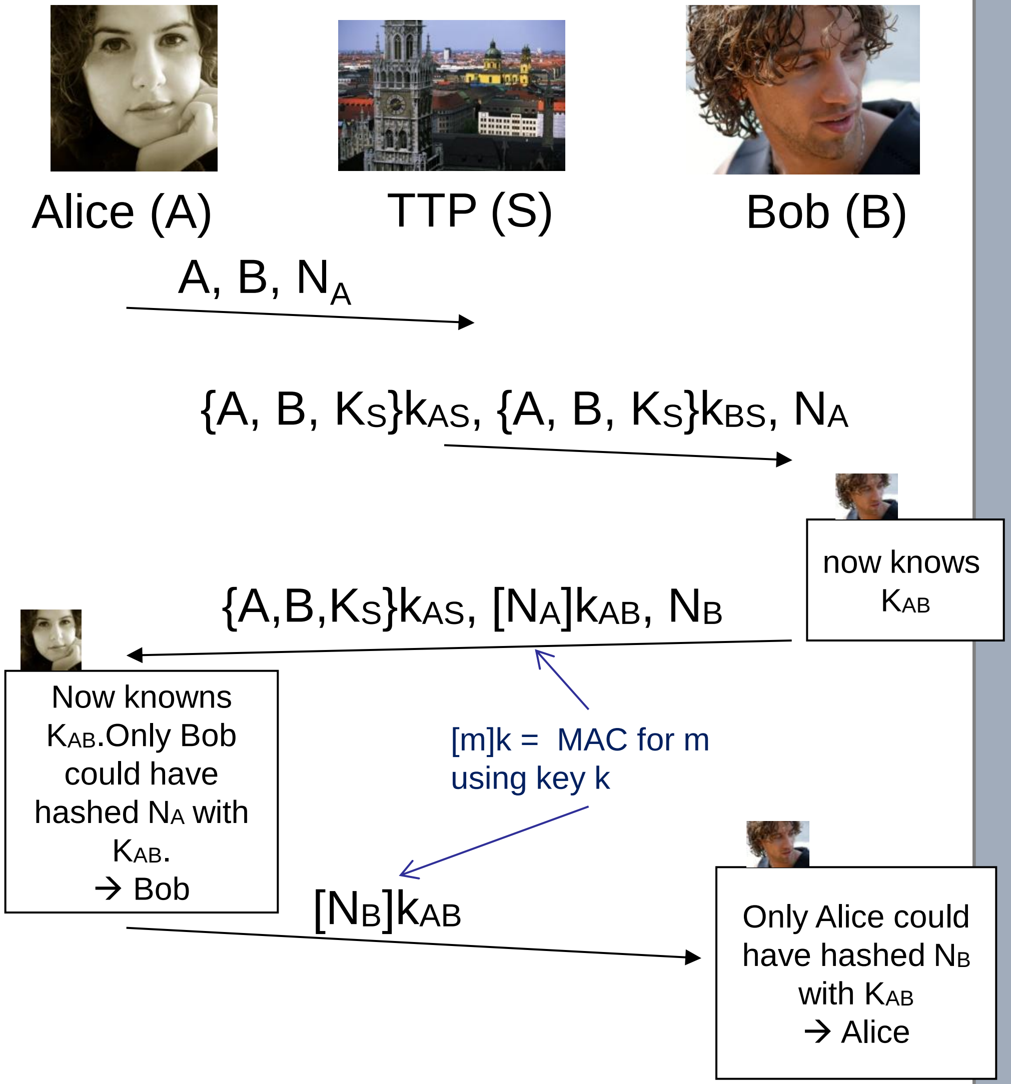
Example for Key Agreement Protocol

Boyd Key Agreement Protocol

- Assumptions
  - Trusted Party TTP exists
  - A and B each share a secret key with a TTP (K_{AS}, K_{BS}).
- Key

$$K_{AB} = MAC_{K_S}(N_A, N_B)$$

- Provides
  - Mutual authentication
  - Key is authenticated, fresh, and confirmed.
  - **Key Agreement**
    - **All 3 entities contribute to key**.
    - **TTP does not know K_{AB}.**
- No known attack.
- No forward secrecy.



Alice (A)

TTP (S)

Bob (B)

