IV – Protocols

David Pointcheval

MPRI – Paris

Ecole normale supérieure/PSL, CNRS & INRIA

Outline

Game-based Security

Simulation-based Security

Encrypted Key Exchange

Conclusion

Game-based Security

Outline

Game-based Security

Key Exchange

Authenticated Key Exchange

Explicit Authentication

Simulation-based Security

Encrypted Key Exchange

Conclusion

Key-Exchange Protocols

A fundamental problem in cryptography:

Enable secure communication over insecure channels

A classical scenario: Users encrypt and authenticate their messages using a common secret key

How to establish such a common secret?

$\longrightarrow$ Key-exchange protocols

Key-Exchange Protocols

A fundamental problem in cryptography:

Enable secure communication over insecure channels

A classical scenario: Users encrypt and authenticate their messages using a common secret key

$\xrightarrow{m_A}$ $\xleftarrow{m'_B}$

$\xrightarrow{m'_A}$ $\xleftarrow{m_B}$

Alice Bob

How to establish such a common secret?

$\longrightarrow$ Key-exchange protocols

Key-Exchange Protocols

A fundamental problem in cryptography:

Enable secure communication over insecure channels

A classical scenario: Users encrypt and authenticate their messages using a common secret key

How to establish such a common secret?

$\longrightarrow$ Key-exchange protocols

Key-Exchange Protocols

A fundamental problem in cryptography:

Enable secure communication over insecure channels

A classical scenario: Users encrypt and authenticate their messages using a common secret key

How to establish such a common secret?

$\longrightarrow$ Key-exchange protocols

$\mathbb{G} = \langle g \rangle$ a group, of prime order q, in which the **CDH** problem is hard

$$
\begin{array}{ccc}
\textit{Alice} & & \textit{Bob} \\
x \stackrel{R}{\leftarrow} \mathbb{Z}_q & & y \stackrel{R}{\leftarrow} \mathbb{Z}_q \\
X = g^x & \xrightarrow{\quad X \quad} & \\
& \xleftarrow{\quad Y \quad} & Y = g^y \\
& Y^x = g^{xy} = X^y &
\end{array}
$$

Allows two parties to establish a common secret:

- The session key should only be known to the involved parties
- The session key should be indistinguishable from a random string for others

Diffie-Hellman Key-Exchange

$\mathbb{G} = \langle g \rangle$ a group, of prime order q, in which the **CDH** problem is hard

$$
\begin{array}{ccc}
\textit{Alice} & & \textit{Bob} \\
x \stackrel{R}{\leftarrow} \mathbb{Z}_q & & y \stackrel{R}{\leftarrow} \mathbb{Z}_q \\
X = g^x & \xrightarrow{\quad X \quad} & \\
& \xleftarrow{\quad Y \quad} & Y = g^y \\
& Y^x = g^{xy} = X^y &
\end{array}
$$

Allows two parties to establish a common secret:

- The session key should only be known to the involved parties
- The session key should be indistinguishable from a random string for others

Diffie-Hellman Key-Exchange

$\mathbb{G} = \langle g \rangle$ a group, of prime order q, in which the **CDH** problem is hard

$$
\begin{array}{ccc}
\textit{Alice} & & \textit{Bob} \\
x \overset{R}{\leftarrow} \mathbb{Z}_q & & y \overset{R}{\leftarrow} \mathbb{Z}_q \\
X = g^x & \xrightarrow{\quad X \quad} & \\
 & \xleftarrow{\quad Y \quad} & Y = g^y \\
 & Y^x = g^{xy} = X^y &
\end{array}
$$

Allows two parties to establish a common secret:

- The session key should only be known to the involved parties
- The session key should be indistinguishable
  from a random string for others

Communication Model

- Users can participate in several executions of the protocol in parallel: Each user's instance is associated to an oracle
  (C^i for the client, and S^j for the server)
- The adversary controls all the communications:
  It can create, modify, transfer, alter, delete messages

This is modeled by various oracle accesses given to oracles

- to let it choose when and what to transmit,
- but also the leakage of information

Communication Model

- Users can participate in several executions of the protocol in parallel: Each user's instance is associated to an oracle
  (C^i for the client, and S^j for the server)
- The adversary controls all the communications:
  It can create, modify, transfer, alter, delete messages

This is modeled by various oracle accesses given to oracles

- to let it choose when and what to transmit,
- but also the leakage of information

Communication Model

- Users can participate in several executions of the protocol in parallel: Each user's instance is associated to an oracle
  (C^i for the client, and S^j for the server)
- The adversary controls all the communications:
  It can create, modify, transfer, alter, delete messages

This is modeled by various oracle accesses given to oracles

- to let it choose when and what to transmit,
- but also the leakage of information

Communication Model

- Users can participate in several executions of the protocol in parallel: Each user's instance is associated to an oracle
  (C^i for the client, and S^j for the server)
- The adversary controls all the communications:
  It can create, modify, transfer, alter, delete messages

This is modeled by various oracle accesses given to oracles

- to let it choose when and what to transmit,
- but also the leakage of information

Security Game: Oracle Accesses

The adversary has access to the oracles:

- Execute(C^i, S^j)

  $\mathcal{A}$ gets the transcript of an execution between C and S

  It models passive attacks (*eavesdropping*)

- Send(U^i, m)

  $\mathcal{A}$ sends the message m to the instance U^i

  It models active attacks against U^i

- Reveal(U^i)

  $\mathcal{A}$ gets the session key established by U^i and its partner

  It models the leakage of the session key, due to a misuse

- Test(U^i) a random bit b is chosen

Security Game: Oracle Accesses

The adversary has access to the oracles:

- Execute(C^i, S^j)

  $\mathcal{A}$ gets the transcript of an execution between C and S

  It models passive attacks (*eavesdropping*)

- Send(U^i, m)

  $\mathcal{A}$ sends the message m to the instance U^i

  It models active attacks against U^i

- Reveal(U^i)

  $\mathcal{A}$ gets the session key established by U^i and its partner

  It models the leakage of the session key, due to a misuse

- Test(U^i) a random bit b is chosen.
  - If $b = 0$, $\mathcal{A}$ gets the session key (*Reveal*(U^i))
  - If $b = 1$, it gets a random key

Security Game: Oracle Accesses

The adversary has access to the oracles:

- Execute(C^i, S^j)

  $\mathcal{A}$ gets the transcript of an execution between C and S

  It models passive attacks (*eavesdropping*)

- Send(U^i, m)

  $\mathcal{A}$ sends the message m to the instance U^i

  It models active attacks against U^i

- Reveal(U^i)

  $\mathcal{A}$ gets the session key established by U^i and its partner

  It models the leakage of the session key, due to a misuse

- Test(U^i) a random bit b is chosen.
  - If $b = 0$, $\mathcal{A}$ gets the session key (Reveal(U^i))
  - If $b = 1$, it gets a random key

Security Game: Oracle Accesses

The adversary has access to the oracles:

- Execute(C^i, S^j)

  $\mathcal{A}$ gets the transcript of an execution between C and S

  It models passive attacks (*eavesdropping*)

- Send(U^i, m)

  $\mathcal{A}$ sends the message m to the instance U^i

  It models active attacks against U^i

- Reveal(U^i)

  $\mathcal{A}$ gets the session key established by U^i and its partner

  It models the leakage of the session key, due to a misuse

- Test(U^i) a random bit b is chosen.
  - If $b = 0$, $\mathcal{A}$ gets the session key (*Reveal*(U^i))
  - If $b = 1$, $\mathcal{A}$ gets a random key

Security Game: Oracle Accesses

The adversary has access to the oracles:

- Execute(C^i, S^j)

  $\mathcal{A}$ gets the transcript of an execution between C and S

  It models passive attacks (*eavesdropping*)
- Send(U^i, m)

  $\mathcal{A}$ sends the message m to the instance U^i

  It models active attacks against U^i
- Reveal(U^i)

  $\mathcal{A}$ gets the session key established by U^i and its partner

  It models the leakage of the session key, due to a misuse
- Test(U^i) a random bit b is chosen.
  - If $b = 0$, $\mathcal{A}$ gets the session key (*Reveal*(U^i))
  - If $b = 1$, it gets a random key

Security Game: Oracle Accesses

The adversary has access to the oracles:

- Execute(C^i, S^j)

  $\mathcal{A}$ gets the transcript of an execution between C and S

  It models passive attacks (*eavesdropping*)
- Send(U^i, m)

  $\mathcal{A}$ sends the message m to the instance U^i

  It models active attacks against U^i
- Reveal(U^i)

  $\mathcal{A}$ gets the session key established by U^i and its partner

  It models the leakage of the session key, due to a misuse
- Test(U^i) a random bit b is chosen.
  - If $b = 0$, $\mathcal{A}$ gets the session key (*Reveal*(U^i))
  - If $b = 1$, it gets a random key

Security Game: Oracle Accesses

The adversary has access to the oracles:

- Execute(C^i, S^j)

  $\mathcal{A}$ gets the transcript of an execution between C and S

  It models passive attacks (*eavesdropping*)

- Send(U^i, m)

  $\mathcal{A}$ sends the message m to the instance U^i

  It models active attacks against U^i

- Reveal(U^i)

  $\mathcal{A}$ gets the session key established by U^i and its partner

  It models the leakage of the session key, due to a misuse

- Test(U^i) a random bit b is chosen.
  - If $b = 0$, $\mathcal{A}$ gets the session key (*Reveal*(U^i))
  - If $b = 1$, it gets a random key

Security Game: Oracle Accesses

The adversary has access to the oracles:

- Execute(C^i, S^j)

  $\mathcal{A}$ gets the transcript of an execution between C and S

  It models passive attacks (*eavesdropping*)
- Send(U^i, m)

  $\mathcal{A}$ sends the message m to the instance U^i

  It models active attacks against U^i
- Reveal(U^i)

  $\mathcal{A}$ gets the session key established by U^i and its partner

  It models the leakage of the session key, due to a misuse
- Test(U^i)   a random bit b is chosen.
  - If $b = 0$, $\mathcal{A}$ gets the session key ($Reveal(U^i)$)
  - If $b = 1$, it gets a random key

  Constraint: no Test-query to a partner of a Reveal-query

Security Game: Some Terminology

Partnership

- two instances are partners
  if they have the same *sid* (session identity)
- the *sid* is set in such a way that two different sessions
  have the same *sid* with negligible probability

Usually, *sid* is the (partial) transcript of the protocol

Freshness

- a user's instance is fresh if a key has been established,
  and it is not trivially known to the adversary
  (a Reveal query has been asked to this instance or its partner)

Security Game: Some Terminology

Partnership

- two instances are partners
  if they have the same *sid* (session identity)
- the *sid* is set in such a way that two different sessions
  have the same *sid* with negligible probability

Usually, *sid* is the (partial) transcript of the protocol

Freshness

- a user's instance is fresh if a key has been established,
  and it is not trivially known to the adversary
  (a Reveal query has been asked to this instance or its partner)

Security Game: Find-then-Guess

Privacy of the key: modeled by a *find-then-guess* security game

$\mathcal{A}$ has to guess the bit b involved in the Test-query:
is the obtained key real or random?

Security Game: Find-then-Guess

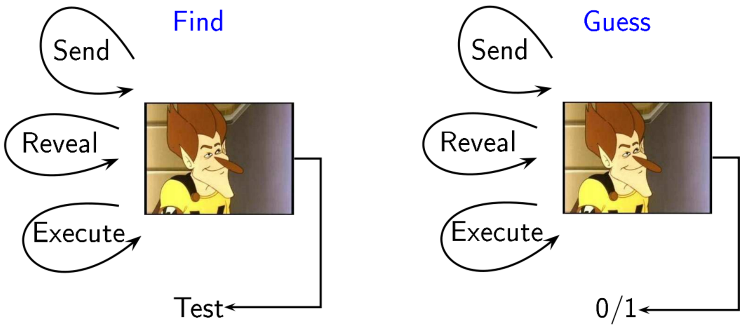
Privacy of the key: modeled by a *find-then-guess* security game

$\mathcal{A}$ has to guess the bit *b* involved in the Test-query:
is the obtained key real or random?

Security Game: Find-then-Guess

Privacy of the key: modeled by a *find-then-guess* security game

$\mathcal{A}$ has to guess the bit b involved in the Test-query:
is the obtained key real or random?



