(Applied) Cryptography

Week #12: Key Exchange Protocols

Manuel Barbosa, mbb@fc.up.pt

MSI/MCC/MERSI – 2022/2023

DCC-FCUP

Part #1: Key Management

How many keys and what are they used for?

User/application data is protected using symmetric cryptography:

- best speed and bandwidth costs for large data

There are two kinds of cryptographic keys:

- long-term or wrapping keys: they protect other (session) keys
- *session* or *data encapsulation* keys: they protect data

How many keys and what are they used for?

User/application data is protected using symmetric cryptography:

- best speed and bandwidth costs for large data

There are two kinds of cryptographic keys:

- long-term or wrapping keys: they protect other (session) keys
- *session* or *data encapsulation* keys: they protect data

Long-term keys:

- can be asymmetric, symmetric, or even passphrases
- small number, require long-term security storage (e.g., HSM)
- typically established using some trusted set-up

How many keys and what are they used for?

User/application data is protected using symmetric cryptography:

- best speed and bandwidth costs for large data

There are two kinds of cryptographic keys:

- long-term or wrapping keys: they protect other (session) keys
- *session* or *data encapsulation* keys: they protect data

Long-term keys:

- can be asymmetric, symmetric, or even passphrases
- small number, require long-term security storage (e.g., HSM)
- typically established using some trusted set-up

Session-keys:

- always symmetric, short-lived, stored temporarily in memory
- if exposed this compromises only a small amount of data

How are session keys established?

Session keys can be established in three ways:

- Directly derived from other session keys
    - This happens in more complex protocols, e.g., TLS
    - Sessions keys for multiple connections derived from master key

How are session keys established?

Session keys can be established in three ways:

- Directly derived from other session keys
  - This happens in more complex protocols, e.g., TLS
  - Sessions keys for multiple connections derived from master key
- Key-distribution using a trusted agent
  - Long-term keys are pre-agreed with a trusted server
  - Trusted server used as a secure channel
  - Prominent examples are Kerberos and RADIUS

How are session keys established?

Session keys can be established in three ways:

- Directly derived from other session keys
  - This happens in more complex protocols, e.g., TLS
  - Sessions keys for multiple connections derived from master key
- Key-distribution using a trusted agent
  - Long-term keys are pre-agreed with a trusted server
  - Trusted server used as a secure channel
  - Prominent examples are Kerberos and RADIUS
- Authenticated key agreement
  - Long-term keys are pre-agreed between the two parties
  - A handshake is run to jointly construct a session key
  - From symmetric crypto: long-term = symmetric key
  - From public-key crypto: long-term = two or more key pairs (why?)

Key Management in Public-Key cryptography

Open-systems:

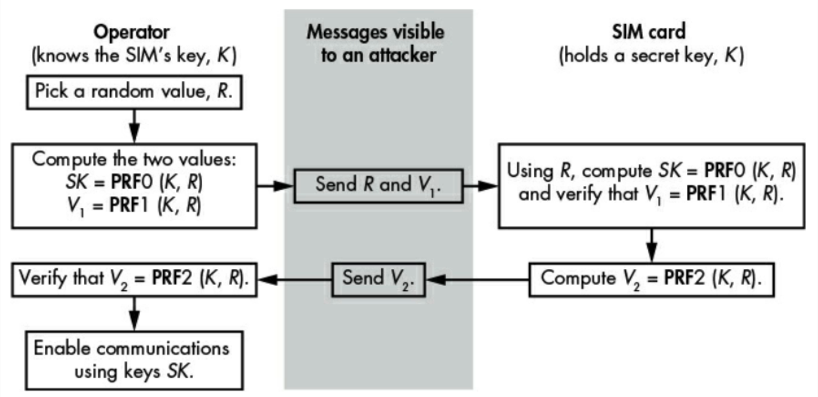
- users come and go without central registration (e.g., Internet)
- not possible to share long-term secret keys with all users/entities

Public-key cryptography solves half of this problem:

- no need to share long-term secret key
- everyone can publish their public keys (one private key per user)
- we still need a way to check public-key authenticity
- this is done using **off-line** trusted-third-parties
- this is called a **Public Key Infrastructure**
- for now we assume we are certain of who owns public key

Symmetric authenticated key agreement

In closed systems, long-term keys can be symmetric (e.g., 3G, 4G)



Operator
(knows the SIM's key, K)

Messages visible to an attacker

SIM card
(holds a secret key, K)

Pick a random value, R.

