IT-Security

Chapter 5: Authentication and Key Establishment

Prof. Dr.-Ing. Ulrike Meyer

Overall Lecture Context

- **In the last chapters we covered**
  - Symmetric and asymmetric mechanisms to provide
  - Integrity protection
    - Message Authentication Codes and digital signatures schemes
  - Confidentiality
    - Symmetric and asymmetric encryption schemes
- **All these mechanisms require keys to be distributed**
  - to the authentic entities
- **In this chapter we learn how to**
  - authenticate entities, i.e., check that they are who they claim to be
  - establish keys between different entities

Overview

- **Building Blocks for Entity Authentication**
  - Definition of Entity Authentication
  - MAC-based authentication
  - Signature-based authentication

- **Key Distribution with trusted Third Parties**
  - Key Distribution Centers
  - Certificates and Public Key Infrastructures

- **Authenticated Session Key Establishment**
  - Definitions around session key establishment
  - Authenticated Diffie Hellman variants
  - Session key establishment w-o DH
  - Session Key derivation principles

- **Password-based authentication**
  - Password-based user authentication
  - Password-based authenticated key establishment
  - Dictionary attacks on password-based authentication

Definition of Entity Authentication

Unilateral entity authentication of A to B

- **A** (claimant) proofs its identity to **B** (verifier)
- **B** is assured that **A** is currently interacting with **B**

Mutual authentication

- **A** authenticates to **B** and **B** authenticates to **A**

Objectives

- Correctness: **A** can always successfully authenticate to **B**
- Resistance against transferability: After **A** authenticated to **B** successfully, **B** cannot authenticate as **A** to **C** (∗)
- Resistance against impersonation: **C** ≠ **A** cannot make **B** believe that it is **A** (∗)

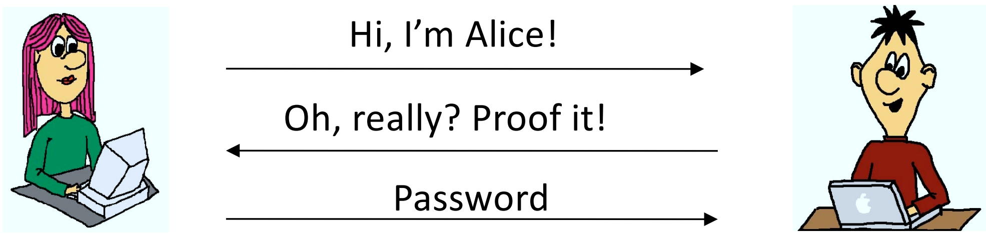
All three objectives still hold

- if an attacker has observed multiple authentication instances between A and B

(∗) Except for with negligible probability: guessing is of course always possible

Example

- Assume **A** and **B** have agreed upon a secret password when they last met
- Now **A** authenticates to **B** with the following protocol



Correct?

- Yes!

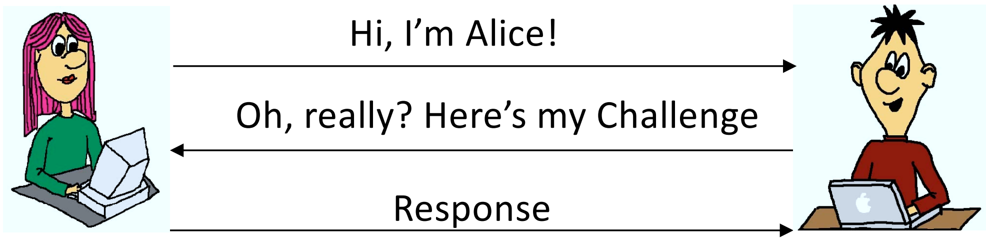
Resistant against transferability?

- Yes, at least if Alice does not use the password in multiple places

Resistant against impersonation?

- No! The password is sent in the clear so any eavesdropper can impersonate Alice after the first run of the protocol

Challenge-Response Authentication

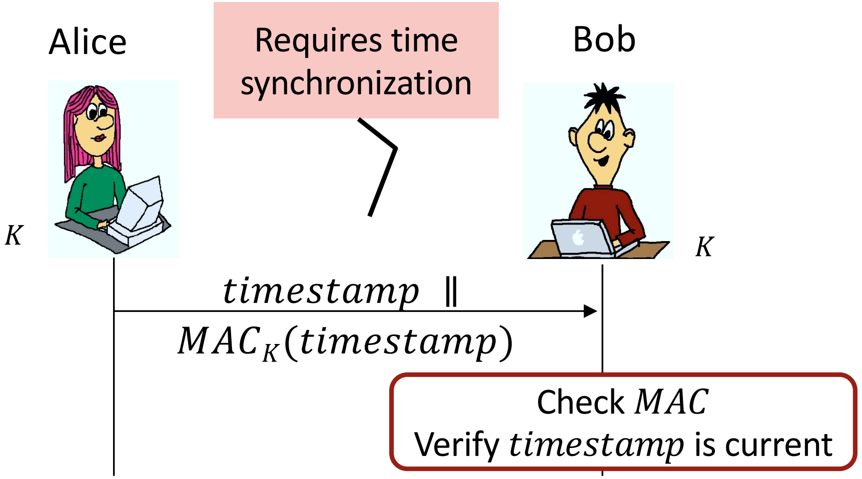


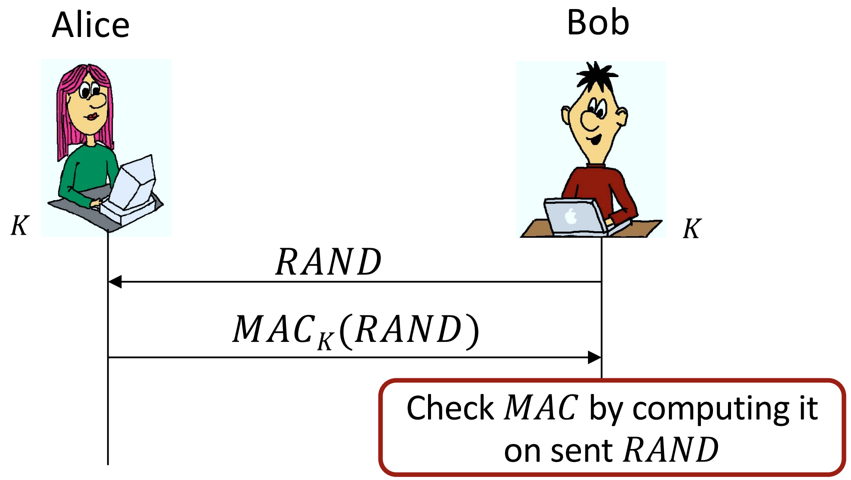
Idea:

- **B** generates a fresh challenge
  - E.g., a random number or a time stamp (implicit challenge)
- **A** proofs its identity by computing a response that
  - Depends on the challenge and a secret
  - Secret can be a secret key shared with B, a private key of A,…

Response Calculation must guarantee that the objectives hold

Example Building Bocks for Unilateral Entity Authentication based on shared key K



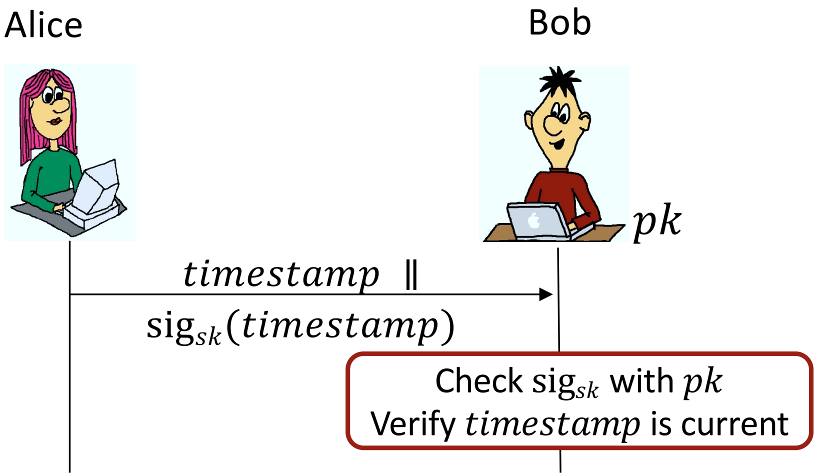


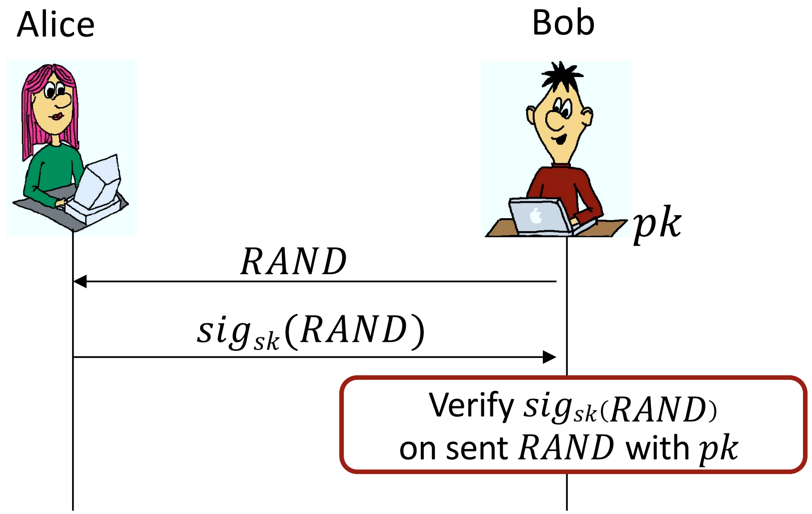
- Alice computes a MAC on timestamp
- Sends timestamp and MAC to Bob
- Bob verifies MAC by computing MAC on received timestamp and comparing it to received MAC
- Bob checks if $timestamp$ is in an acceptable range around Bob's current time

- Bob selects a random number RAND as challenge and sends it to Alice
- Alice computes a MAC on RAND using K
- Bob verifies that the received MAC corresponds to the one he computes using RAND as input

Example Building Bocks for Unilateral Entity Authentication

Unilateral authentication of A to B based on a private key sk of Alice assuming Bob knows Alice's public key pk