Semantic Security: Find-then-Guess

The semantic security is characterized by

$$\mathbf{Adv}^{\text{ftg}}(\mathcal{A}) = 2 \times \mathbf{Succ}(\mathcal{A}) - 1$$

$$\mathbf{Adv}^{\text{ftg}}(t, q_{execute}, q_{send}, q_{reveal}) = \max\{\mathbf{Adv}^{\text{ftg}}(\mathcal{A})\}$$

- where the adversary wins if it correctly guesses the bit *b* involved in the Test-query
- q_{exe}, q_{send} and q_{reveal} are the numbers of Execute, Send and Reveal-queries resp.

Definition

A Key Exchange Scheme is FtG-Semantically Secure if

$$\mathbf{Adv}^{\text{ftg}}(t) \leq \text{negl}()$$

Semantic Security: Find-then-Guess

The semantic security is characterized by

$$\mathbf{Adv}^{\mathsf{ftg}}(\mathcal{A}) = 2 \times \mathbf{Succ}(\mathcal{A}) - 1$$

$$\mathbf{Adv}^{\mathsf{ftg}}(t, q_{execute}, q_{send}, q_{reveal}) = \max\{\mathbf{Adv}^{\mathsf{ftg}}(\mathcal{A})\}$$

- where the adversary wins if it correctly guesses the bit b involved in the Test-query
- q_{exe}, q_{send} and q_{reveal} are the numbers of Execute, Send and Reveal-queries resp.

Definition

A Key Exchange Scheme is FtG-Semantically Secure if

$$\mathbf{Adv}^{\mathsf{ftg}}(t) \leq \mathrm{negl}()$$

Security Game: Real-or-Random

Privacy of the key: modeled by a *real-or-random* security game

$\mathcal{A}$ has to guess the bit b involved in the Test-queries:
are they all real or random keys?

Security Game: Real-or-Random

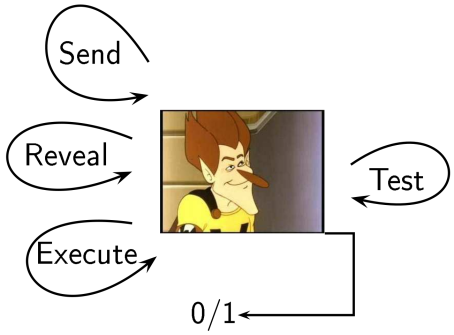
Privacy of the key: modeled by a *real-or-random* security game

$\mathcal{A}$ has to guess the bit b involved in the Test-queries:
are they all real or random keys?

Security Game: Real-or-Random

Privacy of the key: modeled by a *real-or-random* security game

$\mathcal{A}$ has to guess the bit *b* involved in the Test-queries:
are they all real or random keys?



Semantic Security: Real-or-Random

We can even drop the Reveal-Oracle:

- A random bit b is chosen
- Execute(C^i, S^j)

  $\mathcal{A}$ gets the transcript of an execution between C and S

  It models passive attacks (*eavesdropping*)
- Send(U^i, m)

  $\mathcal{A}$ sends the message m to the instance U^i

  It models active attacks against U^i
- Test(U^i) If U^i is not fresh: same answer as for its partner

  Otherwise

We can even drop the Reveal-Oracle:

- A random bit b is chosen
- Execute(C^i, S^j)

  $\mathcal{A}$ gets the transcript of an execution between C and S

  It models passive attacks (*eavesdropping*)
- Send(U^i, m)

  $\mathcal{A}$ sends the message m to the instance U^i

  It models active attacks against U^i
- Test(U^i) If U^i is not fresh: same answer as for its partner

  Otherwise
  - If $b = 0$, $\mathcal{A}$ gets the session key
  - If $b = 1$, it gets a random key

Semantic Security: Real-or-Random

We can even drop the Reveal-Oracle:

- A random bit b is chosen
- Execute(C^i, S^j)

  $\mathcal{A}$ gets the transcript of an execution between C and S

  It models passive attacks (*eavesdropping*)
- Send(U^i, m)

  $\mathcal{A}$ sends the message m to the instance U^i

  It models active attacks against U^i
- Test(U^i) If U^i is not fresh: same answer as for its partner
  Otherwise
  - If $b = 0$, $\mathcal{A}$ gets the session key
  - If $b = 1$, it gets a random key

Semantic Security: Real-or-Random

We can even drop the Reveal-Oracle:

- A random bit b is chosen
- Execute(C^i, S^j)

  $\mathcal{A}$ gets the transcript of an execution between C and S

  It models passive attacks (*eavesdropping*)
- Send(U^i, m)

  $\mathcal{A}$ sends the message m to the instance U^i

  It models active attacks against U^i
- Test(U^i) If U^i is not fresh: same answer as for its partner

  Otherwise
  - If $b = 0$, $\mathcal{A}$ gets the session key
  - If $b = 1$, it gets a random key

Semantic Security: Real-or-Random

We can even drop the Reveal-Oracle:

- A random bit b is chosen
- Execute(C^i, S^j)

  $\mathcal{A}$ gets the transcript of an execution between C and S

  It models passive attacks (*eavesdropping*)
- Send(U^i, m)

  $\mathcal{A}$ sends the message m to the instance U^i

  It models active attacks against U^i
- Test(U^i) If U^i is not fresh: same answer as for its partner
  Otherwise
  - If $b = 0$, $\mathcal{A}$ gets the session key
  - If $b = 1$, it gets a random key

Semantic Security: Real-or-Random

We can even drop the Reveal-Oracle:

- A random bit b is chosen
- Execute(C^i, S^j)

  $\mathcal{A}$ gets the transcript of an execution between C and S

  It models passive attacks (*eavesdropping*)
- Send(U^i, m)

  $\mathcal{A}$ sends the message m to the instance U^i

  It models active attacks against U^i
- Test(U^i) If U^i is not fresh: same answer as for its partner

  Otherwise
  - If $b = 0$, $\mathcal{A}$ gets the session key
  - If $b = 1$, it gets a random key

Semantic Security: Real-or-Random

We can even drop the Reveal-Oracle:

- A random bit b is chosen
- Execute(C^i, S^j)

  $\mathcal{A}$ gets the transcript of an execution between C and S

  It models passive attacks (*eavesdropping*)
- Send(U^i, m)

  $\mathcal{A}$ sends the message m to the instance U^i

  It models active attacks against U^i
- Test(U^i) If U^i is not fresh: same answer as for its partner

  Otherwise
  - If $b = 0$, $\mathcal{A}$ gets the session key
  - If $b = 1$, it gets a random key

Semantic Security: Real-or-Random

The semantic security is characterized by

$$\mathbf{Adv}^{\mathsf{ror}}(\mathcal{A}) = 2 \times \mathbf{Succ}(\mathcal{A}) - 1$$

$$\mathbf{Adv}^{\mathsf{ror}}(t, q_{execute}, q_{send}, q_{test}) = \max\{\mathbf{Adv}^{\mathsf{ror}}(\mathcal{A})\}$$

Definition

A Key Exchange Scheme is RoR-Semantically Secure if

$$\mathbf{Adv}^{\mathsf{ror}}(t, q_{execute}, q_{send}, q_{test}) \leq \mathsf{negl}()$$

Semantic Security: Real-or-Random

The semantic security is characterized by

$$\mathbf{Adv}^{\mathsf{ror}}(\mathcal{A}) = 2 \times \mathbf{Succ}(\mathcal{A}) - 1$$

$$\mathbf{Adv}^{\mathsf{ror}}(t, q_{execute}, q_{send}, q_{test}) = \max\{\mathbf{Adv}^{\mathsf{ror}}(\mathcal{A})\}$$

Definition

A Key Exchange Scheme is RoR-Semantically Secure if

$$\mathbf{Adv}^{\mathsf{ror}}(t, q_{execute}, q_{send}, q_{test}) \leq \mathsf{negl}()$$

Real-or-Random vs. Find-then-Guess

Theorem

$$\mathbf{Adv}^{\mathsf{ftg}}(t, q_{execute}, q_{send}, q_{reveal}) \leq 2 \times \mathbf{Adv}^{\mathsf{ror}}(t, q_{execute}, q_{send}, q_{reveal}+1)$$

Let $\mathcal{A}$ be a FtG-adversary

We build an adversary $\mathcal{B}$ against the RoR security game:

- A random bit b is chosen by the RoR challenger
- Execute(C^i, S^j) and Send(U^i, m) queries are forwarded by $\mathcal{B}$
- Reveal(U^i) is answered Test(U^i)
- Test(U^i) If U^i is not fresh: same answer as for its partner
  Otherwise, $\mathcal{B}$ chooses a random bit β
  - If $\beta = 0$, one answers Test(U^i)
  - If $\beta = 1$, one answers a random key
- From $\mathcal{A}$'s answer β', $\mathcal{B}$ outputs $(\beta = \beta')$

Real-or-Random vs. Find-then-Guess

Theorem

$$\mathbf{Adv}^{\mathsf{ftg}}(t, q_{execute}, q_{send}, q_{reveal}) \leq 2 \times \mathbf{Adv}^{\mathsf{ror}}(t, q_{execute}, q_{send}, q_{reveal} + 1)$$

Let $\mathcal{A}$ be a FtG-adversary

We build an adversary $\mathcal{B}$ against the RoR security game:

- A random bit b is chosen by the RoR challenger
- Execute(C^i, S^j) and Send(U^i, m) queries are forwarded by $\mathcal{B}$
- Reveal(U^i) is answered Test(U^i)
- Test(U^i) If U^i is not fresh: same answer as for its partner
  Otherwise, $\mathcal{B}$ chooses a random bit β
  - If $\beta = 0$, one answers Test(U^i)
  - If $\beta = 1$, one answers a random key
- From $\mathcal{A}$'s answer β', $\mathcal{B}$ outputs $(\beta = \beta')$

Real-or-Random vs. Find-then-Guess

Theorem

$$\mathbf{Adv}^{\mathsf{ftg}}(t, q_{execute}, q_{send}, q_{reveal}) \leq 2 \times \mathbf{Adv}^{\mathsf{ror}}(t, q_{execute}, q_{send}, q_{reveal} + 1)$$

Let $\mathcal{A}$ be a FtG-adversary

We build an adversary $\mathcal{B}$ against the RoR security game:

- A random bit b is chosen by the RoR challenger
- Execute(C^i, S^j) and Send(U^i, m) queries are forwarded by $\mathcal{B}$
- Reveal(U^i) is answered Test(U^i)
- Test(U^i) If U^i is not fresh: same answer as for its partner
  Otherwise, $\mathcal{B}$ chooses a random bit β
  - If $\beta = 0$, one answers Test(U^i)
  - If $\beta = 1$, one answers a random key
- From $\mathcal{A}$'s answer β', $\mathcal{B}$ outputs $(\beta = \beta')$

Real-or-Random vs. Find-then-Guess

Theorem

$$\mathbf{Adv}^{\mathsf{ftg}}(t, q_{execute}, q_{send}, q_{reveal}) \leq 2 \times \mathbf{Adv}^{\mathsf{ror}}(t, q_{execute}, q_{send}, q_{reveal} + 1)$$

Let $\mathcal{A}$ be a FtG-adversary

We build an adversary $\mathcal{B}$ against the RoR security game:

- A random bit b is chosen by the RoR challenger
- Execute(C^i, S^j) and Send(U^i, m) queries are forwarded by $\mathcal{B}$
- Reveal(U^i) is answered Test(U^i)
- Test(U^i) If U^i is not fresh: same answer as for its partner
  Otherwise, $\mathcal{B}$ chooses a random bit β
  - If $\beta = 0$, one answers Test(U^i)
  - If $\beta = 1$, one answers a random key
- From $\mathcal{A}$'s answer β', $\mathcal{B}$ outputs $(\beta = \beta')$

Real-or-Random vs. Find-then-Guess

Theorem

$$\mathbf{Adv}^{\mathsf{ftg}}(t, q_{execute}, q_{send}, q_{reveal}) \leq 2 \times \mathbf{Adv}^{\mathsf{ror}}(t, q_{execute}, q_{send}, q_{reveal} + 1)$$

Let $\mathcal{A}$ be a FtG-adversary

We build an adversary $\mathcal{B}$ against the RoR security game:

- A random bit b is chosen by the RoR challenger
- Execute(C^i, S^j) and Send(U^i, m) queries are forwarded by $\mathcal{B}$
- Reveal(U^i) is answered Test(U^i)
- Test(U^i) If U^i is not fresh: same answer as for its partner
  Otherwise, $\mathcal{B}$ chooses a random bit β
  - If $\beta = 0$, one answers Test(U^i)
  - If $\beta = 1$, one answers a random key
- From $\mathcal{A}$'s answer β', $\mathcal{B}$ outputs $(\beta = \beta')$

Real-or-Random vs. Find-then-Guess

Theorem

$$\mathbf{Adv}^{\mathsf{ftg}}(t, q_{execute}, q_{send}, q_{reveal}) \leq 2 \times \mathbf{Adv}^{\mathsf{ror}}(t, q_{execute}, q_{send}, q_{reveal} + 1)$$

Let $\mathcal{A}$ be a FtG-adversary

We build an adversary $\mathcal{B}$ against the RoR security game:

- A random bit b is chosen by the RoR challenger
- Execute(C^i, S^j) and Send(U^i, m) queries are forwarded by $\mathcal{B}$
- Reveal(U^i) is answered Test(U^i)
- Test(U^i) If U^i is not fresh: same answer as for its partner
  Otherwise, $\mathcal{B}$ chooses a random bit β
  - If $\beta = 0$, one answers Test(U^i)
  - If $\beta = 1$, one answers a random key
- From $\mathcal{A}$'s answer β', $\mathcal{B}$ outputs $(\beta = \beta')$

Real-or-Random vs. Find-then-Guess

Theorem

$$\mathbf{Adv}^{\mathsf{ftg}}(t, q_{execute}, q_{send}, q_{reveal}) \leq 2 \times \mathbf{Adv}^{\mathsf{ror}}(t, q_{execute}, q_{send}, q_{reveal} + 1)$$

Let $\mathcal{A}$ be a FtG-adversary
We build an adversary $\mathcal{B}$ against the RoR security game:

- A random bit b is chosen by the RoR challenger
- Execute(C^i, S^j) and Send(U^i, m) queries are forwarded by $\mathcal{B}$
- Reveal(U^i) is answered Test(U^i)
- Test(U^i) If U^i is not fresh: same answer as for its partner
  Otherwise, $\mathcal{B}$ chooses a random bit β
  - If $\beta = 0$, one answers Test(U^i)
  - If $\beta = 1$, one answers a random key
- From $\mathcal{A}$'s answer β', $\mathcal{B}$ outputs $(\beta = \beta')$

Real-or-Random vs. Find-then-Guess

If b is the Real choice, then the view of $\mathcal{A}$ is

- Execute(C^i, S^j) and Send(U^i, m) queries: correct
- Reveal(U^i): Test(U^i) with Real
- Test(U^i) If U^i is not fresh: same answer as for its partner
  Otherwise, a random bit β is drawn
  - If $\beta = 0$, one answers Test(U^i) with Real
  - If $\beta = 1$, one answers a random key

This is the FtG game

$$2 \times \Pr[\beta' = \beta \mid b = 0] - 1 = \mathbf{Adv}^{\mathsf{ftg}}(\mathcal{A})$$

Real-or-Random vs. Find-then-Guess

If b is the Real choice, then the view of $\mathcal{A}$ is

- $\mathsf{Execute}(C^i, S^j)$ and $\mathsf{Send}(U^i, m)$ queries: correct
- $\mathsf{Reveal}(U^i)$: $\mathsf{Test}(U^i)$ with Real
- $\mathsf{Test}(U^i)$ If U^i is not fresh: same answer as for its partner
  Otherwise, a random bit β is drawn
  - If $\beta = 0$, one answers $\mathsf{Test}(U^i)$ with Real
  - If $\beta = 1$, one answers a random key

This is the FtG game

$$2 \times \Pr[\beta' = \beta \mid b = 0] - 1 = \mathbf{Adv}^{\mathsf{ftg}}(\mathcal{A})$$

Real-or-Random vs. Find-then-Guess

If b is the Real choice, then the view of $\mathcal{A}$ is

- Execute(C^i, S^j) and Send(U^i, m) queries: correct
- Reveal(U^i): Test(U^i) with Real
- Test(U^i) If U^i is not fresh: same answer as for its partner
  Otherwise, a random bit β is drawn
  - If $\beta = 0$, one answers Test(U^i) with Real
  - If $\beta = 1$, one answers a random key

This is the FtG game

$$2 \times \Pr[\beta' = \beta \mid b = 0] - 1 = \mathbf{Adv}^{\mathsf{ftg}}(\mathcal{A})$$

Real-or-Random vs. Find-then-Guess

If b is the Real choice, then the view of $\mathcal{A}$ is

- Execute(C^i, S^j) and Send(U^i, m) queries: correct
- Reveal(U^i): Test(U^i) with Real
- Test(U^i) If U^i is not fresh: same answer as for its partner
  Otherwise, a random bit β is drawn
  - If $\beta = 0$, one answers Test(U^i) with Real
  - If $\beta = 1$, one answers a random key

This is the FtG game

$$2 \times \Pr[\beta' = \beta \mid b = 0] - 1 = \mathbf{Adv}^{\mathsf{ftg}}(\mathcal{A})$$

Real-or-Random vs. Find-then-Guess

If b is the Real choice, then the view of $\mathcal{A}$ is

- Execute(C^i, S^j) and Send(U^i, m) queries: correct
- Reveal(U^i): Test(U^i) with Real
- Test(U^i) If U^i is not fresh: same answer as for its partner
  Otherwise, a random bit β is drawn
  - If $\beta = 0$, one answers Test(U^i) with Real
  - If $\beta = 1$, one answers a random key

This is the FtG game

$$2 \times \Pr[\beta' = \beta \mid b = 0] - 1 = \mathbf{Adv}^{\mathsf{ftg}}(\mathcal{A})$$

Real-or-Random vs. Find-then-Guess

If b is the Real choice, then the view of $\mathcal{A}$ is

- Execute(C^i, S^j) and Send(U^i, m) queries: correct
- Reveal(U^i): Test(U^i) with Real
- Test(U^i) If U^i is not fresh: same answer as for its partner
  Otherwise, a random bit β is drawn
  - If $\beta = 0$, one answers Test(U^i) with Real
  - If $\beta = 1$, one answers a random key

This is the FtG game

$$2 \times \Pr[\beta' = \beta \,|\, b = 0] - 1 = \mathbf{Adv}^{\mathrm{ftg}}(\mathcal{A})$$

Real-or-Random vs. Find-then-Guess

If b is the Real choice, then the view of $\mathcal{A}$ is

- Execute(C^i, S^j) and Send(U^i, m) queries: correct
- Reveal(U^i): Test(U^i) with Real
- Test(U^i) If U^i is not fresh: same answer as for its partner
  Otherwise, a random bit β is drawn
  - If $\beta = 0$, one answers Test(U^i) with Real
  - If $\beta = 1$, one answers a random key

This is the FtG game

$$2 \times \Pr[\beta' = \beta \mid b = 0] - 1 = \mathbf{Adv}^{\text{ftg}}(\mathcal{A})$$

Real-or-Random vs. Find-then-Guess

If b is the Real choice, then the view of $\mathcal{A}$ is

- Execute(C^i, S^j) and Send(U^i, m) queries: correct
- Reveal(U^i): Test(U^i) with Real
- Test(U^i) If U^i is not fresh: same answer as for its partner
  Otherwise, a random bit β is drawn
  - If $\beta = 0$, one answers Test(U^i) with Real
  - If $\beta = 1$, one answers a random key

This is the FtG game

$$2 \times \Pr[\beta' = \beta \mid b = 0] - 1 = \mathbf{Adv}^{\mathsf{ftg}}(\mathcal{A})$$

Real-or-Random vs. Find-then-Guess

If b is the Random choice, then the view of $\mathcal{A}$ is

- $\mathsf{Execute}(C^i, S^j)$ and $\mathsf{Send}(U^i, m)$ queries: correct
- $\mathsf{Reveal}(U^i)$: $\mathsf{Test}(U^i)$ with Random
- $\mathsf{Test}(U^i)$ If U^i is not fresh: same answer as for its partner
  Otherwise, one answers a random key

The view is independent of β

$$2 \times \Pr[\beta' = \beta \mid b = 1] - 1 = 0$$

$$\begin{aligned} \mathbf{Adv}^{\mathsf{ror}}(\mathcal{B}) &= 2 \times \Pr[\beta' = \beta] - 1 = \mathbf{Adv}^{\mathsf{ftg}}(\mathcal{A})/2 \\ &\leq \mathbf{Adv}^{\mathsf{ror}}(t, q_{execute}, q_{send}, q_{reveal} + 1) \end{aligned}$$

$$\mathbf{Adv}^{\mathsf{ftg}}(t, q_{execute}, q_{send}, q_{reveal}) \leq 2 \times \mathbf{Adv}^{\mathsf{ror}}(t, q_{execute}, q_{send}, q_{reveal} + 1)$$

Real-or-Random vs. Find-then-Guess

If b is the Random choice, then the view of $\mathcal{A}$ is

- Execute(C^i, S^j) and Send(U^i, m) queries: correct
- Reveal(U^i): Test(U^i) with Random
- Test(U^i) If U^i is not fresh: same answer as for its partner
  Otherwise, one answers a random key

The view is independent of β

$$2 \times \Pr[\beta' = \beta \,|\, b = 1] - 1 = 0$$

$$\begin{aligned} \mathbf{Adv}^{\mathsf{ror}}(\mathcal{B}) &= 2 \times \Pr[\beta' = \beta] - 1 = \mathbf{Adv}^{\mathsf{ftg}}(\mathcal{A})/2 \\ &\leq \mathbf{Adv}^{\mathsf{ror}}(t, q_{execute}, q_{send}, q_{reveal} + 1) \end{aligned}$$

$$\mathbf{Adv}^{\mathsf{ftg}}(t, q_{execute}, q_{send}, q_{reveal}) \leq 2 \times \mathbf{Adv}^{\mathsf{ror}}(t, q_{execute}, q_{send}, q_{reveal} + 1)$$

Real-or-Random vs. Find-then-Guess

If b is the Random choice, then the view of $\mathcal{A}$ is

- Execute(C^i, S^j) and Send(U^i, m) queries: correct
- Reveal(U^i): Test(U^i) with Random
- Test(U^i) If U^i is not fresh: same answer as for its partner
  Otherwise, one answers a random key

The view is independent of β

$$2 \times \Pr[\beta' = \beta \mid b = 1] - 1 = 0$$

$$\begin{aligned} \mathbf{Adv}^{\mathsf{ror}}(\mathcal{B}) &= 2 \times \Pr[\beta' = \beta] - 1 = \mathbf{Adv}^{\mathsf{ftg}}(\mathcal{A})/2 \\ &\leq \mathbf{Adv}^{\mathsf{ror}}(t, q_{execute}, q_{send}, q_{reveal} + 1) \end{aligned}$$

$$\mathbf{Adv}^{\mathsf{ftg}}(t, q_{execute}, q_{send}, q_{reveal}) \leq 2 \times \mathbf{Adv}^{\mathsf{ror}}(t, q_{execute}, q_{send}, q_{reveal} + 1)$$

Real-or-Random vs. Find-then-Guess

If b is the Random choice, then the view of $\mathcal{A}$ is

- Execute(C^i, S^j) and Send(U^i, m) queries: correct
- Reveal(U^i): Test(U^i) with Random
- Test(U^i) If U^i is not fresh: same answer as for its partner
  Otherwise, one answers a random key

The view is independent of β

$$2 \times \Pr[\beta' = \beta \mid b = 1] - 1 = 0$$

$$\begin{aligned} \mathbf{Adv}^{\text{ror}}(\mathcal{B}) &= 2 \times \Pr[\beta' = \beta] - 1 = \mathbf{Adv}^{\text{ftg}}(\mathcal{A})/2 \\ &\leq \mathbf{Adv}^{\text{ror}}(t, q_{execute}, q_{send}, q_{reveal} + 1) \end{aligned}$$

$$\mathbf{Adv}^{\text{ftg}}(t, q_{execute}, q_{send}, q_{reveal}) \leq 2 \times \mathbf{Adv}^{\text{ror}}(t, q_{execute}, q_{send}, q_{reveal} + 1)$$