A, B, N_A

{A, B, K_S}k_{AS}, {A, B, K_S}k_{BS}, N_A

now knows K_{AB}

{A,B,K_S}k_{AS}, [N_A]k_{AB}, N_B

Now knowns K_{AB}.Only Bob could have hashed N_A with K_{AB}.
→ Bob

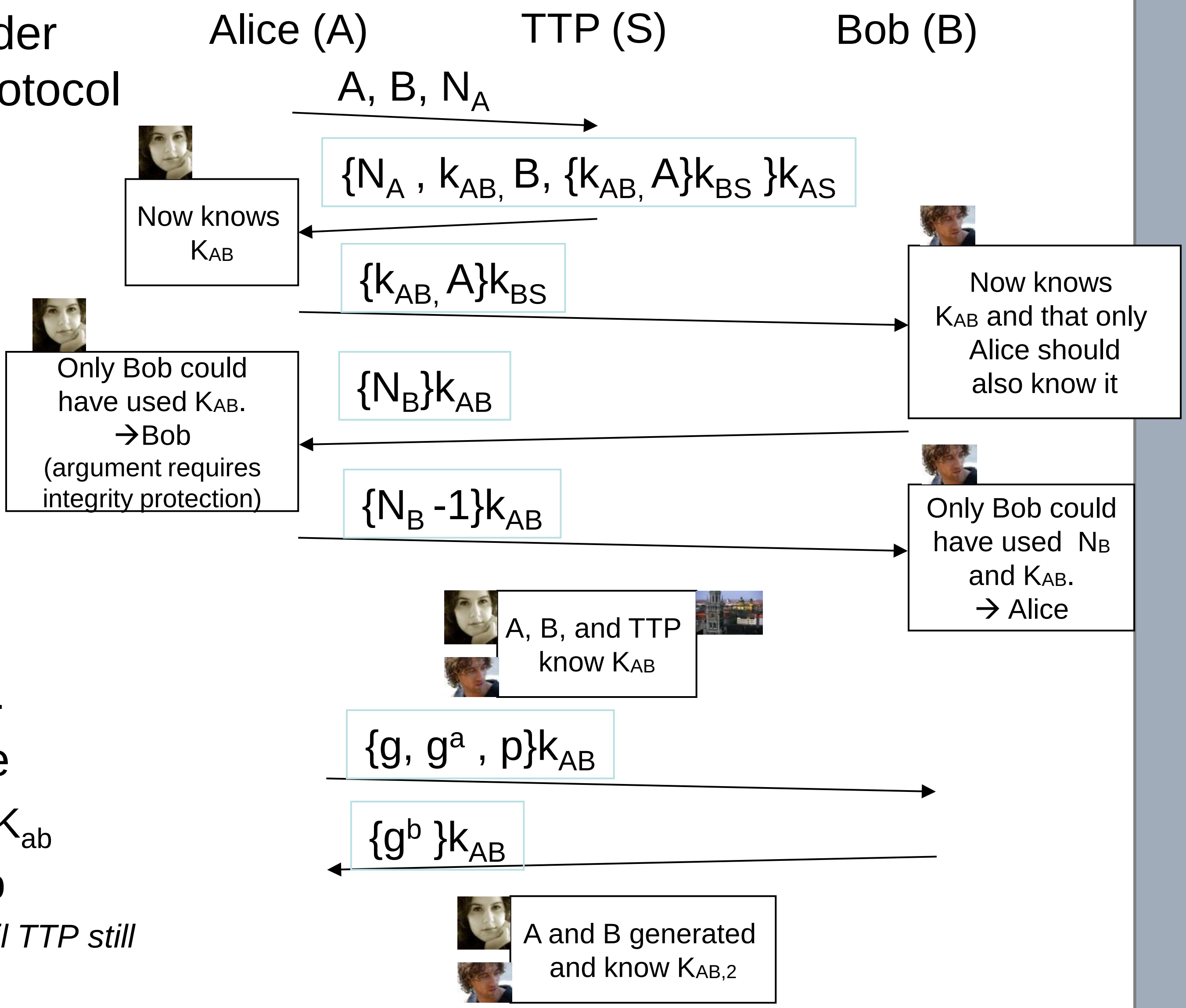
[m]k = MAC for m using key k

[N_B]k_{AB}

Only Alice could have hashed N_B with K_{AB}
→ Alice

Key Agreement using Key Transport plus DH

- Needham-Schroeder Symmetric Key Protocol
  - Key Transport
  - TTP knows key
- Option: Add Diffie-Hellman exchange
  - Secured due to K_{ab}
  - $K_{ab,2} = g^{ab} \bmod p$
  - *Question: Can an evil TTP still attack?*



Alice (A) TTP (S) Bob (B)

A, B, N_A

$\{N_A , k_{AB,} B, \{k_{AB,} A\}k_{BS} \}k_{AS}$

Now knows K_{AB}

$\{k_{AB,} A\}k_{BS}$

Now knows K_{AB} and that only Alice should also know it

$\{N_B\}k_{AB}$

Only Bob could have used K_{AB}. → Bob (argument requires integrity protection)