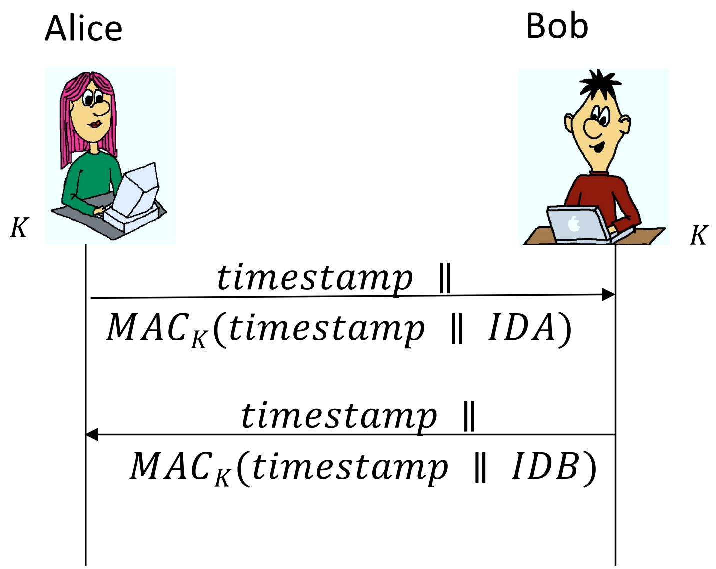


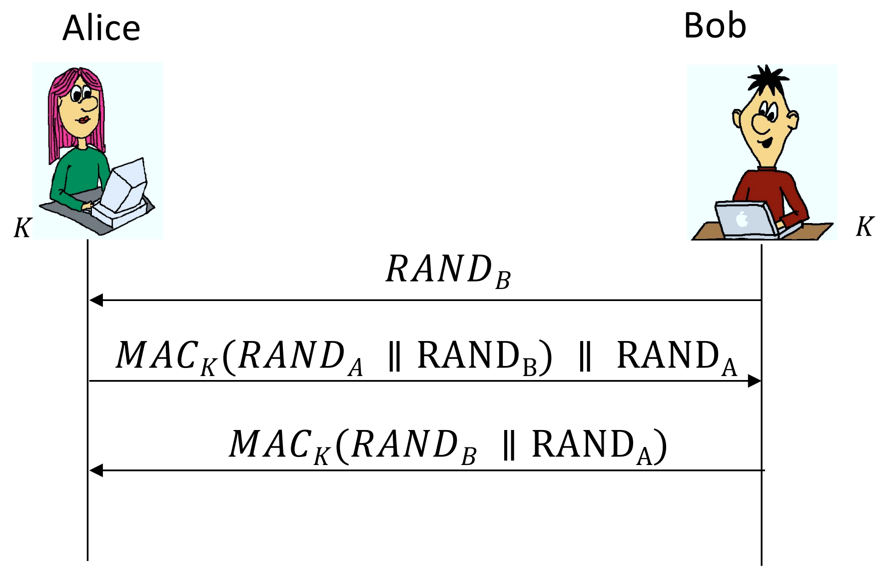
- Alice computes a signature on the current timestamp (implicit challenge) using sk
- Sends the $timestamp$ and the signature to Bob
- Bob verifies signature with pk and checks if $timestamp$ is in an acceptable range

- Bob selects a random number $RAND$ as challenge and sends it to Alice
- Alice computes a signature on $RAND$
- Bob verifies that the received signature is a signature on the sent $RAND$

Example Building Bocks for Mutual Entity Authentication

- **Mutual authentication of A to B and B to A based on a shared secret key K**

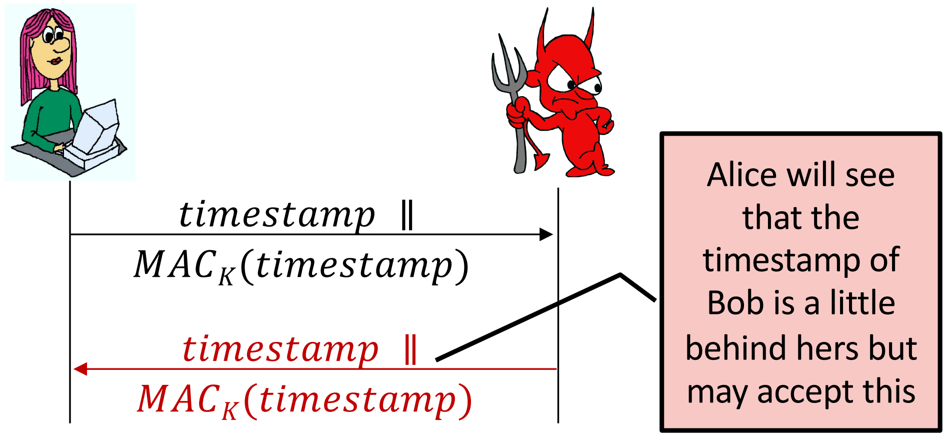


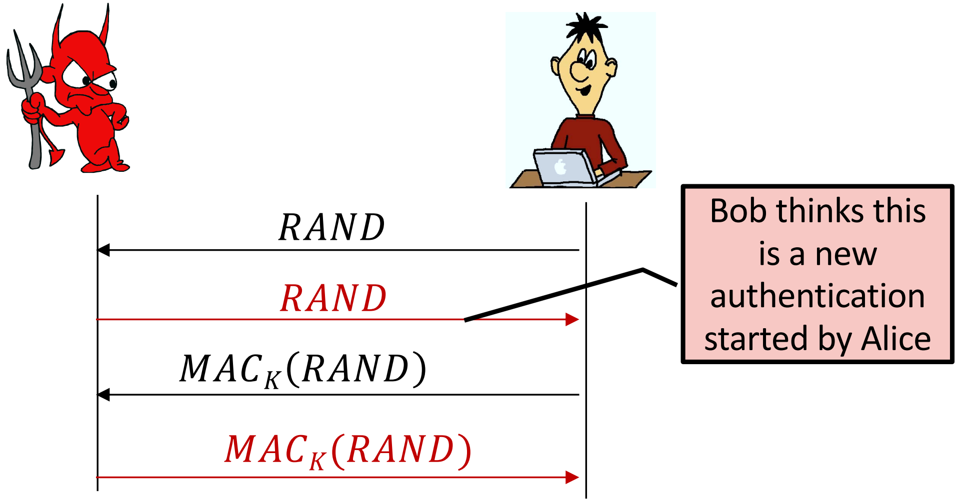


Does work with signatures just as well

Example for Insecure Building Blocks for Mutual Authentication

Simply combining the building blocks for unilateral authentication MAY NOT be SECURE



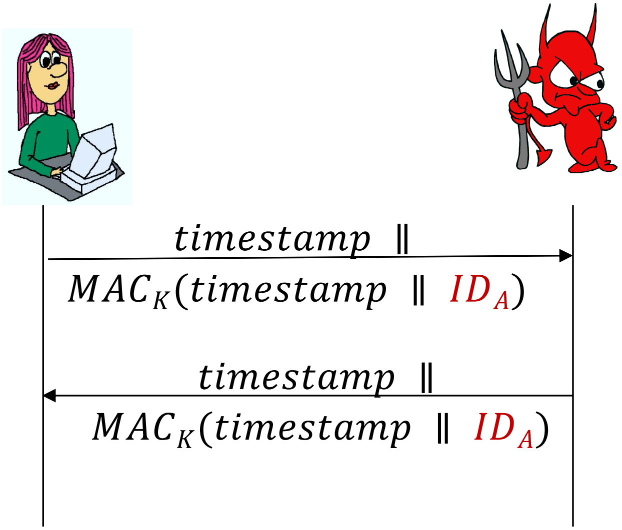


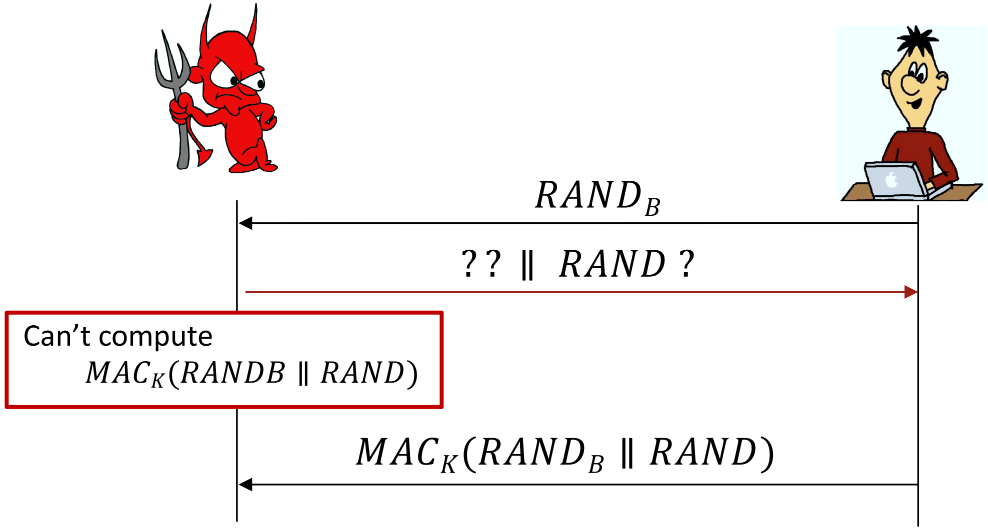
- Attacker could claim to be Bob and just reflect Alice's message to Alice
- Not impersonation resistant
- Need messages of Alice and Bob to be different

- Attacker could start a second run of the protocol by reflecting RAND back to Bob
- Wait for Bob's reply
- Then reflect the MAC computed by Bob back to Bob

Protection against Reflection Attacks

- **Making A and B compute MACs on different messages, where each message contains input controlled by the other part protects these building blocks from reflection attacks**





- Attacker can only reflect message including Alice's ID which will be detected by Alice
- Attacker can only reflect with the random number in Bob's order not in the order expected from Alice

Overview

- **Building Blocks for Entity Authentication**
  - Definition of Entity Authentication
  - MAC-based authentication
  - Signature-based authentication

- **Key Distribution with trusted Third Parties**
  - Key Distribution Centers
  - Certificates and Public Key Infrastructures

- **Authenticated Session Key Establishment**
  - Definitions around session key establishment
  - Authenticated Diffie Hellman variants
  - Session key establishment w-o DH
  - Session Key derivation principles

- **Password-based authentication**
  - Password-based user authentication
  - Password-based authenticated key establishment
  - Dictionary attacks on password-based authentication

Entity Authentication Alone is useless!

- Authentication exchange typically only guarantees that one specific message originates from a particular entity
- If hash of previously sent messages is included, these can be authenticated as well
- But: what about future messages exchanged? And what about encryption?