Compute the two values:
$SK = \mathbf{PRF0}\ (K, R)$
$V_1 = \mathbf{PRF1}\ (K, R)$

Send R and V_1.

Using R, compute $SK = \mathbf{PRF0}\ (K, R)$ and verify that $V_1 = \mathbf{PRF1}\ (K, R)$.

Compute $V_2 = \mathbf{PRF2}\ (K, R)$.

Send V_2.

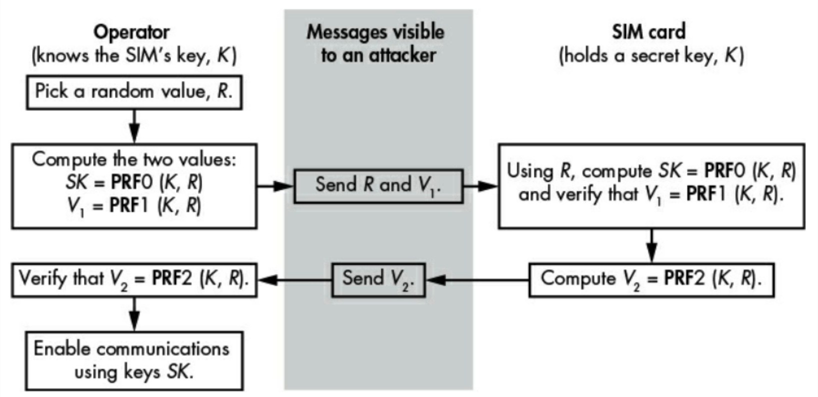
Verify that $V_2 = \mathbf{PRF2}\ (K, R)$.

Enable communications using keys SK.

What happens w/ replay attacks?

Symmetric authenticated key agreement

In closed systems, long-term keys can be symmetric (e.g., 3G, 4G)



Operator (knows the SIM's key, K)

Pick a random value, R.

Compute the two values: SK = **PRF0** (K, R) V_1 = **PRF1** (K, R)

Verify that V_2 = **PRF2** (K, R).

Enable communications using keys SK.

Messages visible to an attacker

Send R and V_1.

Send V_2.

SIM card (holds a secret key, K)

Using R, compute SK = **PRF0** (K, R) and verify that V_1 = **PRF1** (K, R).

Compute V_2 = **PRF2** (K, R).

What happens w/ replay attacks?

What happens if K is compromised (past/future)?

Security of key agreement protocols

All realistic models assume attacker controls communications.

- We need authenticity guarantees on agreed keys
- Protocol assumes long-term keys are authentic
- Should guarantee *mutual authentication*
- The intended party (and only this party) derived the same key

Models vary with respect to what the adversary learns:

- **Channel-only:** what occurs in the network.
- **Data leak:** some session keys, but no long-term keys; other sessions should remain secure.
- **Corruption:** long-term keys; past sessions should remain secure (aka forward secrecy, e.g., in messaging).

Security of key agreement protocols (2)

Other properties (not satisfied by 3G/4G):

- Key control: single party should not determine key, e.g., to decrease entropy of session key
- Key-compromise impersonation: compromising long-term key of party A should not allow me to impersonate B

Efficiency of key agreement protocols

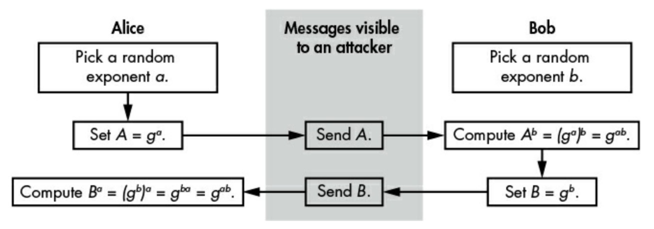
In addition to local computations:

- Number of round-trips, i.e., impact of network latency
- Bandwidth, i.e., how many bits are sent/received
- Precomputation, i.e., can one precompute values so that online computation is much faster?

Public-key Authenticated Key Agreeement: Diffie-Hellman

The first public-key cryptography algorithm dates from 1976.

The basic protocol is secure only over **authenticated channels**.



Alice

Pick a random exponent a.

Set $A = g^a$.

Compute $B^a = (g^b)^a = g^{ba} = g^{ab}$.

Messages visible to an attacker

Send A.

Send B.

Bob

Pick a random exponent b.

Compute $A^b = (g^a)^b = g^{ab}$.

Set $B = g^b$.

This is sometimes called anonymous Diffie-Hellman.