$\{N_B -1\}k_{AB}$

Only Bob could have used N_B and K_{AB}. → Alice

A, B, and TTP know K_{AB}

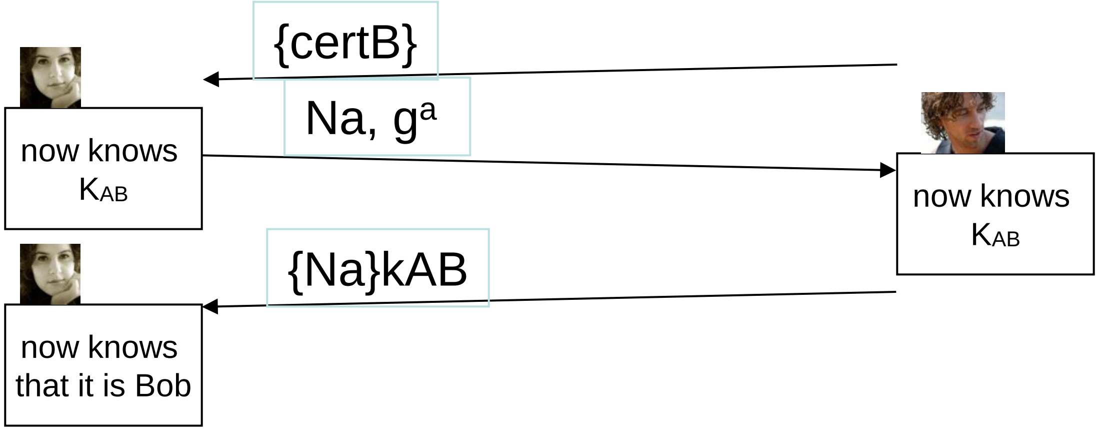
$\{g, g^a , p\}k_{AB}$

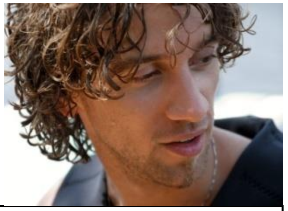
$\{g^b \}k_{AB}$

A and B generated and know $K_{AB,2}$

Forward Secrecy and Diffie-Hellman Value as Public Key?

- Assume that Bob has this certificate.





Certificate
--- for ----
Name: Bob
Public Key:
DH 49583385
g 9303
p 2094739744
--- by ---
CA: GlobalCA
--- Signature ---
10493850405

- The result is a shared key that only Bob could have generated from Alice‘s request.
- If g and p are fixed, then also Alice could also send a certificate and mutual authentication would be possible.
- However, you cannot sign or encrypt with it. It only generates a symmetric key.

➔ Possible to build a PKI from DH. Actually, SSL/TLS support this (hardly used, if at all).

➔ *No Forward Secrecy!*

References

[Bell95] M. Bellare and P. Rogaway, Provably Secure Session Key Distribution - The Three Party Case, Proc. 27th STOC, 1995, pp 57--64

[Boyd03] Colin Boyd, Anish Mathuria, “Protocols for Authentication and Key Establishment”, Springer, 2003

[Bry88a] R. Bryant. *Designing an Authentication System: A Dialogue in Four Scenes.* Project Athena, Massachusetts Institute of Technology, Cambridge, USA, 1988.

[Diff92] W. Diffie, P. C. van Oorschot, and M. J. Wiener. Authentication and authenticated key exchanges. Designs, Codes, and Cryptography, 1992

[Dol81a] D. Dolev, A.C. Yao. *On the security of public key protocols.* Proceedings of IEEE 22nd Annual Symposium on Foundations of Computer Science, pp. 350-357, 1981.

[Fer00] Niels Ferguson, Bruce Schneier, “A Cryptographic Evaluation of IPsec”. http://www.counterpane.com/ipsec.pdf 2000

[Fer03] Niels Ferguson, Bruce Schneier, „Practical Cryptography“, John Wiley & Sons, 2003

[Gar03] Jason Garman, “Kerberos. The Definitive Guide”, O'Reilly Media, 1st Edition, 2003

References

[Kau02a]	C. Kaufman, R. Perlman, M. Speciner. *Network Security*. Prentice Hall, 2nd edition, 2002.
[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.
[Mao04a]	W. Mao. *Modern Cryptography: Theory & Practice*. Hewlett-Packard Books, 2004.
[Nee78]	R. Needham, M. Schroeder. *Using Encryption for Authentication in Large Networks of Computers.* Communications of the ACM, Vol. 21, No. 12, 1978.
[Woo92a]	T.Y.C Woo, S.S. Lam. *Authentication for distributed systems.* Computer, 25(1):39-52, 1992.
[Lowe95]	G. Lowe, „An Attack on the Needham-Schroeder Public-Key Authentication Protocol", *Information Processing Letters*, volume 56, number 3, pages 131-133, 1995.

Additional references from the IETF

[RFC2560] M. Myers, et al., “X.509 Internet Public Key Infrastructure Online Certificate Status Protocol – OCSP”, June 1999

[RFC3961] K. Raeburn, “Encryption and Checksum Specifications for Kerberos 5”, February 2005

[RFC3962] K. Raeburn, “Advanced Encryption Standard (AES) Encryption for Kerberos 5”, February 2005

[RFC4757] K. Jaganathan, et al., “The RC4-HMAC Kerberos Encryption Types Used by Microsoft Windows ”, December 2006

[RFC4120] C. Neuman, et al., “The Kerberos Network Authentication Service (V5)”, July 2005

[RFC4537] L. Zhu, et al, “Kerberos Cryptosystem Negotiation Extension”, June 2006

[RFC5055] T. Freeman, et al, “Server-Based Certificate Validation Protocol (SCVP)”, December 2007