- **Could keep signing messages if signatures are used**
  - Very inefficient
- **Could keep computing MACs with key K on all messages**
  - Key K would be used repeatedly on lots of traffic

Solution: Session Keys

- Establish new session keys for integrity protection and encryption
- Thus, create independence across communication sessions
- Limit amount of data protected under the same key

Session Key Establishment Protocols

A session key establishment protocol is a protocol

- that establishes a shared secret key between two parties

There are two types of key establishment protocols

- **Key transport protocols**
  - Key generated by one party, securely transported to the other party
- **Key agreement protocols**
  - shared key is derived from input of bother parties, e.g. like in the Diffie-Hellman key agreement protocol

Examples

- **Simple key transport protocol**
  - Assume A and B share a long-term key K
  - A selects a session key SK
  - Computes $E_K(SK)$ and sends it to B
  - B decrypts $E_K(SK)$ with K and thus obtains SK
- **Diffie-Hellman key agreement (Chapter4)**
  - Each party selects a random private value
  - Computes a public value based on private one
  - Parties exchange the public values
  - Each computes that key as function of own private and other party's public value

Objectives of Key Establishment Protocols

Authenticated key Establishment

- **Entity authentication (see above)**
- **Implicit key authentication**: a party is assured that no other party but a particular second party may gain access to the established key

Explicit key authentication

- Implicit key authentication
- **Key confirmation**: a party is assured that a second party has possession of the established key

Additional Objectives

- **Key freshness**: a party is assured that the key is newly generated and not a replayed old key
- **Perfect forward secrecy**: a future compromise of long-term keys does not compromise past session keys
- **Protection against known-key attacks**: the compromise of a past session key does not allow
  - a passive adversary to compromise future session keys
  - an active attacker to impersonate a party in the future

The objectives can hold for none, only one or both parties

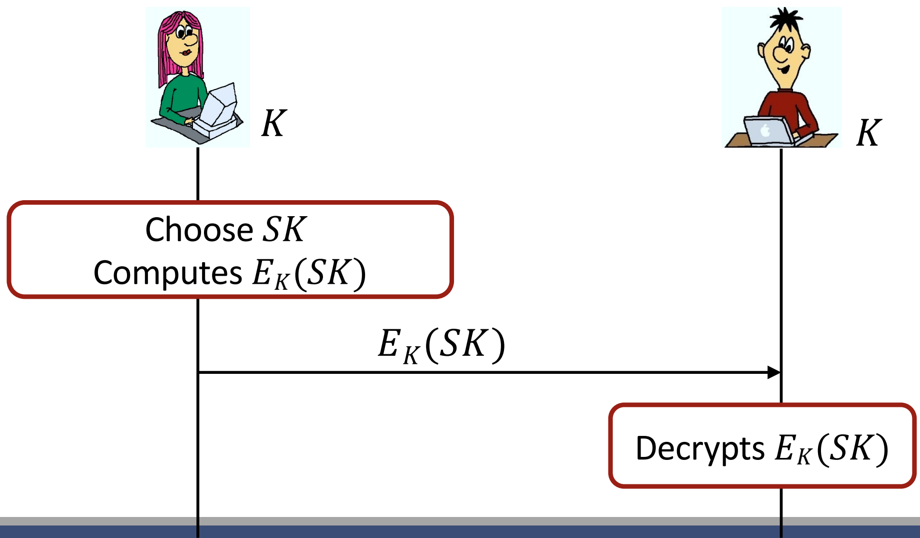
Efficiency Considerations

- **When analyzing the efficiency of protocols, we consider**
  - Number of messages exchanged between parties
  - Bandwidth required by the messages (total number of bits transmitted)
  - Complexity of computations that need to be carried out by the parties
  - Possibility for pre-computation to reduce the online load during protocol execution

Example: Simple key transport protocol

Simple key transport protocol

- Assume A and B share a long-term key K
- A selects a session key SK
- Computes $E_K(SK)$ and sends it to B
- B decrypts $E_K(SK)$ with K and thus obtains SK



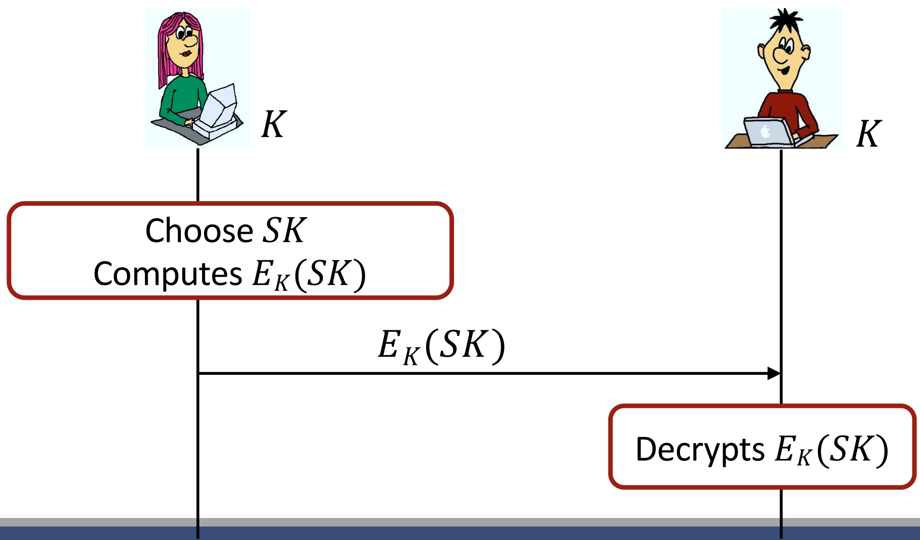
Properties

- Implicit key authentication
  - Yes, from both parties' point of view
- Key freshness
  - Yes, from A's point of view
  - No from B's point of view
- Perfect forward secrecy
  - No
- Protection against known keys
  - Past session keys have no influence on new future ones
- Authenticated key establishment
  - No! No entity authentication (replay possible)

Example: Simple key transport protocol

Simple key transport protocol

- Assume A and B share a long-term key K
- A selects a session key SK
- Computes $E_K(SK)$ and sends it to B
- B decrypts $E_K(SK)$ with K and thus obtains SK

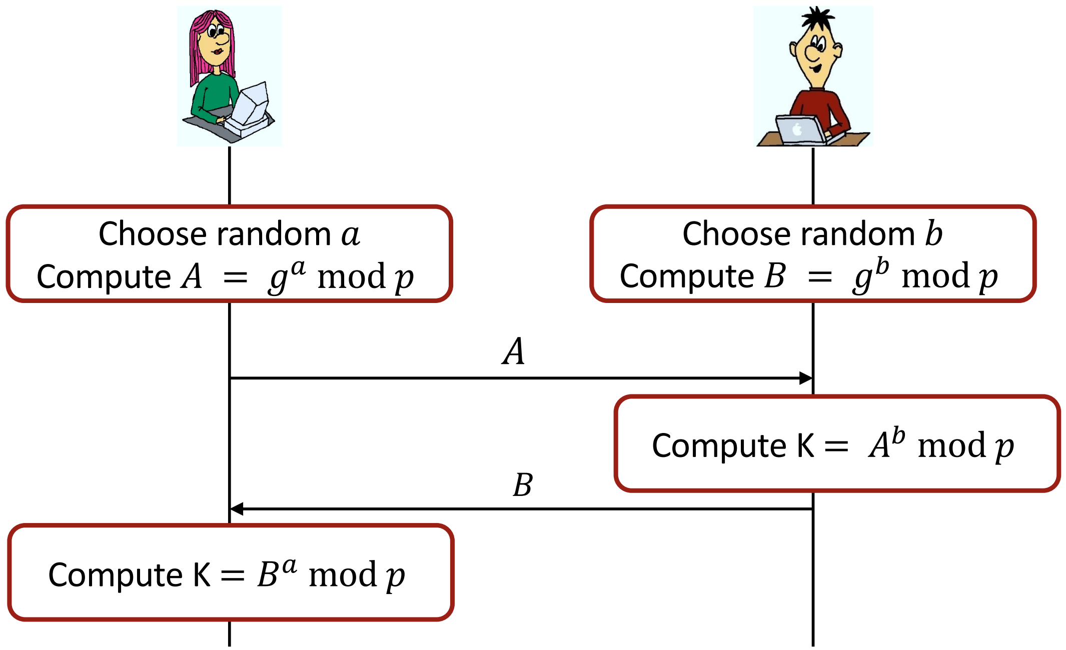


Properties

- Implicit key authentication
  - Yes, from both parties' point of view
- Key freshness
  - Yes, from A's point of view
  - No from B's point of view
- Perfect forward secrecy
  - No
- Protection against known keys
  - Past session keys have no influence on new ones
- Authenticated key establishment
  - No! No entity authentication (replay possible)

Diffie-Hellman Key Agreement

- Implicit key authentication
  - No
- Key freshness
  - Yes, from both parties' point of view
- Perfect forward secrecy
  - Yes, future keys completely independent
- Protection against known keys
  - Past session keys have no influence on future ones
- Authenticated key establishment
  - No! No entity authentication (replay possible), no implicit key authentication