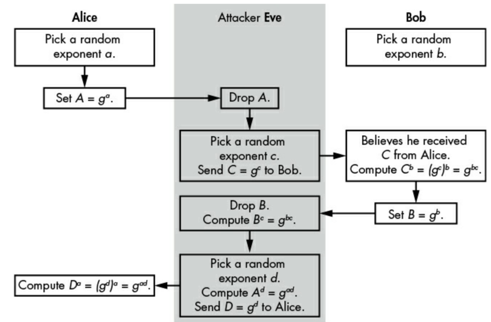
Breaking security equivalent to solving the Diffie-Hellman problem:

- Assuming attacker only observes exchanged public messages
- These are often called *ephemeral public keys*

Man-in-the-Middle Attacks

Essentially all public-key algorithms are vulnerable:

- attacker just gives us a fake public-key that he owns
- if we cannot tell that this is not the correct key
- we talk to attacker thinking it is the intended partner



Alice

Pick a random exponent *a*.

Set $A = g^a$.

Compute $D^a = (g^d)^a = g^{ad}$.

Attacker **Eve**

Drop *A*.

Pick a random exponent *c*. Send $C = g^c$ to Bob.

Drop *B*. Compute $B^c = g^{bc}$.

Pick a random exponent *d*. Compute $A^d = g^{ad}$. Send $D = g^d$ to Alice.

Bob

Pick a random exponent *b*.

Believes he received *C* from Alice. Compute $C^b = (g^c)^b = g^{bc}$.

Set $B = g^b$.

Always works if no way of authenticating public keys

Full authenticated Diffie-Hellman (a la Station-to-Station)

Parties have long-term signature key pairs (one each):

- both parties independently sign protocol execution trace
  - E.g., trace $= (\mathrm{pub}_A, \mathrm{pub}_B, g^a, g^b)$
  - A computes: $\sigma_A = \mathrm{Sign}(\mathrm{priv}_A, \mathrm{trace})$
  - B computes: $\sigma_B = \mathrm{Sign}(\mathrm{priv}_B, \mathrm{trace})$
- parties exchange σ_A, σ_B and check both saw trace (how?)
- key confirmation step MACs public data using derived key, e.g.:
  - A sends: $\mathrm{MAC}(\mathrm{H}(g^{ab}), \mathrm{trace})$
  - B sends: $\mathrm{MAC}(\mathrm{H}(g^{ab}), \mathrm{trace})$

Security of Authenticated Diffie-Hellman

Signing trace and adding key confirmation:

- guarantees mutual authentication.

No party controls final key.

Learning session keys has no influence on other sessions.

Corrupting signing keys has no influence on past keys:

- Forward secrecy crucial for protocols such as TLS and Signal

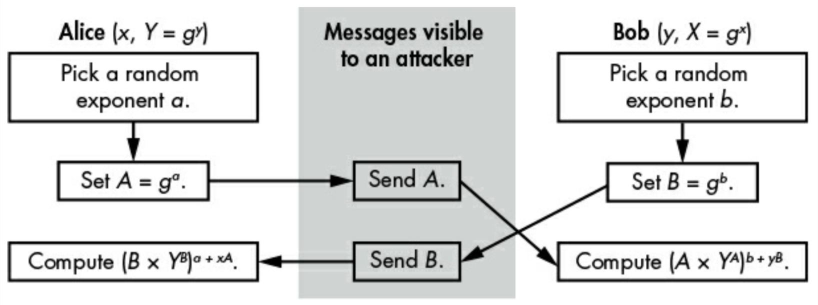
What happens for future sessions?

The use of DH in TLS 1.3 is close to that described here

TLS 1.2 used a version closer to that depicted in book

Menezes-Qu-Vanstone (MQV) protocol

More efficient alternative to previous protocol.



Alice (x, $Y = g^y$)
Messages visible to an attacker
Bob (y, $X = g^x$)
Pick a random exponent a.
Pick a random exponent b.
Set $A = g^a$.
Send A.
Set $B = g^b$.
Compute $(B \times Y^B)^{a+xA}$.
Send B.
Compute $(A \times Y^A)^{b+yB}$.

Learning ephemeral exponent not sufficient to learn session key.

No perfect-forward secrecy:

- Active attacker can choose g^a and record encrypted data
- Later learn long-term secret and decrypt

What can go wrong

Not using a proper key derivation function on g^{ab}.

- Standard KDFs based on cryptographic hash functions

Anonymous DH over unauthenticated channels

Accept non-authenticated public keys

Using groups where DH problem is easier than anticipated

Thank you!

mbb@fc.up.pt

http://www.dcc.fc.up.pt/~mbb