Real-or-Random vs. Find-then-Guess

If b is the Random choice, then the view of $\mathcal{A}$ is

- Execute(C^i, S^j) and Send(U^i, m) queries: correct
- Reveal(U^i): Test(U^i) with Random
- Test(U^i) If U^i is not fresh: same answer as for its partner
  Otherwise, one answers a random key

The view is independent of β

$$2 \times \Pr[\beta' = \beta \mid b = 1] - 1 = 0$$

$$\begin{aligned} \mathbf{Adv}^{\mathsf{ror}}(\mathcal{B}) &= 2 \times \Pr[\beta' = \beta] - 1 = \mathbf{Adv}^{\mathsf{ftg}}(\mathcal{A})/2 \\ &\leq \mathbf{Adv}^{\mathsf{ror}}(t, q_{execute}, q_{send}, q_{reveal} + 1) \end{aligned}$$

$$\mathbf{Adv}^{\mathsf{ftg}}(t, q_{execute}, q_{send}, q_{reveal}) \leq 2 \times \mathbf{Adv}^{\mathsf{ror}}(t, q_{execute}, q_{send}, q_{reveal} + 1)$$

Real-or-Random vs. Find-then-Guess

If b is the Random choice, then the view of $\mathcal{A}$ is

- Execute(C^i, S^j) and Send(U^i, m) queries: correct
- Reveal(U^i): Test(U^i) with Random
- Test(U^i) If U^i is not fresh: same answer as for its partner
  Otherwise, one answers a random key

The view is independent of β

$$2 \times \Pr[\beta' = \beta \mid b = 1] - 1 = 0$$

$$\begin{aligned} \mathbf{Adv}^{\mathsf{ror}}(\mathcal{B}) &= 2 \times \Pr[\beta' = \beta] - 1 = \mathbf{Adv}^{\mathsf{ftg}}(\mathcal{A})/2 \\ &\leq \mathbf{Adv}^{\mathsf{ror}}(t, q_{execute}, q_{send}, q_{reveal} + 1) \end{aligned}$$

$$\mathbf{Adv}^{\mathsf{ftg}}(t, q_{execute}, q_{send}, q_{reveal}) \leq 2 \times \mathbf{Adv}^{\mathsf{ror}}(t, q_{execute}, q_{send}, q_{reveal} + 1)$$

Real-or-Random vs. Find-then-Guess

If b is the Random choice, then the view of $\mathcal{A}$ is

- Execute(C^i, S^j) and Send(U^i, m) queries: correct
- Reveal(U^i): Test(U^i) with Random
- Test(U^i) If U^i is not fresh: same answer as for its partner
  Otherwise, one answers a random key

The view is independent of β

$$2 \times \Pr[\beta' = \beta \mid b = 1] - 1 = 0$$

$$\begin{aligned} \mathbf{Adv}^{\text{ror}}(\mathcal{B}) &= 2 \times \Pr[\beta' = \beta] - 1 = \mathbf{Adv}^{\text{ftg}}(\mathcal{A})/2 \\ &\le \mathbf{Adv}^{\text{ror}}(t, q_{execute}, q_{send}, q_{reveal} + 1) \end{aligned}$$

$$\mathbf{Adv}^{\text{ftg}}(t, q_{execute}, q_{send}, q_{reveal}) \le 2 \times \mathbf{Adv}^{\text{ror}}(t, q_{execute}, q_{send}, q_{reveal} + 1)$$

Real-or-Random vs. Find-then-Guess

If b is the Random choice, then the view of $\mathcal{A}$ is

- Execute(C^i, S^j) and Send(U^i, m) queries: correct
- Reveal(U^i): Test(U^i) with Random
- Test(U^i) If U^i is not fresh: same answer as for its partner
  Otherwise, one answers a random key

The view is independent of β

$$2 \times \Pr[\beta' = \beta \,|\, b = 1] - 1 = 0$$

$$\begin{aligned} \mathbf{Adv}^{\mathsf{ror}}(\mathcal{B}) &= 2 \times \Pr[\beta' = \beta] - 1 = \mathbf{Adv}^{\mathsf{ftg}}(\mathcal{A})/2 \\ &\leq \mathbf{Adv}^{\mathsf{ror}}(t, q_{execute}, q_{send}, q_{reveal} + 1) \end{aligned}$$

$$\mathbf{Adv}^{\mathsf{ftg}}(t, q_{execute}, q_{send}, q_{reveal}) \leq 2 \times \mathbf{Adv}^{\mathsf{ror}}(t, q_{execute}, q_{send}, q_{reveal} + 1)$$

Real-or-Random vs. Find-then-Guess

Theorem

$$\mathbf{Adv}^{\mathsf{ror}}(t, q_{execute}, q_{send}, q_{test}) \leq q_{test} \times \mathbf{Adv}^{\mathsf{ftg}}(t, q_{execute}, q_{send}, q_{test} - 1)$$

Let $\mathcal{A}$ be a RoR-adversary
We build an adversary $\mathcal{B}$ against the FtG security game:

- A random bit b is chosen by the FtG challenger
- $\mathcal{B}$ chooses a random index J
- Execute(C^i, S^j) and Send(U^i, m) queries are forwarded by $\mathcal{B}$
- The j-th Test(U^i) query:
  - If $j < J$, one answers Reveal(U^i)
  - If $j = J$, one answers Test(U^i)
  - If $j > J$, one answers a random key
- From $\mathcal{A}$'s answer b', $\mathcal{B}$ forwards b'

Real-or-Random vs. Find-then-Guess

Theorem

$$\mathbf{Adv}^{\mathsf{ror}}(t, q_{execute}, q_{send}, q_{test}) \leq q_{test} \times \mathbf{Adv}^{\mathsf{ftg}}(t, q_{execute}, q_{send}, q_{test} - 1)$$

Let $\mathcal{A}$ be a RoR-adversary
We build an adversary $\mathcal{B}$ against the FtG security game:

- A random bit b is chosen by the FtG challenger
- $\mathcal{B}$ chooses a random index J
- Execute(C^i, S^j) and Send(U^i, m) queries are forwarded by $\mathcal{B}$
- The j-th Test(U^i) query:
  - If $j < J$, one answers Reveal(U^i)
  - If $j = J$, one answers Test(U^i)
  - If $j > J$, one answers a random key
- From $\mathcal{A}$'s answer b', $\mathcal{B}$ forwards b'

Real-or-Random vs. Find-then-Guess

Theorem

$$\mathbf{Adv}^{\text{ror}}(t, q_{execute}, q_{send}, q_{test}) \leq q_{test} \times \mathbf{Adv}^{\text{ftg}}(t, q_{execute}, q_{send}, q_{test} - 1)$$

Let $\mathcal{A}$ be a RoR-adversary
We build an adversary $\mathcal{B}$ against the FtG security game:

- A random bit b is chosen by the FtG challenger
- $\mathcal{B}$ chooses a random index J
- Execute(C^i, S^j) and Send(U^i, m) queries are forwarded by $\mathcal{B}$
- The j-th Test(U^i) query:
  - If $j < J$, one answers Reveal(U^i)
  - If $j = J$, one answers Test(U^i)
  - If $j > J$, one answers a random key
- From $\mathcal{A}$'s answer b', $\mathcal{B}$ forwards b'

Real-or-Random vs. Find-then-Guess

Theorem

$$\mathbf{Adv}^{\mathsf{ror}}(t, q_{execute}, q_{send}, q_{test}) \leq q_{test} \times \mathbf{Adv}^{\mathsf{ftg}}(t, q_{execute}, q_{send}, q_{test} - 1)$$

Let $\mathcal{A}$ be a RoR-adversary

We build an adversary $\mathcal{B}$ against the FtG security game:

- A random bit b is chosen by the FtG challenger
- $\mathcal{B}$ chooses a random index J
- Execute(C^i, S^j) and Send(U^i, m) queries are forwarded by $\mathcal{B}$
- The j-th Test(U^i) query:
    - If $j < J$, one answers Reveal(U^i)
    - If $j = J$, one answers Test(U^i)
    - If $j > J$, one answers a random key
- From $\mathcal{A}$'s answer b', $\mathcal{B}$ forwards b'

Real-or-Random vs. Find-then-Guess

Theorem

$$\mathbf{Adv}^{\mathsf{ror}}(t, q_{execute}, q_{send}, q_{test}) \leq q_{test} \times \mathbf{Adv}^{\mathsf{ftg}}(t, q_{execute}, q_{send}, q_{test} - 1)$$

Let $\mathcal{A}$ be a RoR-adversary

We build an adversary $\mathcal{B}$ against the FtG security game:

- A random bit b is chosen by the FtG challenger
- $\mathcal{B}$ chooses a random index J
- Execute(C^i, S^j) and Send(U^i, m) queries are forwarded by $\mathcal{B}$
- The j-th Test(U^i) query:
  - If $j < J$, one answers Reveal(U^i)
  - If $j = J$, one answers Test(U^i)
  - If $j > J$, one answers a random key
- From $\mathcal{A}$'s answer b', $\mathcal{B}$ forwards b'

Real-or-Random vs. Find-then-Guess

Theorem

$$\mathbf{Adv}^{\mathsf{ror}}(t, q_{execute}, q_{send}, q_{test}) \leq q_{test} \times \mathbf{Adv}^{\mathsf{ftg}}(t, q_{execute}, q_{send}, q_{test} - 1)$$

Let $\mathcal{A}$ be a RoR-adversary

We build an adversary $\mathcal{B}$ against the FtG security game:

- A random bit b is chosen by the FtG challenger
- $\mathcal{B}$ chooses a random index J
- Execute(C^i, S^j) and Send(U^i, m) queries are forwarded by $\mathcal{B}$
- The j-th Test(U^i) query:
  - If $j < J$, one answers Reveal(U^i)
  - If $j = J$, one answers Test(U^i)
  - If $j > J$, one answers a random key
- From $\mathcal{A}$'s answer b', $\mathcal{B}$ forwards b'

Real-or-Random vs. Find-then-Guess

Theorem

$$\mathbf{Adv}^{\mathsf{ror}}(t, q_{execute}, q_{send}, q_{test}) \leq q_{test} \times \mathbf{Adv}^{\mathsf{ftg}}(t, q_{execute}, q_{send}, q_{test} - 1)$$

Let $\mathcal{A}$ be a RoR-adversary
We build an adversary $\mathcal{B}$ against the FtG security game:

- A random bit b is chosen by the FtG challenger
- $\mathcal{B}$ chooses a random index J
- Execute(C^i, S^j) and Send(U^i, m) queries are forwarded by $\mathcal{B}$
- The j-th Test(U^i) query:
  - If $j < J$, one answers Reveal(U^i)
  - If $j = J$, one answers Test(U^i)
  - If $j > J$, one answers a random key
- From $\mathcal{A}$'s answer b', $\mathcal{B}$ forwards b'

Real-or-Random vs. Find-then-Guess

Theorem

$$\mathbf{Adv}^{\text{ror}}(t, q_{execute}, q_{send}, q_{test}) \leq q_{test} \times \mathbf{Adv}^{\text{ftg}}(t, q_{execute}, q_{send}, q_{test} - 1)$$

Let $\mathcal{A}$ be a RoR-adversary
We build an adversary $\mathcal{B}$ against the FtG security game:

- A random bit b is chosen by the FtG challenger
- $\mathcal{B}$ chooses a random index J
- Execute(C^i, S^j) and Send(U^i, m) queries are forwarded by $\mathcal{B}$
- The j-th Test(U^i) query:
  - If $j < J$, one answers Reveal(U^i)
  - If $j = J$, one answers Test(U^i)
  - If $j > J$, one answers a random key
- From $\mathcal{A}$'s answer b', $\mathcal{B}$ forwards b'

Real-or-Random vs. Find-then-Guess

This is a sequence of hybrid games G_J:

- G_1, with b Random, is the RoR game with Random
- $G_{q_{test}}$, with b Real, is the RoR game with Real
- G_{J-1} with b Real is identical to G_J with b Random

$$\begin{aligned}
|\Pr_1[b' = 1 \mid b = 1] - \Pr_{q_{test}}[b' = 1 \mid b = 0]| &= \mathbf{Adv}^{\mathsf{ror}}(\mathcal{A}) \\
|\Pr_J[b' = 1 \mid b = 0] - \Pr_J[b' = 1 \mid b = 1]| &\leq \mathbf{Adv}^{\mathsf{ftg}}(t, q_{execute}, q_{send}, J-1) \\
&\leq \mathbf{Adv}^{\mathsf{ftg}}(t, q_{execute}, q_{send}, q_{test}-1)
\end{aligned}$$

$$\mathbf{Adv}^{\mathsf{ror}}(t, q_{execute}, q_{send}, q_{test}) \leq q_{test} \times \mathbf{Adv}^{\mathsf{ftg}}(t, q_{execute}, q_{send}, q_{test}-1)$$

Outline

Man-in-the-Middle Attacks

The Diffie-Hellman key-exchange, without authentication is insecure, because of the malleability of the CDH problem:

Client C | Server S

Man-in-the-Middle Attacks

The Diffie-Hellman key-exchange, without authentication is insecure, because of the malleability of the CDH problem:

Client C		Server S
$x \leftarrow \mathbb{Z}_q$	$\xleftarrow{Send(C, start)}$	

Man-in-the-Middle Attacks

The Diffie-Hellman key-exchange, without authentication is insecure, because of the malleability of the CDH problem:

Client C			Server S
$x \leftarrow \mathbb{Z}_q$	$\xleftarrow{Send(C,start)}$		
$X \leftarrow g^x$	$\longrightarrow X$	$\xrightarrow{Send(S,Xg)}$	$y \leftarrow \mathbb{Z}_q$

The Diffie-Hellman key-exchange, without authentication is insecure, because of the malleability of the CDH problem:

Client C			Server S
$x \leftarrow \mathbb{Z}_q$	$\xleftarrow{Send(C,start)}$		
$X \leftarrow g^x$	$\longrightarrow X$	$\xrightarrow{Send(S,Xg)}$	$y \leftarrow \mathbb{Z}_q$
$sk_C \leftarrow Y^x$	$\xleftarrow{Send(C,Y)}$	$Y \longleftarrow$	$Y \leftarrow g^y$

Man-in-the-Middle Attacks

The Diffie-Hellman key-exchange, without authentication is insecure, because of the malleability of the CDH problem:

Client C			Server S
$x \leftarrow \mathbb{Z}_q$	$\xleftarrow{Send(C,start)}$		
$X \leftarrow g^x$	$\longrightarrow X$	$\xrightarrow{Send(S,Xg)}$	$y \leftarrow \mathbb{Z}_q$
$sk_C \leftarrow Y^x$	$\xleftarrow{Send(C,Y)}$	$Y \longleftarrow$	$Y \leftarrow g^y$
		$\xrightarrow{Reveal(S)}$	

The Diffie-Hellman key-exchange, without authentication is insecure, because of the malleability of the CDH problem:

$$
\begin{array}{r|l c r|l}
\multicolumn{2}{c}{\text{Client } C} & & \multicolumn{2}{c}{\text{Server } S} \\
\hline
x \leftarrow \mathbb{Z}_q & \xleftarrow{Send(C,start)} & & & \\
X \leftarrow g^x & \longrightarrow X & & \xrightarrow{Send(S,Xg)} & y \leftarrow \mathbb{Z}_q \\
sk_C \leftarrow Y^x & \xleftarrow{Send(C,Y)} & & Y \longleftarrow & Y \leftarrow g^y \\
 & & & \xrightarrow{Reveal(S)} & sk_S \leftarrow (Xg)^y \\
 & & & sk_S \longleftarrow & \\
\hline
\end{array}
$$

Man-in-the-Middle Attacks

The Diffie-Hellman key-exchange, without authentication is insecure, because of the malleability of the CDH problem:

$$
\begin{array}{r|cccc|l}
\multicolumn{1}{c}{\text{Client } C} & & & & & \multicolumn{1}{c}{\text{Server } S} \\
x \leftarrow \mathbb{Z}_q & \xleftarrow{\mathit{Send}(C,\mathit{start})} & & & & \\
X \leftarrow g^x & \longrightarrow X & & & \xrightarrow{\mathit{Send}(S,Xg)} & y \leftarrow \mathbb{Z}_q \\
sk_C \leftarrow Y^x & \xleftarrow{\mathit{Send}(C,Y)} & & & Y \longleftarrow & Y \leftarrow g^y \\
 & \xleftarrow{\mathit{Test}(C)} & & & \xrightarrow{\mathit{Reveal}(S)} & sk_S \leftarrow (Xg)^y \\
 & & & & sk_S \longleftarrow &
\end{array}
$$

Man-in-the-Middle Attacks

The Diffie-Hellman key-exchange, without authentication is insecure, because of the malleability of the CDH problem:

$$
\begin{array}{|r|cc|l|}
\multicolumn{1}{c}{\text{Client } C} & & & \multicolumn{1}{c}{\text{Server } S} \\
\hline
x \leftarrow \mathbb{Z}_q & \xleftarrow{\mathit{Send}(C,\mathit{start})} & & \\
X \leftarrow g^x & \longrightarrow X & \xrightarrow{\mathit{Send}(S,Xg)} & y \leftarrow \mathbb{Z}_q \\
sk_C \leftarrow Y^x & \xleftarrow{\mathit{Send}(C,Y)} & Y \longleftarrow & Y \leftarrow g^y \\
 & \xleftarrow{\mathit{Test}(C)} & \xrightarrow{\mathit{Reveal}(S)} & sk_S \leftarrow (Xg)^y \\
 & \longrightarrow \begin{cases} sk_C \\ \$ \end{cases} & sk_S \longleftarrow & \\
\hline
\end{array}
$$

Man-in-the-Middle Attacks

The Diffie-Hellman key-exchange, without authentication is insecure, because of the malleability of the CDH problem:

$$
\begin{array}{|r|ccc|l|}
\multicolumn{1}{c}{\text{Client } C} & & & & \multicolumn{1}{c}{\text{Server } S} \\
\hline
x \leftarrow \mathbb{Z}_q & \xleftarrow{\mathit{Send}(C,\mathit{start})} & & & \\
X \leftarrow g^x & \longrightarrow X & & \xrightarrow{\mathit{Send}(S,Xg)} & y \leftarrow \mathbb{Z}_q \\
sk_C \leftarrow Y^x & \xleftarrow{\mathit{Send}(C,Y)} & & Y \longleftarrow & Y \leftarrow g^y \\
 & \xleftarrow{\mathit{Test}(C)} & & \xrightarrow{\mathit{Reveal}(S)} & sk_S \leftarrow (Xg)^y \\
 & \longrightarrow \left\{ \begin{array}{l} sk_C \\ \$ \end{array} \right. & & sk_S \longleftarrow & \\
\hline
\end{array}
$$

$$sk_S \stackrel{?}{=} sk_C \times Y$$

Man-in-the-Middle Attacks

The Diffie-Hellman key-exchange, without authentication is insecure, because of the malleability of the CDH problem:

Client C			Server S
$x \leftarrow \mathbb{Z}_q$	$\xleftarrow{\quad Send(C,start) \quad}$		
$X \leftarrow g^x$	$\longrightarrow X$	$\xrightarrow{\quad Send(S,Xg) \quad}$	$y \leftarrow \mathbb{Z}_q$
$sk_C \leftarrow Y^x$	$\xleftarrow{\quad Send(C,Y) \quad}$	$Y \longleftarrow$	$Y \leftarrow g^y$
	$\xleftarrow{\quad Test(C) \quad}$	$\xrightarrow{\quad Reveal(S) \quad}$	$sk_S \leftarrow (Xg)^y$
	$\longrightarrow \begin{cases} sk_C \\ \$ \end{cases}$	$sk_S \longleftarrow$	

$$sk_S \stackrel{?}{=} sk_C \times Y$$

No authentication provided!

Authenticated Key-Exchange

Allow two parties to establish a common secret
in an authenticated way

- The session key should only be known to the involved parties
- The session key should be indistinguishable
  from a random string for others

Authenticated Key-Exchange

Allow two parties to establish a common secret
in an authenticated way

- The session key should only be known to the involved parties
- The session key should be indistinguishable
  from a random string for others

Authenticated Key-Exchange

Allow two parties to establish a common secret
in an authenticated way

- The session key should only be known to the involved parties
- The session key should be indistinguishable from a random string for others

Authentication Techniques: PKI

If one assumes a PKI (*public-key infrastructure*),
any user owns a pair of keys, certified by a CA.

By simply signing the flows, one gets an authenticated key-exchange:
$\mathbb{G} = \langle g \rangle$ a group, of prime order q, in which the **DDH** problem is hard

Alice		*Bob*
$x \xleftarrow{R} \mathbb{Z}_q$		$y \xleftarrow{R} \mathbb{Z}_q$
$X = g^x$	$\xrightarrow{Sign_A(B, X)}$	
	$\xleftarrow{Sign_B(A, X, Y)}$	$Y = g^y$

$$Y^x = g^{xy} = X^y$$

Authentication Techniques: PKI

If one assumes a PKI (*public-key infrastructure*),
any user owns a pair of keys, certified by a CA.

By simply signing the flows, one gets an authenticated key-exchange:
$\mathbb{G} = \langle g \rangle$ a group, of prime order q, in which the **DDH** problem is hard

Alice		*Bob*
$x \stackrel{R}{\leftarrow} \mathbb{Z}_q$		$y \stackrel{R}{\leftarrow} \mathbb{Z}_q$
$X = g^x$	$\xrightarrow{Sign_A(B, X)}$	
	$\xleftarrow{Sign_B(A, X, Y)}$	$Y = g^y$

$$Y^x = g^{xy} = X^y$$

Authentication Techniques: PKI

If one assumes a PKI (*public-key infrastructure*),
any user owns a pair of keys, certified by a CA.

By simply signing the flows, one gets an authenticated key-exchange:
$\mathbb{G} = \langle g \rangle$ a group, of prime order q, in which the **DDH** problem is hard

Alice		*Bob*
$x \stackrel{R}{\leftarrow} \mathbb{Z}_q$		$y \stackrel{R}{\leftarrow} \mathbb{Z}_q$
$X = g^x$	$\xrightarrow{Sign_A(B, X)}$	
	$\xleftarrow{Sign_B(A, X, Y)}$	$Y = g^y$

$$Y^x = g^{xy} = X^y$$

Authentication Techniques: PKI

If one assumes a PKI (*public-key infrastructure*),
any user owns a pair of keys, certified by a CA.

By simply signing the flows, one gets an authenticated key-exchange:
$\mathbb{G} = \langle g \rangle$ a group, of prime order q, in which the **DDH** problem is hard

$$\begin{array}{ccc}
\textit{Alice} & & \textit{Bob} \\
x \stackrel{R}{\leftarrow} \mathbb{Z}_q & & y \stackrel{R}{\leftarrow} \mathbb{Z}_q \\
X = g^x & \xrightarrow{Sign_A(B, X)} & \\
& \xleftarrow{Sign_B(A, X, Y)} & Y = g^y \\
& Y^x = g^{xy} = X^y &
\end{array}$$

Signed Diffie-Hellman and DDH

Theorem

*The Signed Diffie-Hellman key exchange is secure under the **DDH** assumption and the security of the signature scheme*

$$\begin{aligned}&\mathbf{Adv}^{\mathsf{ror}}(t, q_{user}, q_{execute}, q_{send}, q_{test})\\ &\qquad \leq q_{user} \times \mathbf{Succ}^{\mathsf{euf-cma}}\left(\begin{array}{l} t + (3q_{execute} + q_{send} + q_{test})\tau_{exp}, \\ q_{send} + q_{execute} \quad (\textit{signing queries}) \end{array}\right)\\ &\qquad + \mathbf{Adv}^{\mathbf{ddh}}(t + (7q_{execute} + 2q_{send} + 4q_{test})\tau_{exp})\end{aligned}$$

Let $\mathcal{A}$ be a RoR-adversary, we use it to break either the signature scheme or the **DDH**.