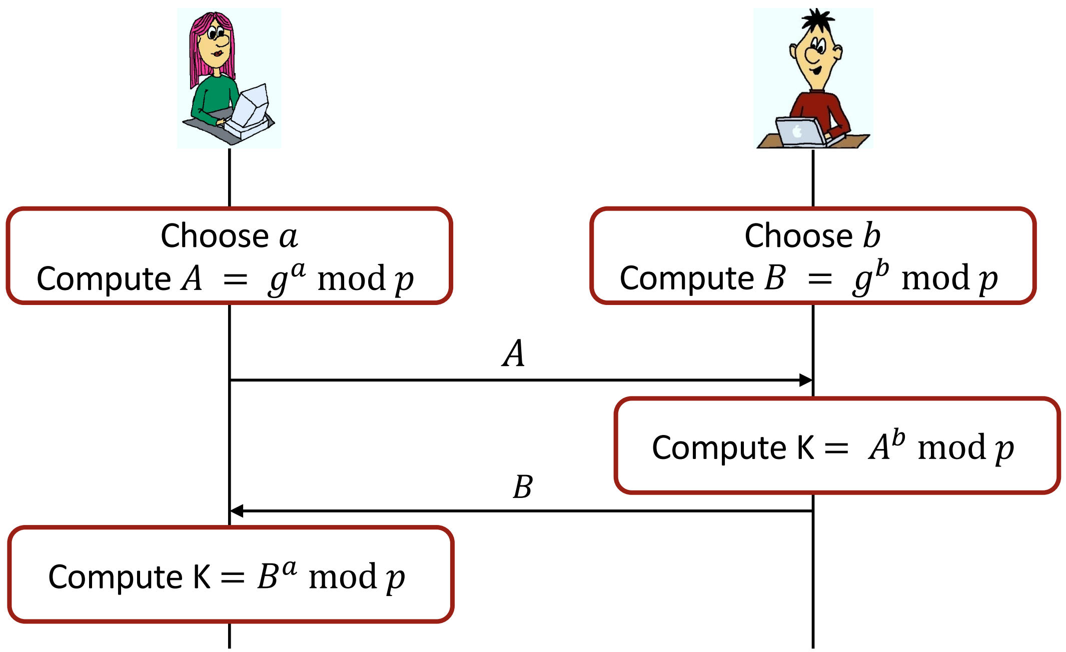


As $A^b \bmod p = g^{ab} = g^{ba} = B^a \bmod p$

Alice and Bob now share the secret key K = g^{ab}

Diffie-Hellman Key Agreement

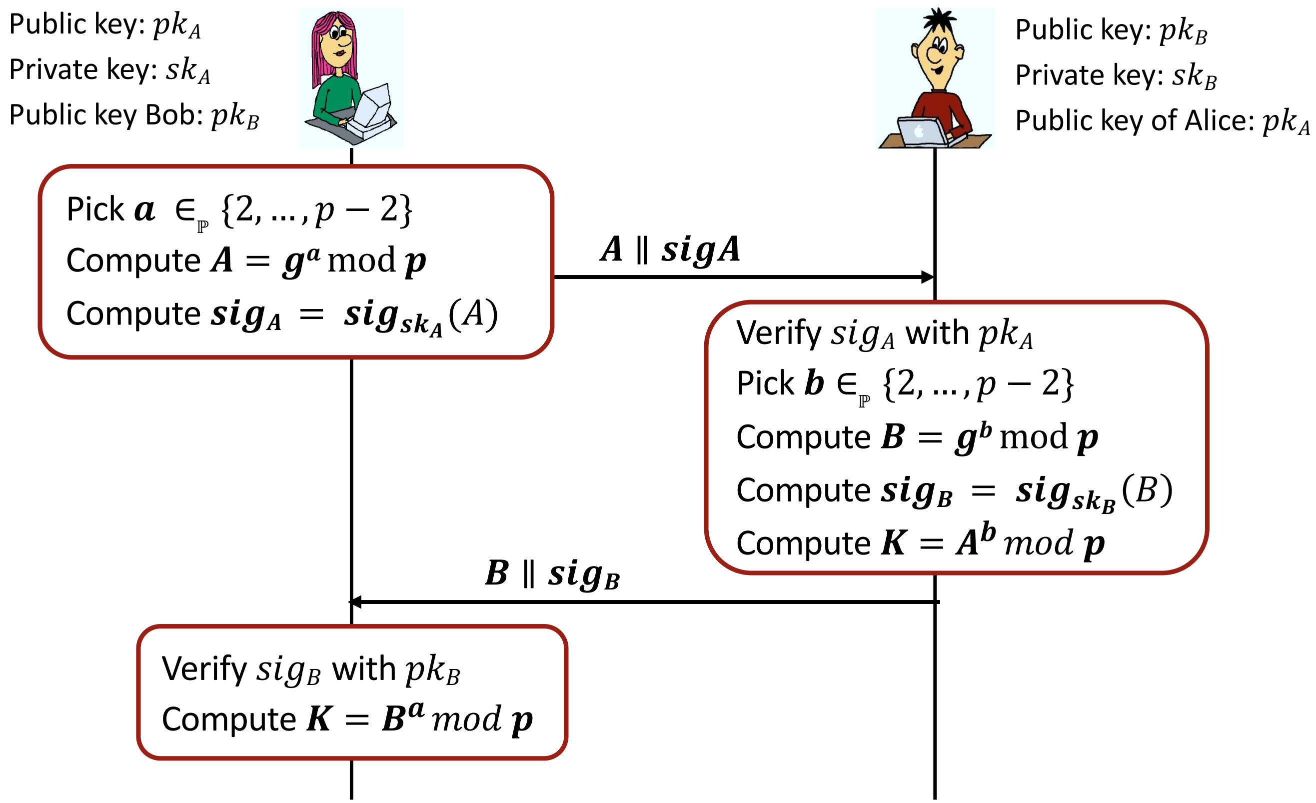
- Implicit key authentication
  - No
- Key freshness
  - Yes, from both parties' point of view
- Perfect forward secrecy
  - Yes, future keys completely independent
- Protection against known keys
  - Yes, past session keys have no influence on future ones
- Authenticated key establishment
  - No! No entity authentication (replay possible), no implicit key authentication



As $A^b \bmod p = g^{ab} = g^{ba} = B^a \bmod p$

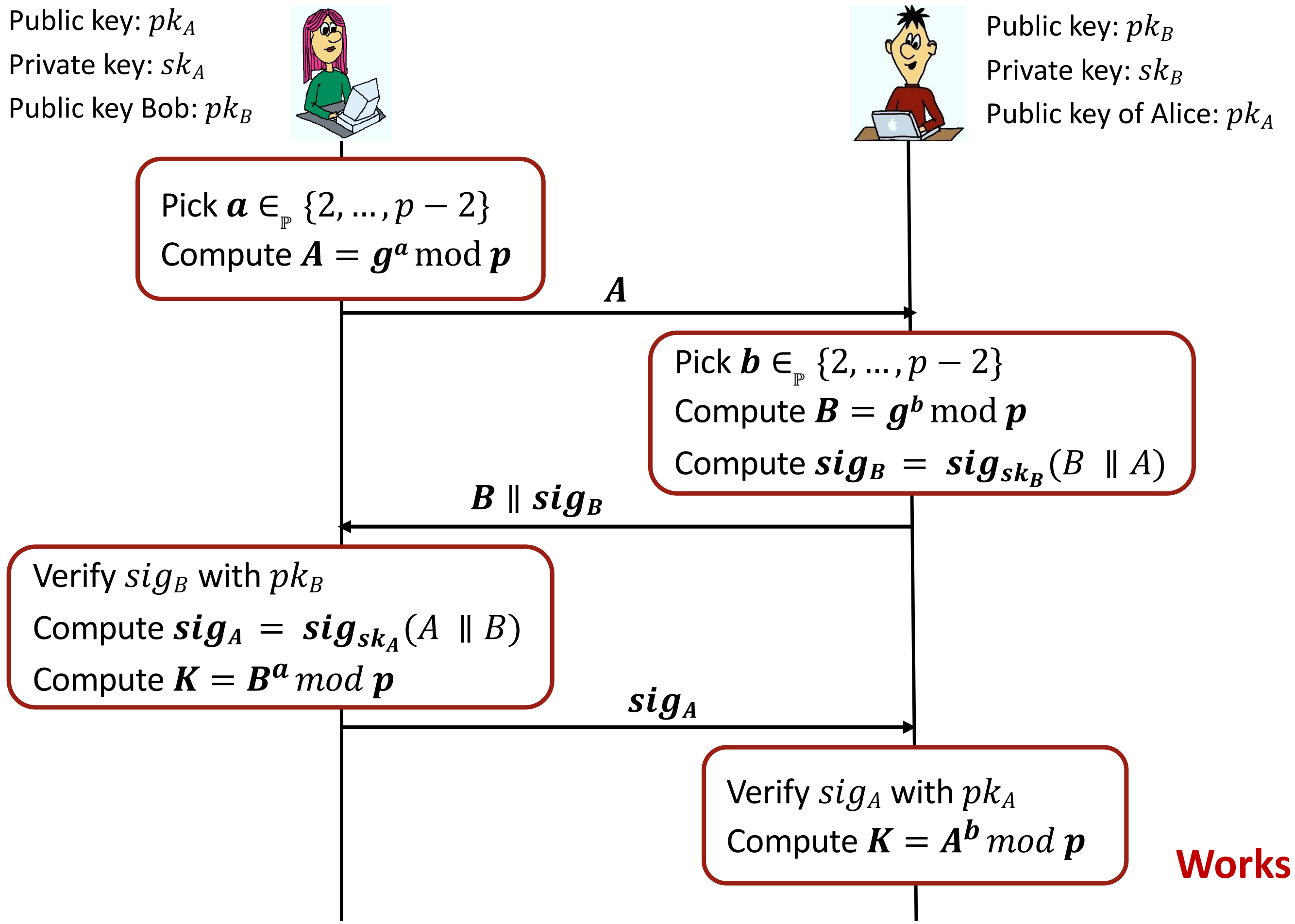
Alice and Bob now share the secret key K = g^{ab}

Diffie-Hellman Key Agreement with Implicit Key Authentication



- **Implicit key authentication**
  - Public DH value has been signed by the desired second party
  - Only that party (if any) will be able to compute K
- **But: no entity authentication**
  - Old messages could be replayed
  - Parties do not get guarantee that other party interacts right now

Authenticated Diffie-Hellman Key Agreement with Signatures

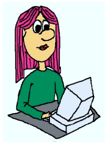


- **Mutual authentication between Alice and Bob**
  - See slide 8
  - A and B act as random values here
- **Implicit key authentication**
  - Alice is assured that B is from B so only Bob can compute K (and herself)
  - Same holds for Bob
- **Authenticated key agreement**

Works with a shared key and MACs as well!

Example Session Key Establishment without DH

Shared secret key K



Shared secret key K

Pick a random number $\boldsymbol{RAND_A}$

$\boldsymbol{RAND_A}$ →

Pick a random number $\boldsymbol{RAND_B}$
Compute
$\boldsymbol{SK} = HMAC_{\boldsymbol{K}}(RAND_A \parallel \mathrm{RAND_B})$

← $MAC_{SK}(RAND_{\mathrm{B}} \parallel RAND_{\mathrm{A}}) \parallel RAND_{\mathrm{B}}$

Compute
$\boldsymbol{SK} = HMAC_{\boldsymbol{K}}(RAND_A \parallel \mathrm{RAND_B})$
Verify MAC received

$MAC_{SK}(RANDA \parallel RAND_{\mathrm{B}})$ →

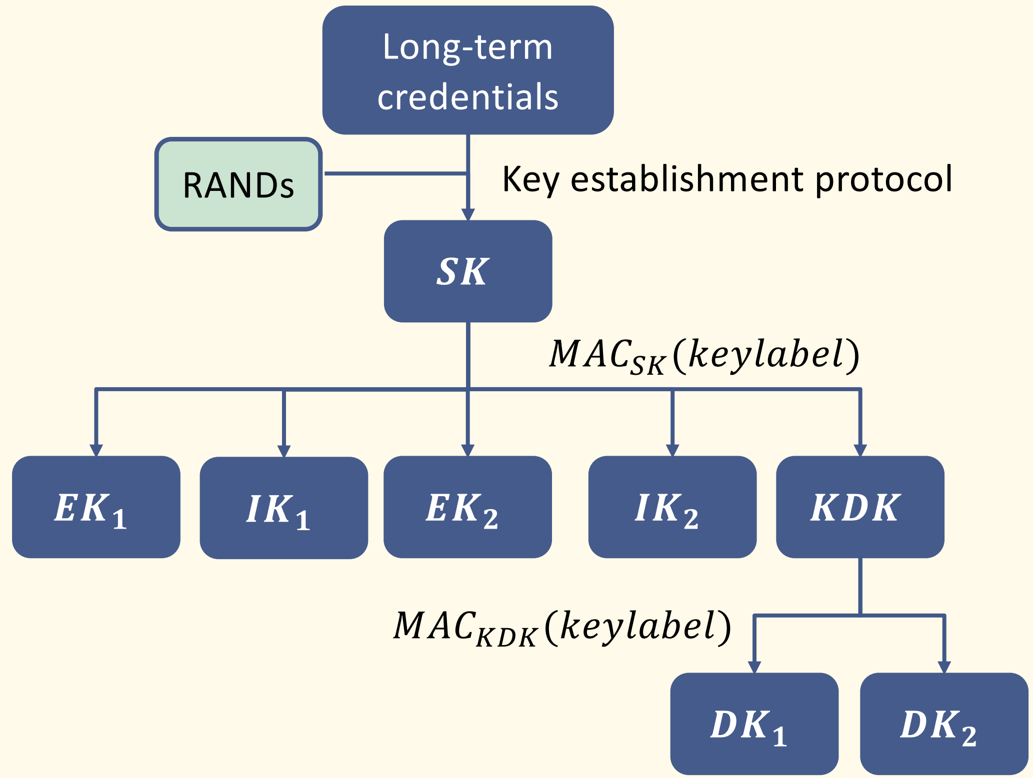
Verify MAC received

- Implicit key authentication
  - Yes! K required to compute SK
- Key freshness
  - Yes, for both parties
- Perfect forward secrecy
  - No! If K broken and exchange recorded, then SK broken
- Protection against known keys
  - Past session keys have no influence on future ones
- Authenticated key establishment
  - Yes!

Session Key Derivation: Key Hierarchies

- **Key establishment protocols**
  - establish a session key SK based on long term credentials and session specific random numbers
- **SK often used to derive additional keys, e.g.**
  - Integrity key and an encryption key
  - Different keys for different directions
  - A key derivation key for future derivations
- **Results in key hierarchy**
  - Key derivation should be efficient
  - A break of a lower layer key does not break higher layer keys or keys on the same layer

Example Hierarchy



Overview

- **Building Blocks for Entity Authentication**
  - Definition of Entity Authentication
  - MAC-based authentication
  - Signature-based authentication

- **Key Distribution with trusted Third Parties**
  - Key Distribution Centers
  - Certificates and Public Key Infrastructures

- **Authenticated Session Key Establishment**
  - Definitions around session key establishment
  - Authenticated Diffie Hellman variants
  - Session key establishment w-o DH
  - Session Key derivation principles

- **Password-based authentication**
  - Password-based user authentication
  - Password-based authenticated key establishment
  - Dictionary attacks on password-based authentication

Facilitating Key Distribution with Trusted Third Parties

Assumption so far: Alice and Bob

- Either already share a secret (long-term) key
- Or have an authentic copy of each other's public keys

Trusted Third Party

- Mediator to reduce the number of pre-installed keys required

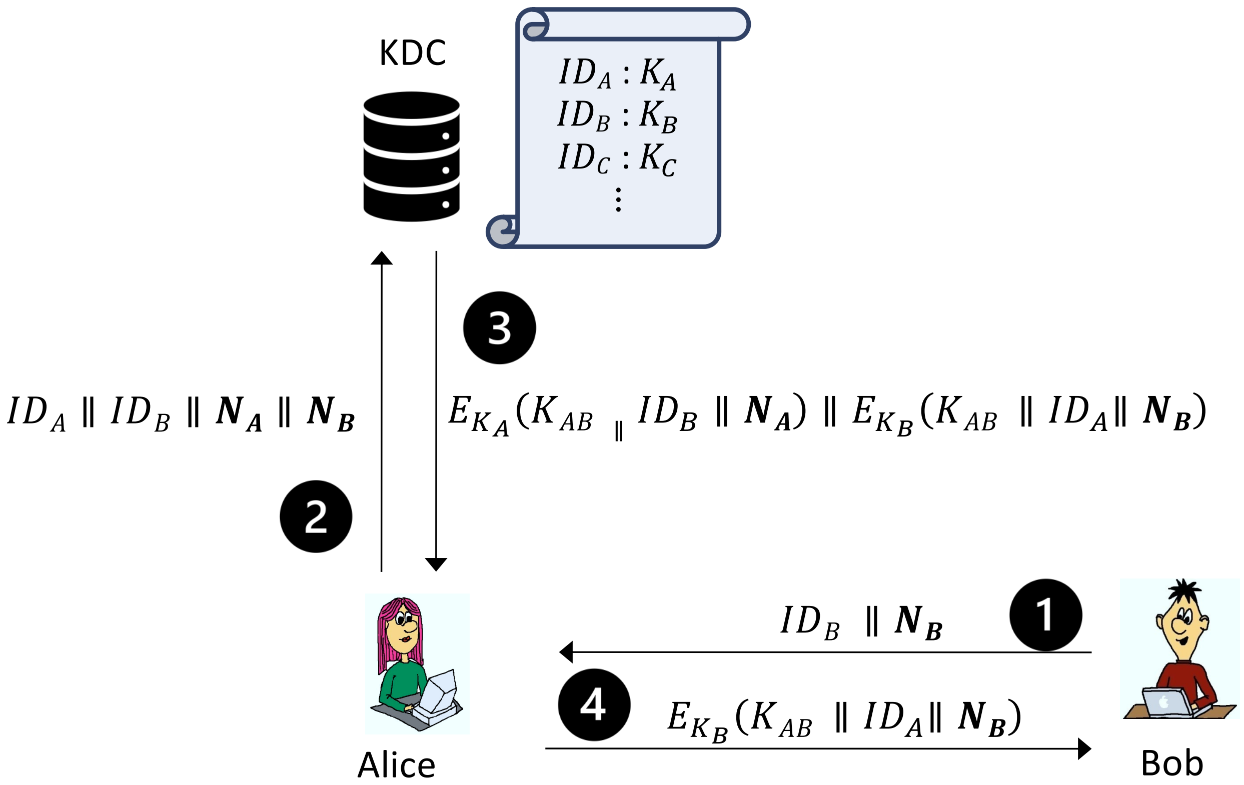
Symmetric Case: Key Distribution Centers

- Each client shares a **secret key** with the key distribution center
- The key distribution center helps to establish keys between its clients

Asymmetric Case: Certification Authorities

- Each client has the public key of a certification authority pre-installed
- The certification authority helps to distribute authentic copies of public keys

Example: Key Transport with a KDC



- KDC shares a long-term secret key K_A with Alice and K_B with Bob
- Upon request, KDC generates a session key K_{AB} for Alice and Bob
- E_K here stands for an AEAD encryption with K
- N_B and N_A authenticates KDC to Bob and Alice respectively
- Inclusion of ID_B in $E_{K_A}(K_{AB} \parallel ID_B \parallel \boldsymbol{NA})$ gives Alice implicit key authentication of K_{AB}
- Inclusion of ID_A in $E_{K_B}(K_{AB} \parallel ID_A \parallel \boldsymbol{NB})$ gives Bob implicit key authentication of K_{AB}
- No perfect forward secrecy, no key freshness, protection against known key attacks

Facilitating Key Distribution with Trusted Third Parties

Assumption so far: Alice and Bob

- Either already share a secret (long-term) key
- Or have an authentic copy of each other's public keys

Trusted Third Party

- Mediator to reduce the number of pre-installed keys required

Symmetric Case: Key Distribution Centers

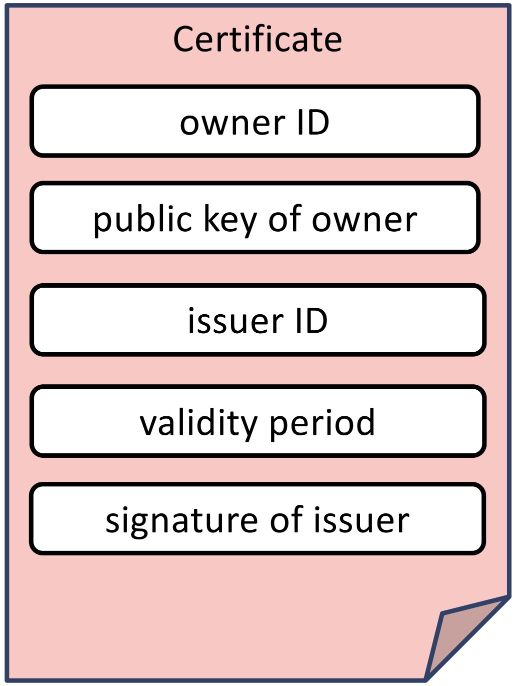
- Each client shares a **secret key** with the key distribution center
- The key distribution center helps to establish keys between its clients

Asymmetric Case: Certification Authorities

- Each client has the public key of a certification authority pre-installed
- The certification authority helps to distribute authentic copies of public keys

Certification Authorities and Public Key Infrastructures

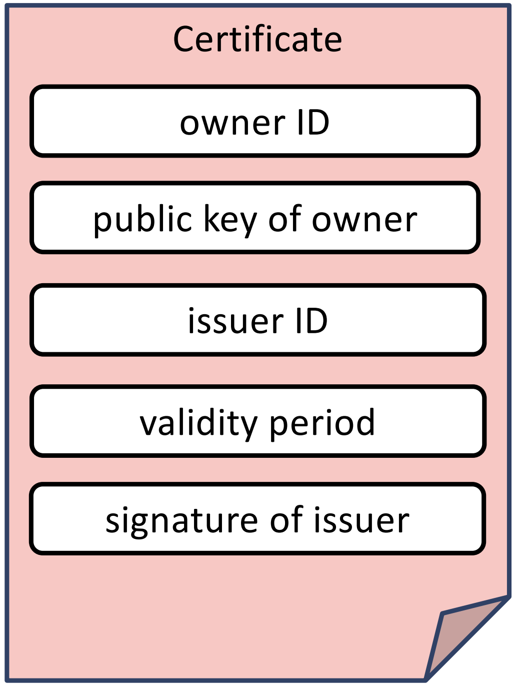
- **Certification Authority**
  - Sings a certificate for each of its clients
  - Certificate
    - owner ID: identifier of the owner of the public key
    - public key of owner
    - issuer ID: identifier for the CA that issued the certificate
    - Validity period: not before, until dates defining when this certificate becomes valid and when it expires
    - Signature of the issuing CA on all of the content of the certificate, binds public key to owner ID



- **Anyone in possession of the public key of the CA**
  - Can verify the **authenticity of the public key** of the owner

Certificate Verification

- **Anyone in possession of the public key of the CA**
  - Can verify the **authenticity of the public key** of the owner
- **Certificate verification entails**
  - checking the validity period of the certificate
  - checking that the owner ID is as expected
    - E.g., in the context of web does the domain name included as identifier in the certificate match the host name part of the URL of the visited website
  - checking the signature on the certificate with the public key of the issuer
  - checking the revocation status of the certificate



Certificate Revocation Approaches

Certificates may need to be revoked before they expire

- Due to stolen devices, precaution after malware infection,…
- Due to lost passwords unlocking private keys