If the adversary can generate a flow in the name of a user,
we can break the signature scheme:

- We are given a verification key for a user A
- Execute(A, B^i) or Execute(B^i, A): we use the signing oracle
- Send(A, m): we use the signing oracle
- Send$(B, Sign_A(m))$: if not signed by the signing oracle, we reject
- Test(U): as usual

If we reject a valid signature, this signature is a forgery:
all the signatures are oracle generated but with probability less than

$$q_{user} \times \mathbf{Succ}^{\mathrm{euf-cma}} \begin{pmatrix} t + (3q_{execute} + q_{send} + q_{test})\tau_{exp}, \\ q_{send} + q_{execute} \quad (\textit{signing queries}) \end{pmatrix}$$

Signed Diffie-Hellman: Signature

If the adversary can generate a flow in the name of a user,
we can break the signature scheme:

- We are given a verification key for a user A
- $\mathsf{Execute}(A, B^j)$ or $\mathsf{Execute}(B^i, A)$: we use the signing oracle
- $\mathsf{Send}(A, m)$: we use the signing oracle
- $\mathsf{Send}(B, Sign_A(m))$: if not signed by the signing oracle, we reject
- $\mathsf{Test}(U)$: as usual

If we reject a valid signature, this signature is a forgery:
all the signatures are oracle generated but with probability less than

$$q_{user} \times \mathbf{Succ}^{\mathrm{euf-cma}} \begin{pmatrix} t + (3q_{execute} + q_{send} + q_{test})\tau_{exp}, \\ q_{send} + q_{execute} \quad (\textit{signing queries}) \end{pmatrix}$$

Signed Diffie-Hellman: Signature

If the adversary can generate a flow in the name of a user,
we can break the signature scheme:

- We are given a verification key for a user A
- Execute(A, B^j) or Execute(B^i, A): we use the signing oracle
- Send(A, m): we use the signing oracle
- Send$(B, Sign_A(m))$: if not signed by the signing oracle, we reject
- Test(U): as usual

If we reject a valid signature, this signature is a forgery:
all the signatures are oracle generated but with probability less than

$$q_{user} \times \mathbf{Succ}^{\mathrm{euf-cma}} \begin{pmatrix} t + (3q_{execute} + q_{send} + q_{test})\tau_{exp}, \\ q_{send} + q_{execute} \quad (signing\ queries) \end{pmatrix}$$

Signed Diffie-Hellman: Signature

If the adversary can generate a flow in the name of a user,
we can break the signature scheme:

- We are given a verification key for a user A
- $\mathsf{Execute}(A, B^j)$ or $\mathsf{Execute}(B^i, A)$: we use the signing oracle
- $\mathsf{Send}(A, m)$: we use the signing oracle
- $\mathsf{Send}(B, Sign_A(m))$: if not signed by the signing oracle, we reject
- $\mathsf{Test}(U)$: as usual

If we reject a valid signature, this signature is a forgery:
all the signatures are oracle generated but with probability less than

$$q_{user} \times \mathbf{Succ}^{\mathrm{euf-cma}} \left(\begin{array}{ll} t + (3q_{execute} + q_{send} + q_{test})\tau_{exp}, & \\ q_{send} + q_{execute} & (signing\ queries) \end{array} \right)$$

Signed Diffie-Hellman: Signature

If the adversary can generate a flow in the name of a user,
we can break the signature scheme:

- We are given a verification key for a user A
- $\mathsf{Execute}(A, B^j)$ or $\mathsf{Execute}(B^i, A)$: we use the signing oracle
- $\mathsf{Send}(A, m)$: we use the signing oracle
- $\mathsf{Send}(B, Sign_A(m))$: if not signed by the signing oracle, we reject
- $\mathsf{Test}(U)$: as usual

If we reject a valid signature, this signature is a forgery:
all the signatures are oracle generated but with probability less than

$$q_{user} \times \mathbf{Succ}^{\mathrm{euf-cma}} \begin{pmatrix} t + (3q_{execute} + q_{send} + q_{test})\tau_{exp}, \\ q_{send} + q_{execute} \quad (signing\ queries) \end{pmatrix}$$

If the adversary can generate a flow in the name of a user,
we can break the signature scheme:

- We are given a verification key for a user A
- $\mathsf{Execute}(A, B^j)$ or $\mathsf{Execute}(B^i, A)$: we use the signing oracle
- $\mathsf{Send}(A, m)$: we use the signing oracle
- $\mathsf{Send}(B, Sign_A(m))$: if not signed by the signing oracle, we reject
- $\mathsf{Test}(U)$: as usual

If we reject a valid signature, this signature is a forgery:
all the signatures are oracle generated but with probability less than

$$q_{user} \times \mathbf{Succ}^{\mathrm{euf-cma}} \left(\begin{array}{l} t + (3q_{execute} + q_{send} + q_{test})\tau_{exp}, \\ q_{send} + q_{execute} \quad (\textit{signing queries}) \end{array} \right)$$

Signed Diffie-Hellman: Signature

If the adversary can generate a flow in the name of a user,
we can break the signature scheme:

- We are given a verification key for a user A
- Execute(A, B^j) or Execute(B^i, A): we use the signing oracle
- Send(A, m): we use the signing oracle
- Send($B, Sign_A(m)$): if not signed by the signing oracle, we reject
- Test(U): as usual

If we reject a valid signature, this signature is a forgery:
all the signatures are oracle generated but with probability less than

$$q_{user} \times \mathbf{Succ}^{\text{euf}-\text{cma}} \left(\begin{array}{l} t + (3q_{execute} + q_{send} + q_{test})\tau_{exp}, \\ q_{send} + q_{execute} \quad (\textit{signing queries}) \end{array} \right)$$

Signed Diffie-Hellman: DDH

Given a triple $(X = g^x, Y = g^y, Z = g^z)$, we can derive a list of triples:

$$X_i = g^{x_i} = X \cdot g^{\alpha_i} \qquad Z_{i,j} = g^{z_{i,j}} = Z^{\beta_{i,j}} \cdot X^{\gamma_{i,j}} \cdot Y^{\alpha_i \beta_{i,j}} \cdot g^{\alpha_i \gamma_{i,j}}$$

$$Y_{i,j} = g^{y_{i,j}} = Y^{\beta_{i,j}} \cdot g^{\gamma_{i,j}}$$

We thus have

$$x_i = x + \alpha_i \quad y_{i,j} = y\beta_{i,j} + \gamma_{i,j} \quad z_{i,j} = x_i y_i + (z - xy)\beta_{i,j}$$

If (X, Y, Z) is a Diffie-Hellman triple (*i.e.*, $z = xy$),
all the triples are random and independent Diffie-Hellman triples

Signed Diffie-Hellman: DDH

Given a triple $(X = g^x, Y = g^y, Z = g^z)$, we can derive a list of triples:

$$X_i = g^{x_i} = X \cdot g^{\alpha_i} \qquad Z_{i,j} = g^{z_{i,j}} = Z^{\beta_{i,j}} \cdot X^{\gamma_{i,j}} \cdot Y^{\alpha_i \beta_{i,j}} \cdot g^{\alpha_i \gamma_{i,j}}$$
$$Y_{i,j} = g^{y_{i,j}} = Y^{\beta_{i,j}} \cdot g^{\gamma_{i,j}}$$

We thus have

$$x_i = x + \alpha_i \quad y_{i,j} = y\beta_{i,j} + \gamma_{i,j} \quad z_{i,j} = x_i y_i + (z - xy)\beta_{i,j}$$

If (X, Y, Z) is a Diffie-Hellman triple (*i.e.*, $z = xy$),
all the triples are random and independent Diffie-Hellman triples

Signed Diffie-Hellman: DDH

Given a triple $(X = g^x, Y = g^y, Z = g^z)$, we can derive a list of triples:

$$X_i = g^{x_i} = X \cdot g^{\alpha_i} \qquad Z_{i,j} = g^{z_{i,j}} = Z^{\beta_{i,j}} \cdot X^{\gamma_{i,j}} \cdot Y^{\alpha_i \beta_{i,j}} \cdot g^{\alpha_i \gamma_{i,j}}$$
$$Y_{i,j} = g^{y_{i,j}} = Y^{\beta_{i,j}} \cdot g^{\gamma_{i,j}}$$

We thus have

$$x_i = x + \alpha_i \quad y_{i,j} = y\beta_{i,j} + \gamma_{i,j} \quad z_{i,j} = x_i y_i + (z - xy)\beta_{i,j}$$

If (X, Y, Z) is a Diffie-Hellman triple (*i.e.*, $z = xy$),
all the triples are random and independent Diffie-Hellman triples

Signed Diffie-Hellman and DDH

Given a triple $(X = g^x, Y = g^y, Z = g^z)$

$$x_i = x + \alpha_i \quad y_{i,j} = y\beta_{i,j} + \gamma_{i,j} \quad z_{i,j} = x_i y_i + (z - xy)\beta_{i,j}$$

For any random list of triples $(X_i = g^{x_i}, Y_{i,j} = g^{y_{i,j}}, Z_{i,j} = g^{z_{i,j}})$,
if $d = z - xy \neq 0$, we can define

$$\alpha_i = x_i - x \qquad \beta_{i,j} = (z_{i,j} - x_i y_{i,j})/d \qquad \gamma_{i,j} = y_{i,j} - y\beta_{i,j}$$

If (X, Y, Z) is not a Diffie-Hellman triple (*i.e.*, $z \neq xy$),
all the triples are independent random triples

Signed Diffie-Hellman and DDH

Given a triple $(X = g^x, Y = g^y, Z = g^z)$

$$x_i = x + \alpha_i \quad y_{i,j} = y\beta_{i,j} + \gamma_{i,j} \quad z_{i,j} = x_i y_i + (z - xy)\beta_{i,j}$$

For any random list of triples $(X_i = g^{x_i}, Y_{i,j} = g^{y_{i,j}}, Z_{i,j} = g^{z_{i,j}})$,
if $d = z - xy \neq 0$, we can define

$$\alpha_i = x_i - x \qquad \beta_{i,j} = (z_{i,j} - x_i y_{i,j})/d \qquad \gamma_{i,j} = y_{i,j} - y\beta_{i,j}$$

If (X, Y, Z) is not a Diffie-Hellman triple (*i.e.*, $z \neq xy$),
all the triples are independent random triples

Signed Diffie-Hellman: DDH

We now assume that all the flows are oracle generated

- We are given a triple (X, Y, Z)
- Execute(A^i, B^j): we use a fresh X_i but $Y' = g^{y'}$ for a known y'
  We can compute Z'
- Send$(A, Start)$: we use a fresh X_i
- Send$(B, Sign_A(B, X))$: if valid, we look for $X_i = X$, use a fresh $Y_{i,j}$
  The associated key is $Z_{i,j}$
- Send$(A, Sign_B(A, X, Y))$: if valid, we look for $X_i = X$, $Y_{i,j} = Y$.
  The associated key is $Z_{i,j}$
- Test(U): the associated key is outputted

Signed Diffie-Hellman: DDH

We now assume that all the flows are oracle generated

- We are given a triple (X, Y, Z)
- Execute(A^i, B^j): we use a fresh X_i but $Y' = g^{y'}$ for a known y'
  We can compute Z'
- Send$(A, Start)$: we use a fresh X_i
- Send$(B, Sign_A(B, X))$: if valid, we look for $X_i = X$, use a fresh $Y_{i,j}$
  The associated key is $Z_{i,j}$
- Send$(A, Sign_B(A, X, Y))$: if valid, we look for $X_i = X$, $Y_{i,j} = Y$.
  The associated key is $Z_{i,j}$
- Test(U): the associated key is outputted

Signed Diffie-Hellman: DDH

We now assume that all the flows are oracle generated

- We are given a triple (X, Y, Z)
- Execute(A^i, B^j): we use a fresh X_i but $Y' = g^{y'}$ for a known y'
  We can compute Z'
- Send$(A, Start)$: we use a fresh X_i
- Send$(B, Sign_A(B, X))$: if valid, we look for $X_i = X$, use a fresh $Y_{i,j}$
  The associated key is $Z_{i,j}$
- Send$(A, Sign_B(A, X, Y))$: if valid, we look for $X_i = X$, $Y_{i,j} = Y$.
  The associated key is $Z_{i,j}$
- Test(U): the associated key is outputted

Signed Diffie-Hellman: DDH

We now assume that all the flows are oracle generated

- We are given a triple (X, Y, Z)
- Execute(A^i, B^j): we use a fresh X_i but $Y' = g^{y'}$ for a known y'
  We can compute Z'
- Send$(A, Start)$: we use a fresh X_i
- Send$(B, Sign_A(B, X))$: if valid, we look for $X_i = X$, use a fresh $Y_{i,j}$
  The associated key is $Z_{i,j}$
- Send$(A, Sign_B(A, X, Y))$: if valid, we look for $X_i = X$, $Y_{i,j} = Y$.
  The associated key is $Z_{i,j}$
- Test(U): the associated key is outputted