Certificate revocation lists = CRLs

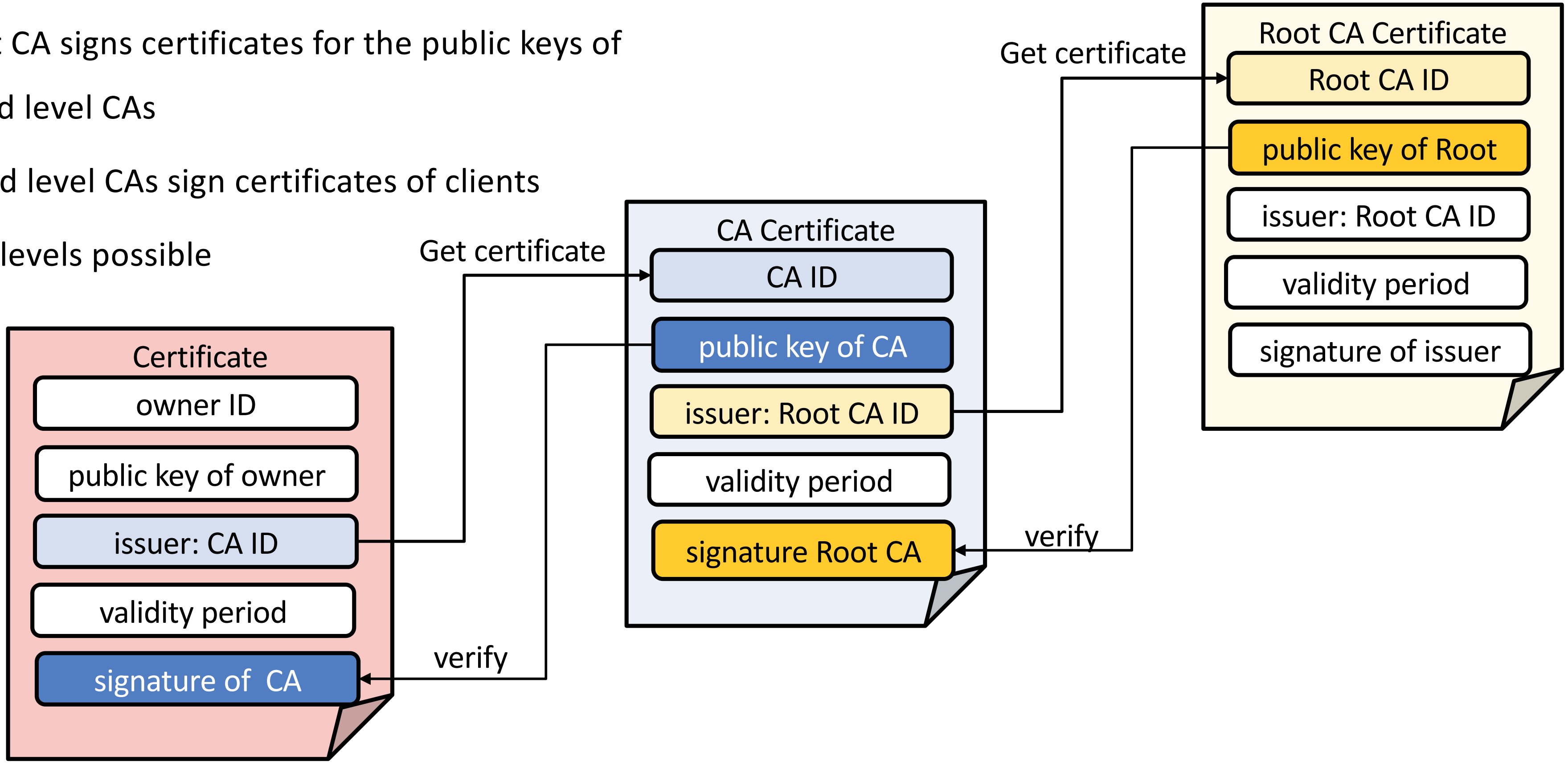
- Issuing CA periodically publishes a signed CRL
- CRL includes serial numbers of all revoked unexpired certificates
- Disadvantage: revocation only as timely as period used to publish CRLs

Online Certificate Status Protocol = OSCP

- Protocol to obtain immediate feedback on the revocation status of certificates
- Advantage: very timely revocation possible
- May add additional overhead and requires connectivity to the OSCP server

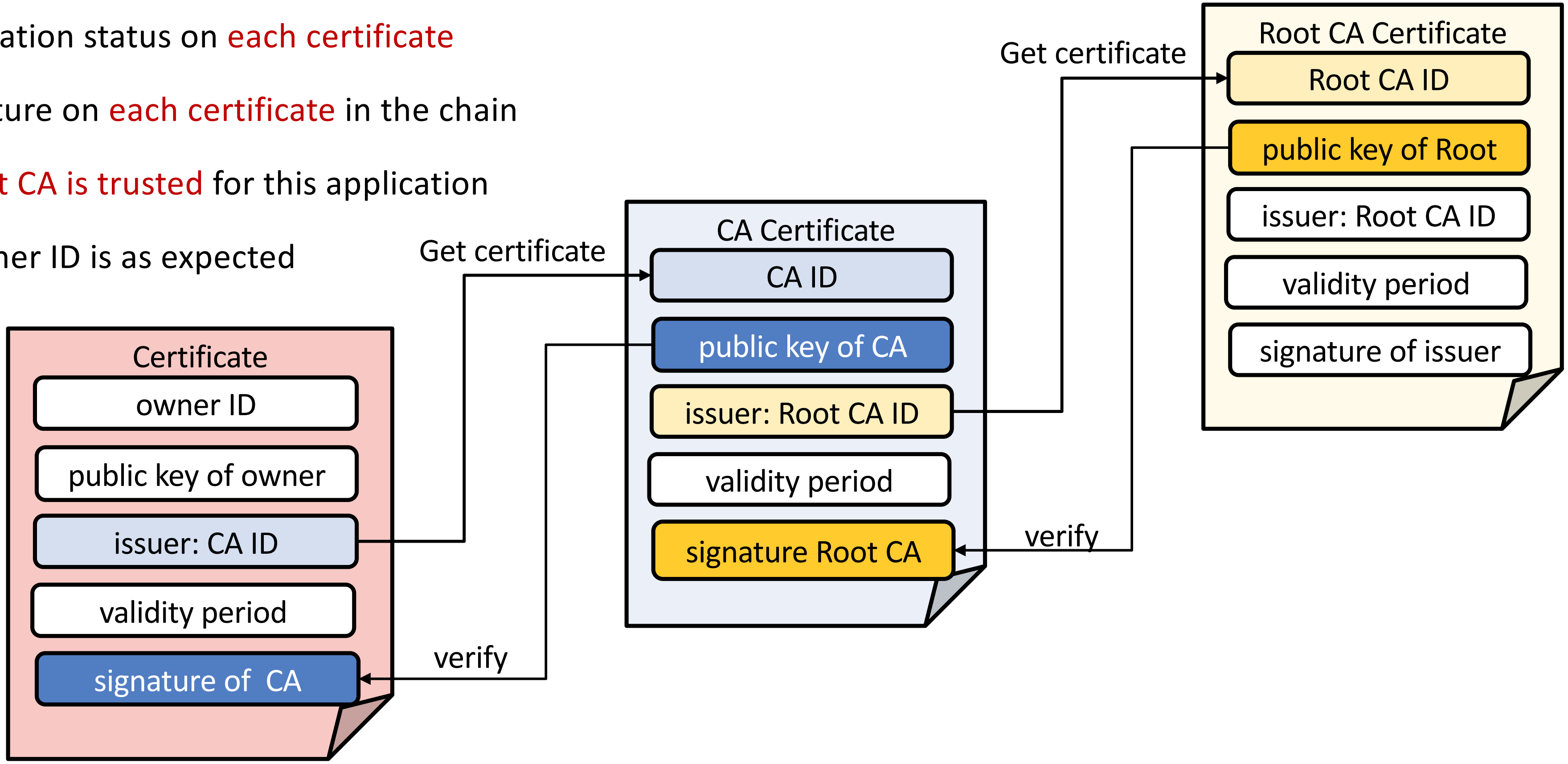
Chains of Certificates

- **Hierarchies of certification authorities**
  - A root CA signs certificates for the public keys of second level CAs
  - Second level CAs sign certificates of clients
  - More levels possible



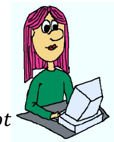
Verifying Chains of Certificates

- Check validity period of each certificate
- Check revocation status on each certificate
- Verify signature on each certificate in the chain
- Check if root CA is trusted for this application
- Check if owner ID is as expected



Example Secure Authenticated DH with Chain of Certificates

Public key: pk_A
Private key: sk_A
Public key of root CA: pk_{root}



Public key: pk_B
Private key: sk_B
Public key of root CA: pk_{root}

Pick $\boldsymbol{a} \in_{\mathbb{R}} \{2, \dots, p-2\}$
Compute $\boldsymbol{A} = \boldsymbol{g^a} \bmod \boldsymbol{p}$

$\boldsymbol{A}$ →

Pick $\boldsymbol{b} \in_{\mathbb{R}} \{2, \dots, p-2\}$
Compute $\boldsymbol{B} = \boldsymbol{g^b} \bmod \boldsymbol{p}$
Compute $\boldsymbol{sig_B} = \boldsymbol{sig_{sk_B}}(B \parallel A)$

← $\boldsymbol{B} \parallel \boldsymbol{sig_B} \parallel \boldsymbol{cert_B}$

Verify chain of certificates $\boldsymbol{cert_B}$
Verify sig_B with pk_B extracted from B's certificate
Compute $\boldsymbol{sig_A} = \boldsymbol{sig_{sk_A}}(A \parallel B)$
Compute $\boldsymbol{K} = \boldsymbol{B^a} \bmod \boldsymbol{p}$

$\boldsymbol{sig_A} \parallel \boldsymbol{cert_A}$ →

Verify chain of certificates $\boldsymbol{cert_A}$
Verify sig_A with pk_A extracted from A's certificate
Verify sig_A with pk_A
Compute $\boldsymbol{K} = \boldsymbol{A^b} \bmod \boldsymbol{p}$

- $cert_B$ / $cert_A$: chain of certificates starting with a certificate for A / B, where the last one is the root certificate

Overview

- **Building Blocks for Entity Authentication**
  - Definition of Entity Authentication
  - MAC-based authentication
  - Signature-based authentication
- **Key Distribution with trusted Third Parties**
  - Key Distribution Centers
  - Certificates and Public Key Infrastructures
- **Authenticated Session Key Establishment**
  - Definitions around session key establishment
  - Authenticated Diffie Hellman variants
  - Session key establishment w-o DH
  - Session Key derivation principles
- **Password-based authentication**
  - Password-based user authentication
  - Password-based authenticated key establishment
  - Dictionary attacks on password-based authentication

Password-based Authentication

Three main flavors used in practice

- Certificate-based server authentication
- Password-based user authentication
- Vulnerable to dictionary attacks if password file stolen

Used, e.g., in HTTPs

- MAC-based authenticated key exchange
- MAC-key derived from password
- Vulnerable to dictionary attacks

Used, e.g., in 4-Way-Handshake in WPA2 WLAN

- Password-Authenticated Diffie Hellman
- Protected against Dictionary attacks
- Same (one-time) password entered on both devices

Used, e.g., Secure Authentication of Equals in WPA3

Password-based Authentication

Three main flavors used in practice

Advance ITSec Lecture

- Certificate-based server authentication
- Password-based user authentication
- Vulnerable to dictionary attacks if password file stolen

Used, e.g., in HTTPs

- MAC-based authenticated key exchange
- MAC-key derived from password
- Vulnerable to dictionary attacks

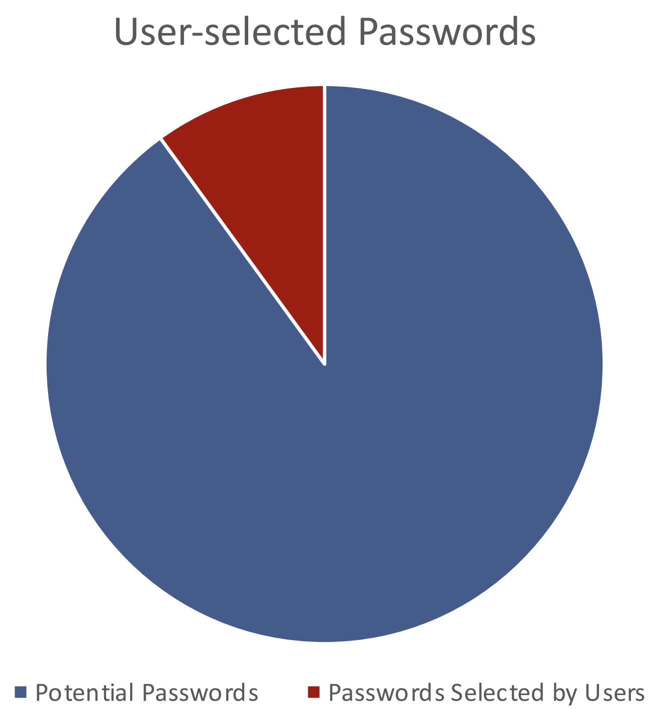
Used, e.g., in 4-Way-Handshake in WPA2 WLAN

- Password-Authe[nticated] Diffie Hellman
- Protected against Dictionary attacks
- Same password entered on both devices

Used, e.g., Secure Authentication of Equals in WPA3

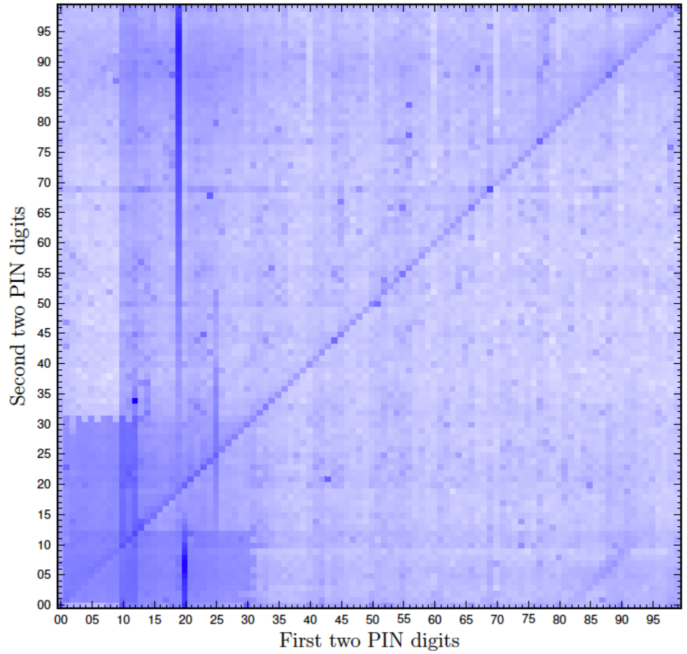
Password Length and Bit-Equivalence

- **Assume users can chose n character passwords**
  - small letters = 26 and capital letters = 26
  - numbers = 10, special characters except for space = 32
- **Then there are 94^n theoretically possible passwords**
  - n = 8 ⇒ ≈ 2^{52} possible passwords ≙ random secret key of 52 bit
  - n = 16 ⇒ ≈ 2^{104} possible passwords ≙ random secret key of 104 bit
- **Users tend NOT to select passwords randomly!**
  - Mainly because they cannot remember random passwords longer than 8 characters
  - And on average only one of these



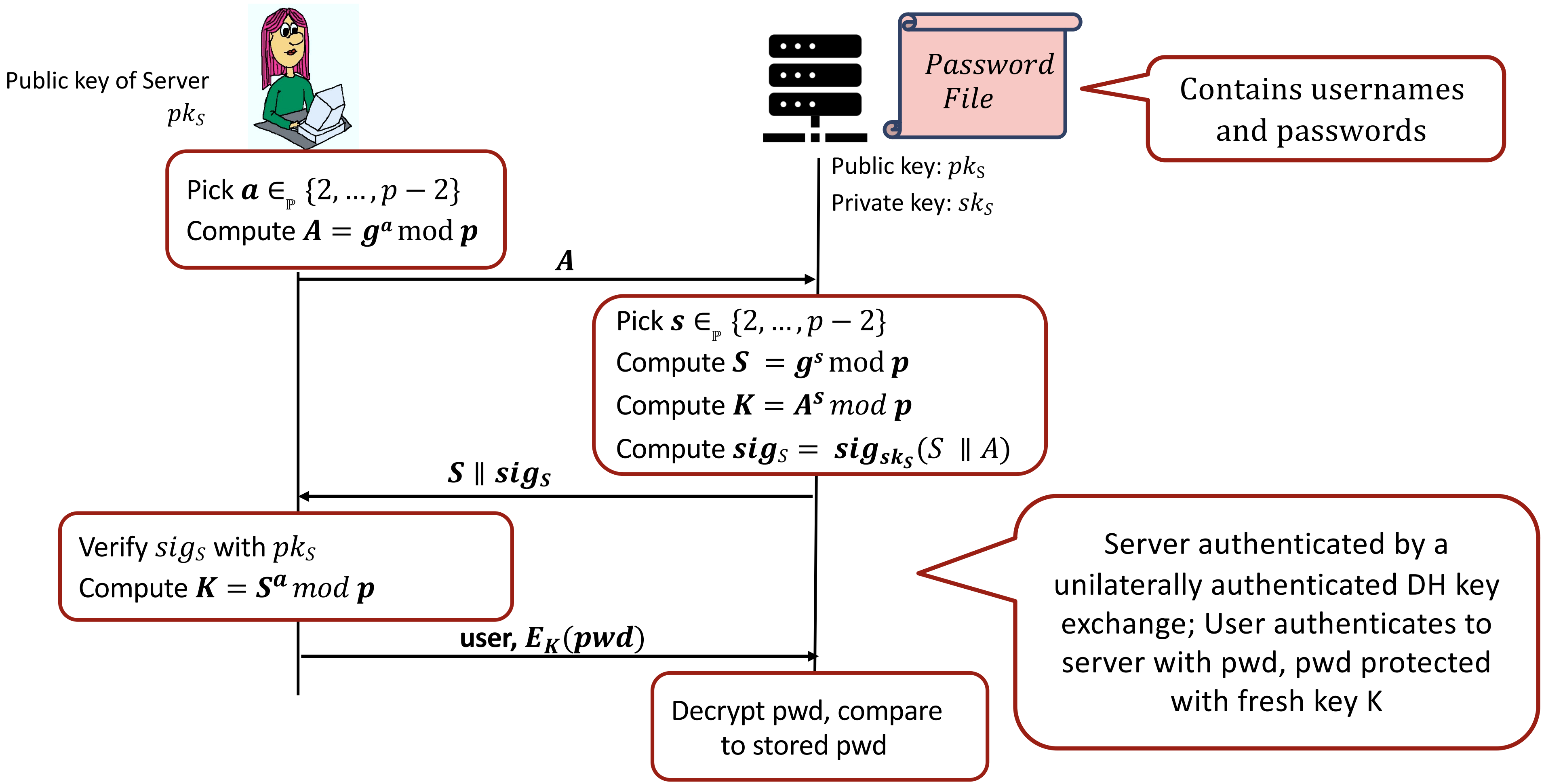
Classic Example: User's self-selected Banking PINs 2012

- **Distribution of 4-Digit PINs in a data base of 32 Million Banking PINS**
- **Enforcing rules on the password selection reduces the overall number of possible passwords**
  - E.g., if 8 characters are used and at least one of them needs to be an upper case letter, one a lower case letter one a number and one a special character
  - Longer passwords required



- **General recommendation**
  - Use random passwords and a password manager

Password-based User and Certificate-based Server Authentication



Storing Passwords in Password Files (1)

In the clear?

- If attacker gains access to the file, break is immediate

User	pwd
Alice	D^6as$%kjahG
Bob	(*&)A8a;sdifh

Encrypted?

- No immediate access
- But: encryption key needs to be stored somewhere
- Decryption adds overhead

User	pwd	$E_K(pwd)$
Alice	D^6as$%kjahG	Svl0EKlmp76XcePiC+wL7g
Bob	(*&)A8a;sdifh	1YE/i6MU4lBEnmbq/Wn1Zw

Key
a57987a344d32336

Storing Passwords in Password Files (2)

Store $\mathrm{h}(pwd)$ using a cryptographic hash function

- Attacker only learns hashes from file
- Cannot compute pre-images of the hashes
- But: what if multiple users use same pwd?