Signed Diffie-Hellman: DDH

We now assume that all the flows are oracle generated

- We are given a triple (X, Y, Z)
- Execute(A^i, B^j): we use a fresh X_i but $Y' = g^{y'}$ for a known y'
  We can compute Z'
- Send$(A, Start)$: we use a fresh X_i
- Send$(B, Sign_A(B, X))$: if valid, we look for $X_i = X$, use a fresh $Y_{i,j}$
  The associated key is $Z_{i,j}$
- Send$(A, Sign_B(A, X, Y))$: if valid, we look for $X_i = X$, $Y_{i,j} = Y$.
  The associated key is $Z_{i,j}$
- Test(U): the associated key is outputted

Signed Diffie-Hellman: DDH

We now assume that all the flows are oracle generated

- We are given a triple (X, Y, Z)
- Execute(A^i, B^j): we use a fresh X_i but $Y' = g^{y'}$ for a known y'
  We can compute Z'
- Send$(A, Start)$: we use a fresh X_i
- Send$(B, Sign_A(B, X))$: if valid, we look for $X_i = X$, use a fresh $Y_{i,j}$
  The associated key is $Z_{i,j}$
- Send$(A, Sign_B(A, X, Y))$: if valid, we look for $X_i = X$, $Y_{i,j} = Y$.
  The associated key is $Z_{i,j}$
- Test(U): the associated key is outputted

If the triple (X, Y, Z) is a DDH triple, we are in the Real case since all the keys are correctly computed

If the triple (X, Y, Z) is not a DDH triple, we are in the Random case since all the keys are independent random values

Signed Diffie-Hellman: DDH

If the triple (X, Y, Z) is a DDH triple, we are in the Real case since all the keys are correctly computed

If the triple (X, Y, Z) is not a DDH triple, we are in the Random case since all the keys are independent random values

Authentication Techniques: Symmetric

Users share a common secret k of high entropy
A MAC can be used for authenticating the flows.

$$\begin{array}{ccc}
\text{Alice} & & \text{Bob} \\
x \stackrel{R}{\leftarrow} \mathbb{Z}_q & & y \stackrel{R}{\leftarrow} \mathbb{Z}_q \\
X = g^x & \xrightarrow{MAC_k(A, B, X)} & \\
 & \xleftarrow{MAC_k(B, A, X, Y)} & Y = g^y \\
 & Y^x = g^{xy} = X^y &
\end{array}$$

The same security result holds

Authentication Techniques: Symmetric

Users share a common secret k of high entropy
A MAC can be used for authenticating the flows.

$$\begin{array}{ccc}
\textit{Alice} & & \textit{Bob} \\
x \stackrel{R}{\leftarrow} \mathbb{Z}_q & & y \stackrel{R}{\leftarrow} \mathbb{Z}_q \\
X = g^x & \xrightarrow{MAC_k(A, B, X)} & \\
 & \xleftarrow{MAC_k(B, A, X, Y)} & Y = g^y \\
 & Y^x = g^{xy} = X^y &
\end{array}$$

The same security result holds

Authentication Techniques: Symmetric

Users share a common secret k of high entropy
A MAC can be used for authenticating the flows.

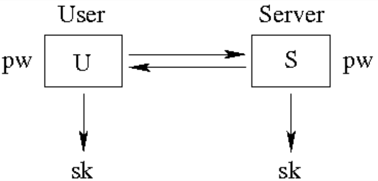
$$
\begin{array}{ccc}
\textit{Alice} & & \textit{Bob} \\
x \stackrel{R}{\leftarrow} \mathbb{Z}_q & & y \stackrel{R}{\leftarrow} \mathbb{Z}_q \\
X = g^x & \xrightarrow{MAC_k(A, B, X)} & \\
 & \xleftarrow{MAC_k(B, A, X, Y)} & Y = g^y \\
 & Y^x = g^{xy} = X^y &
\end{array}
$$

The same security result holds

Password-Based AKE

Realistic: Real-life applications usually rely on weak passwords

Convenient to use: Users do not need to store a long secret

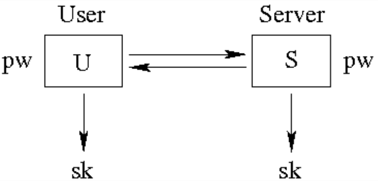


Subject to on-line dictionary attacks:
Non-negligible probability of success due to the small dictionary

On-line Dictionary Attacks

- the adversary chooses a password pw
- tries to authenticate to the server
- in case of failure, it starts over

Password-Based AKE



Realistic: Real-life applications usually rely on weak passwords

Convenient to use: Users do not need to store a long secret

Subject to on-line dictionary attacks:
Non-negligible probability of success due to the small dictionary

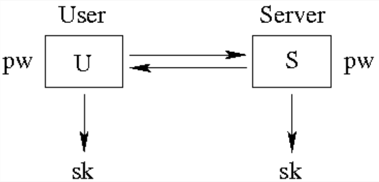
On-line Dictionary Attacks

- the adversary chooses a password pw
- tries to authenticate to the server
- in case of failure, it starts over

Password-Based AKE

Realistic: Real-life applications usually rely on weak passwords

Convenient to use: Users do not need to store a long secret



Subject to on-line dictionary attacks:

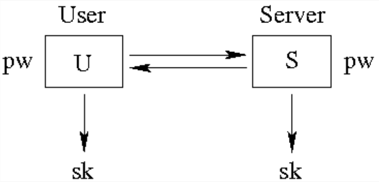
Non-negligible probability of success due to the small dictionary

On-line Dictionary Attacks

- the adversary chooses a password pw
- tries to authenticate to the server
- in case of failure, it starts over

Realistic: Real-life applications usually rely on weak passwords

Convenient to use: Users do not need to store a long secret



Subject to on-line dictionary attacks:

Non-negligible probability of success due to the small dictionary

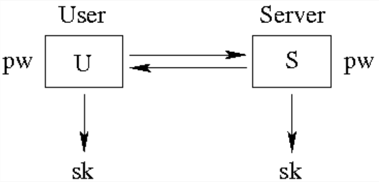
On-line Dictionary Attacks

- the adversary chooses a password pw
- tries to authenticate to the server
- in case of failure, it starts over

Password-Based AKE

Realistic: Real-life applications usually rely on weak passwords

Convenient to use: Users do not need to store a long secret



Subject to on-line dictionary attacks:

Non-negligible probability of success due to the small dictionary

On-line Dictionary Attacks

- the adversary chooses a password pw
- tries to authenticate to the server
- in case of failure, it starts over

Find-then-Guess vs. Real-or-Random

Definition

A PAKE scheme is Semantically Secure if the best attack is the *online dictionary attack*:

$$\mathbf{Adv}^{\mathsf{ftg}}(t) \leq q_{send}/|D| + \mathsf{negl}()$$

or even better

$$\mathbf{Adv}^{\mathsf{ror}}(t) \leq q_{send}/|D| + \mathsf{negl}()$$

We cannot get better than the former, but we can expect the latter.

Find-then-Guess vs. Real-or-Random

Definition

A PAKE scheme is Semantically Secure if the best attack is the *online dictionary attack*:

$$\mathbf{Adv}^{\mathsf{ftg}}(t) \leq q_{send}/|D| + \mathsf{negl}()$$

or even better

$$\mathbf{Adv}^{\mathsf{ror}}(t) \leq q_{send}/|D| + \mathsf{negl}()$$

We cannot get better than the former, but we can expect the latter.

Find-then-Guess vs. Real-or-Random

Definition

A PAKE scheme is Semantically Secure if the best attack is the *online dictionary attack*:

$$\mathbf{Adv}^{\mathsf{ftg}}(t) \leq q_{send}/|D| + \mathsf{negl}()$$

or even better

$$\mathbf{Adv}^{\mathsf{ror}}(t) \leq q_{send}/|D| + \mathsf{negl}()$$

We cannot get better than the former, but we can expect the latter.

Outline

Mutual Authentication

The Semantic Security tells that the session key should be indistinguishable from a random string for others

What about the case where the key is random for everybody, and then, no key is shared at all!

Client Authentication

If the server accepts a key, then a client has the material to compute the same key.

Mutual Authentication

If a party accepts a key, then its partner has the material to compute the same key.

Mutual Authentication

The Semantic Security tells that the session key should be indistinguishable from a random string for others

What about the case where the key is random for everybody, and then, no key is shared at all!

Client Authentication

If the server accepts a key, then a client has the material to compute the same key.

Mutual Authentication

If a party accepts a key, then its partner has the material to compute the same key.

Mutual Authentication

The Semantic Security tells that the session key should be indistinguishable from a random string for others

What about the case where the key is random for everybody, and then, no key is shared at all!

Client Authentication

If the server accepts a key, then a client has the material to compute the same key.

Mutual Authentication

If a party accepts a key, then its partner has the material to compute the same key.

Mutual Authentication

The Semantic Security tells that the session key should be indistinguishable from a random string for others

What about the case where the key is random for everybody, and then, no key is shared at all!

Client Authentication

If the server accepts a key, then a client has the material to compute the same key.

Mutual Authentication

If a party accepts a key, then its partner has the material to compute the same key.

Explicit Authentication: Game-based Definition

The session-ID should determine the session-key (not in a computable way): this formally determines partnership.

Definition (Client Authentication)

The attacker wins the client authentication game if a server instance terminates, without exactly one accepting client partner.

Flags

- the flag Accept means that
  - the player has enough material to compute the key
- the flag Terminate means that
  - the player thinks that its partners has accepted

Explicit Authentication: Game-based Definition

The session-ID should determine the session-key (not in a computable way): this formally determines partnership.

Definition (Client Authentication)

The attacker wins the client authentication game if a server instance terminates, without exactly one accepting client partner.

Flags

- the flag Accept means that
  the player has enough material to compute the key
- the flag Terminate means that
  the player thinks that its partners has accepted

Explicit Authentication: Game-based Definition

The session-ID should determine the session-key (not in a computable way): this formally determines partnership.

Definition (Client Authentication)

The attacker wins the client authentication game if a server instance terminates, without exactly one accepting client partner.

Flags

- the flag Accept means that
  the player has enough material to compute the key
- the flag Terminate means that
  the player thinks that its partners has accepted

Explicit Authentication: Game-based Definition

The session-ID should determine the session-key (not in a computable way): this formally determines partnership.

Definition (Client Authentication)

The attacker wins the client authentication game if a server instance terminates, without exactly one accepting client partner.

Flags

- the flag Accept means that

  the player has enough material to compute the key
- the flag Terminate means that

  the player thinks that its partners has accepted

Corruption

In the previous model, all the players are honest,
and the adversary is not registered (no signing keys)

Wa can add a Corrupt query,
which gives the long-term secret to the adversary

Forward Secrecy

The security of the current session key is preserved even if the long-term secrets (authentication means) are exposed in the future

Corruption

In the previous model, all the players are honest,
and the adversary is not registered (no signing keys)

Wa can add a Corrupt query,
which gives the long-term secret to the adversary

Forward-Secrecy

The security of the current session key is preserved even if the long-term secrets (authentication means) are exposed in the future

Simulation-based Security

Outline

Game-based Security

Simulation-based Security

- Simulation-based Security
- Universal Composability
- Password-based Key Exchange

Encrypted Key Exchange

Conclusion

Ideal Functionality – Real Protocol

Real Protocol

The real protocol $\mathcal{P}$ is run by players $P_1, \ldots, P_n$, with their own private inputs $x_1, \ldots, x_n$. After interactions, they get outputs $y_1, \ldots, y_n$.

Ideal Functionality

An ideal function $\mathcal{F}$ is defined:

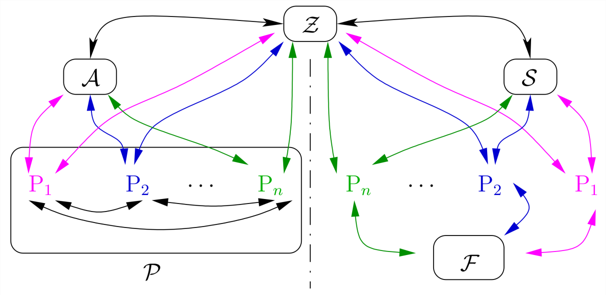
- it takes as input $x_1, \ldots, x_n$, the private information of each players,
- and outputs $y_1, \ldots, y_n$, given privately to each player.