Better: store random salt and $\mathrm{h}(pwd \parallel salt)$

- Now users using the same passwords will have different hashes

User	pwd	salt	$SHA256$
Alice	D^6as$%kjahG		c25559cad0aca1566d4ba7609759e2de824c8af9e1e0b27891e99ac495e77877
Bob	(*&)A8a;sdifh		f69f1260b38daf282d8d729df34e40c0bdf0fb634f72fe7c17b09054d96c5724
Clare	(*&)A8a;sdifh		f69f1260b38daf282d8d729df34e40c0bdf0fb634f72fe7c17b09054d96c5724
Alice	D^6as$%kjahG	(*daw	3bcc5a93e5510780f3ce13b8f673758cee1e246963be321ced2d6f2d74054558
Bob	(*&)A8a;sdifh	&OGa8	373d0dd007c4409bdc5a05e6174e5322e88cc16d736d71c99a8876f01c70a9d9
Clare	(*&)A8a;sdifh	6YY34	5ee7d56e09d86f7d262fc0d68f27861644252c1dbd80cb59bbd6cedf6c080831

Dictionary Attacks on Password Files

- **Dictionary**
  - List of commonly used passwords
- **Dictionary attack**
  - Try out all passwords in the dictionary

Attack on a stolen password files w/o salts

- Pre-compute $\mathbf{h}(\boldsymbol{pwd})$ for any pwd in the dictionary
- Compare computed hashes with stored ones

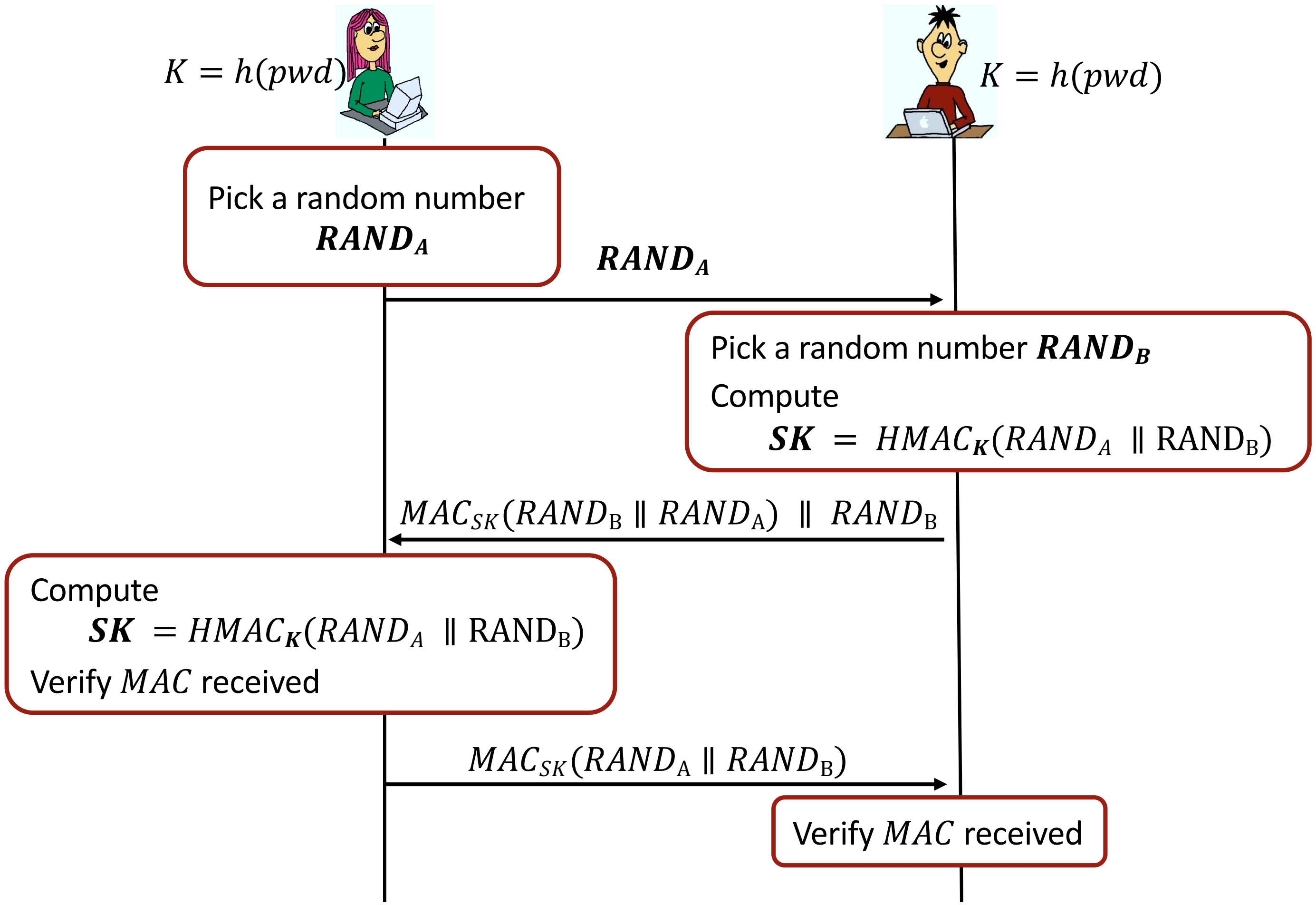
Needs to be done only once

Attack on a stolen password file with salts

- Compute $\mathbf{h}(\boldsymbol{pwd} \parallel \boldsymbol{salt})$ for any $salt$ in the password file and any pwd in the dictionary
- Compare computed hashes with stored ones

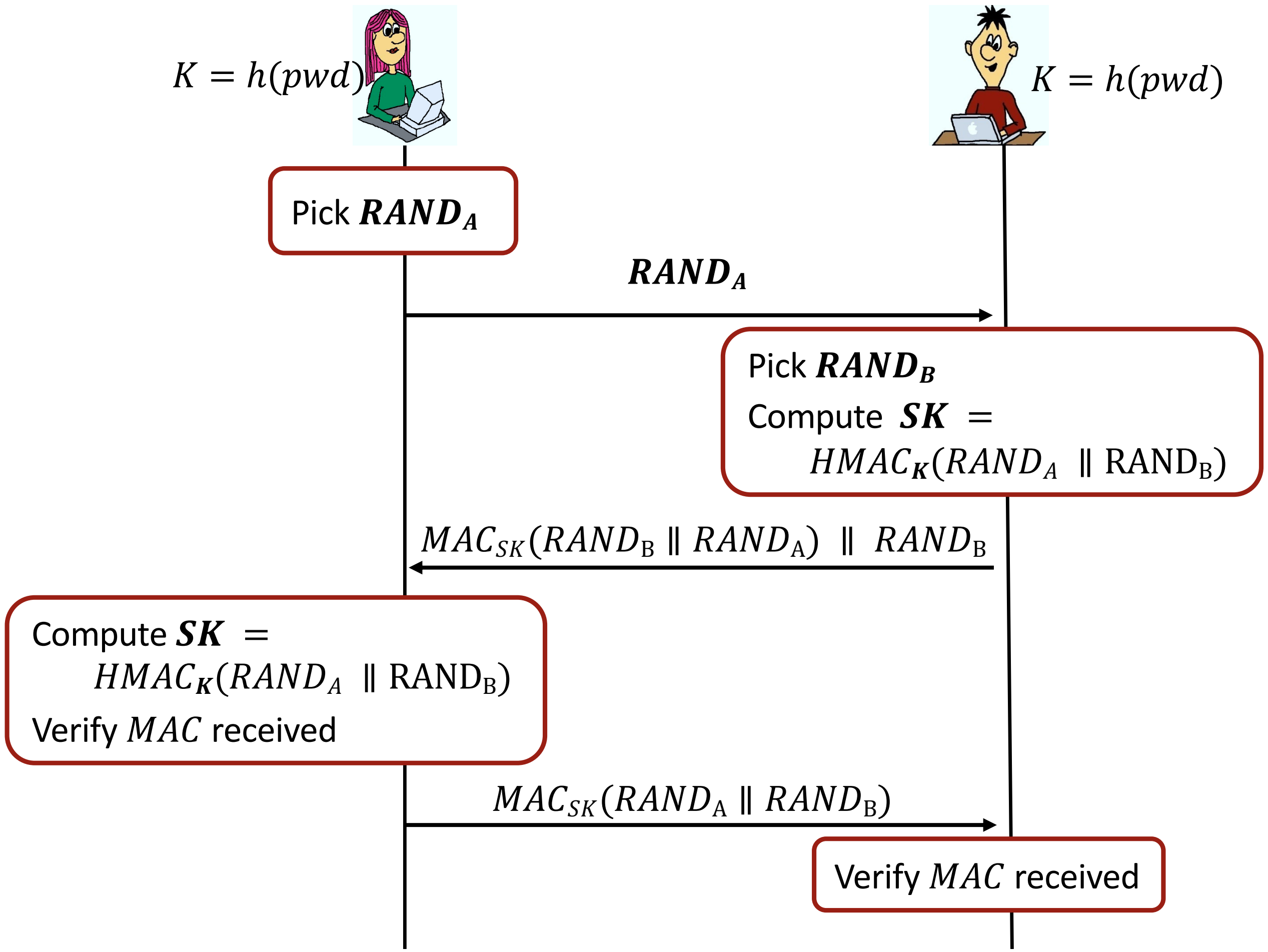
Salts are pwd-file specific

Authentication and Key Agreement with Password-Generated MAC Keys



- **Secret key generated by a shared password**
- **Note that the key is only as strong as the password**
  - K will be 128 bit but will be as easily guessable as the pwd

Dictionary Attack on Password-Authenticated Key Agreement



- **Record the message flow**
  - $RAND_A$, $RAND_B$
  - $MAC_{SK}(RAND_B \parallel RAND_A)$
- **For pwd in the dictionary**
  - compute $\boldsymbol{K = h(pwd)}$
  - compute **SK** from recorded RANDs
  - Check if

    $MAC_{\boldsymbol{SK}}(RAND_B \parallel RAND_A)$ =

    $MAC_{SK}(RAND_B \parallel RAND_A)$ recorded
  - If yes: **pwd** = pwd
  - Else: try next **pwd** in dictionary

Summary

- **Entity authentication requires**
  - an unforgeable proof that the other entity is active in the current protocol
  - session key establishment
    - Ensures continuous authentication of the authenticated entity
- **Entity authentication can be**
  - unilateral or mutual
  - be based on
    - secret keys using message authentication codes
    - or public/private key pairs
- **Key Establishment protocols**
  - can be key agreement or key transport protocols

Summary

Potential properties of key establishment protocols

- entity authentication
- implicit key authentication
- key confirmation
- key freshness
- perfect forward secrecy
- protection against known key attacks

entity authentication + implicit key authentication → authenticated key establishment

implicit key authentication + key confirmation → explicit key authentication

Summary

- **Trusted third parties can help to**
  - reduce the amount of pre-stored keys that need to be exchanged
  - Key distribution centers are TTPs that
    - help their clients establish symmetric keys
  - CAs are TTP that
    - help to distribute authentic copies of their clients' public keys
- **End-users are often authenticated with the help of passwords**
  - The larger the alphabet and the longer the password the stronger the password is
- **End-users tend to pick specific passwords more often than others**
  - Can compile a dictionary of often picked passwords

References

- **W. Stallings, Cryptography and Network Security: Principles and Practice, 8th edition, Pearson 2022**
  - Chapter 15: Cryptographic Key Management and Distribution
  - Chapter 16: User Authentication
- RFC 5869 HMAC-based Extract-and-Expand Key Derivation Function (HKDF)