The players get their results, without interacting:
this is a "by definition" secure primitive.

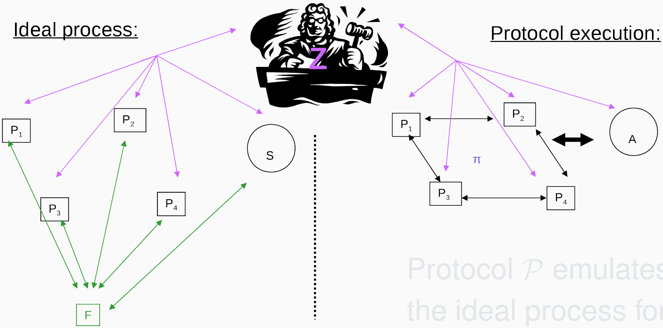
Simulator

For any environment $\mathcal{Z}$, for any adversary $\mathcal{A}$,
there exists a simulator $\mathcal{S}$ so that, the view of $\mathcal{Z}$ is the same for

- $\mathcal{A}$ attacking the real protocol
- $\mathcal{S}$ attacking the ideal functionality



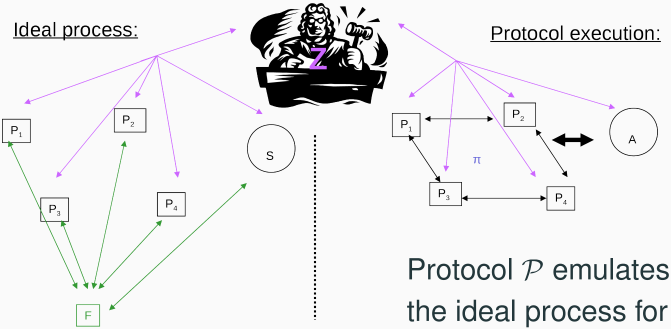
Emulation



Protocol $\mathcal{P}$ emulates the ideal process for $\mathcal{F}$ if

- for any adversary $\mathcal{A}$
- there exists a simulator $\mathcal{S}$
- such that no environment $\mathcal{Z}$ can make the difference between the ideal process and the protocol execution

Emulation



Protocol $\mathcal{P}$ emulates the ideal process for $\mathcal{F}$ if

- for any adversary $\mathcal{A}$
- there exists a simulator $\mathcal{S}$
- such that no environment $\mathcal{Z}$ can make the difference between the ideal process and the protocol execution

Emulation

Protocol $\mathcal{P}$ emulates the ideal process for $\mathcal{F}$ if

- for any adversary $\mathcal{A}$
- there exists a simulator $\mathcal{S}$
- such that for every environment $\mathcal{Z}$

the views are indistinguishable:

$$\forall \mathcal{A}, \exists \mathcal{S}, \forall \mathcal{Z}, \mathit{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \approx \mathit{EXEC}_{\mathcal{P},\mathcal{A},\mathcal{Z}}$$

Equivalent Formulations

$$\forall \mathcal{A}, \exists \mathcal{S}, \forall \mathcal{Z}, \mathit{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \approx \mathit{EXEC}_{\mathcal{P},\mathcal{A},\mathcal{Z}}$$

$$\forall \mathcal{A}, \forall \mathcal{Z}, \exists \mathcal{S}, \mathit{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \approx \mathit{EXEC}_{\mathcal{P},\mathcal{A},\mathcal{Z}}$$

$$\exists \mathcal{S}, \forall \mathcal{Z}, \mathit{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \approx \mathit{EXEC}_{\mathcal{P},\mathcal{A}_d,\mathcal{Z}}$$

where $\mathcal{A}_d$ is the dummy adversary: under the control of the environment (forwards every input/output).

Equivalent Formulations

$$\forall \mathcal{A}, \exists \mathcal{S}, \forall \mathcal{Z}, \mathit{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \approx \mathit{EXEC}_{\mathcal{P},\mathcal{A},\mathcal{Z}}$$

$$\forall \mathcal{A}, \forall \mathcal{Z}, \exists \mathcal{S}, \mathit{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \approx \mathit{EXEC}_{\mathcal{P},\mathcal{A},\mathcal{Z}}$$

$$\exists \mathcal{S}, \forall \mathcal{Z}, \mathit{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \approx \mathit{EXEC}_{\mathcal{P},\mathcal{A}_d,\mathcal{Z}}$$

where $\mathcal{A}_d$ is the dummy adversary: under the control of the environment (forwards every input/output).

Equivalent Formulations

$$\forall \mathcal{A}, \exists \mathcal{S}, \forall \mathcal{Z}, \mathit{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \approx \mathit{EXEC}_{\mathcal{P},\mathcal{A},\mathcal{Z}}$$

$$\forall \mathcal{A}, \forall \mathcal{Z}, \exists \mathcal{S}, \mathit{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \approx \mathit{EXEC}_{\mathcal{P},\mathcal{A},\mathcal{Z}}$$

$$\exists \mathcal{S}, \forall \mathcal{Z}, \mathit{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \approx \mathit{EXEC}_{\mathcal{P},\mathcal{A}_d,\mathcal{Z}}$$

where $\mathcal{A}_d$ is the dummy adversary: under the control of the environment (forwards every input/output).

Equivalent Formulations

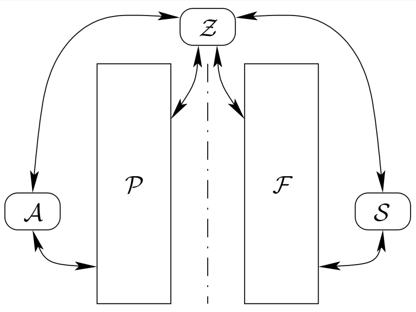
$$\forall \mathcal{A}, \exists \mathcal{S}, \forall \mathcal{Z}, \mathit{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \approx \mathit{EXEC}_{\mathcal{P},\mathcal{A},\mathcal{Z}}$$

$$\forall \mathcal{A}, \forall \mathcal{Z}, \exists \mathcal{S}, \mathit{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \approx \mathit{EXEC}_{\mathcal{P},\mathcal{A},\mathcal{Z}}$$

$$\exists \mathcal{S}, \forall \mathcal{Z}, \mathit{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \approx \mathit{EXEC}_{\mathcal{P},\mathcal{A}_d,\mathcal{Z}}$$

where $\mathcal{A}_d$ is the dummy adversary: under the control of the environment (forwards every input/output).

Security



- Everything that the adversary $\mathcal{A}$ can do against $\mathcal{P}$ can be done by the simulator $\mathcal{S}$ against $\mathcal{F}$
- But the ideal functionality $\mathcal{F}$ is perfectly secure: nothing can be done against $\mathcal{F}$

Then, nothing can be done against $\mathcal{P}$

Outline

Game-based Security

Simulation-based Security

- Simulation-based Security
- Universal Composability
- Password-based Key Exchange

Encrypted Key Exchange

Conclusion

Implications of UC

Can design and analyze protocols in a modular way:

- Divide a given task $\mathcal{F}$ into sub-tasks $\mathcal{F}_1, \ldots, \mathcal{F}_n$
  $\mathcal{F}$ is equivalent to $\mathcal{F}_1 \cup \mathcal{F}_2 \cup \mathcal{F}_3 \cup \mathcal{F}_4$
- Construct protocols $\pi_1, \ldots, \pi_n$ emulating $\mathcal{F}_1, \ldots, \mathcal{F}_n$
- Combine them into a protocol π
- Composition theorem: π emulates $\mathcal{F}$

Implications of UC

Can design and analyze protocols in a modular way:

- Divide a given task $\mathcal{F}$ into sub-tasks $\mathcal{F}_1, \ldots, \mathcal{F}_n$
  $\mathcal{F}$ is equivalent to $\mathcal{F}_1 \cup \mathcal{F}_2 \cup \mathcal{F}_3 \cup \mathcal{F}_4$
- Construct protocols $\pi_1, \ldots, \pi_n$ emulating $\mathcal{F}_1, \ldots, \mathcal{F}_n$
- Combine them into a protocol π
- Composition theorem: π emulates $\mathcal{F}$

Implications of UC

Can design and analyze protocols in a modular way:

- Divide a given task $\mathcal{F}$ into sub-tasks $\mathcal{F}_1, \ldots, \mathcal{F}_n$
  $\mathcal{F}$ is equivalent to $\mathcal{F}_1 \cup \mathcal{F}_2 \cup \mathcal{F}_3 \cup \mathcal{F}_4$
- Construct protocols $\pi_1, \ldots, \pi_n$ emulating $\mathcal{F}_1, \ldots, \mathcal{F}_n$
- Combine them into a protocol π
- Composition theorem: π emulates $\mathcal{F}$

Implications of UC

Can design and analyze protocols in a modular way:

- Divide a given task $\mathcal{F}$ into sub-tasks $\mathcal{F}_1, \ldots, \mathcal{F}_n$
  $\mathcal{F}$ is equivalent to $\mathcal{F}_1 \cup \mathcal{F}_2 \cup \mathcal{F}_3 \cup \mathcal{F}_4$
- Construct protocols $\pi_1, \ldots, \pi_n$ emulating $\mathcal{F}_1, \ldots, \mathcal{F}_n$
- Combine them into a protocol π
- Composition theorem: π emulates $\mathcal{F}$

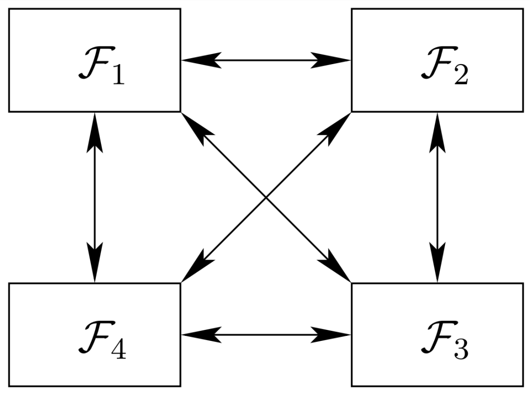
Implications of UC

Can design and analyze protocols in a modular way:

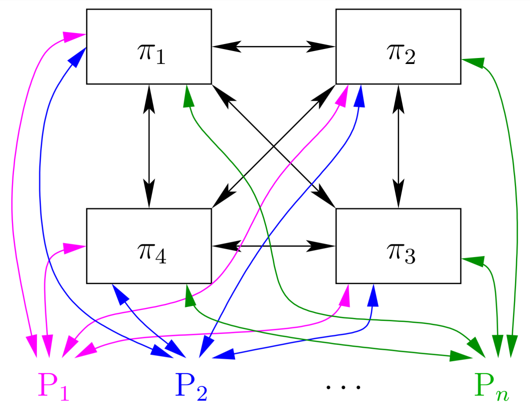
- Divide a given task $\mathcal{F}$ into sub-tasks $\mathcal{F}_1, \ldots, \mathcal{F}_n$
  $\mathcal{F}$ is equivalent to $\mathcal{F}_1 \cup \mathcal{F}_2 \cup \mathcal{F}_3 \cup \mathcal{F}_4$
- Construct protocols $\pi_1, \ldots, \pi_n$ emulating $\mathcal{F}_1, \ldots, \mathcal{F}_n$
- Combine them into a protocol π
- Composition theorem: π emulates $\mathcal{F}$

Can be done concurrently and in parallel

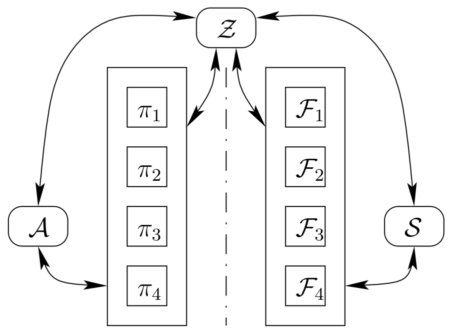
Composition of Ideal Functionalities



Composition of Real Protocols



UC Security



Theorem (Universal Composition)

If each ideal functionality $\mathcal{F}_i$ is emulated by π_i, then the composition of the π_i's emulates the composition of the $\mathcal{F}_i$'s

Outline

Ideal Functionality of PAKE

Session key:

- no corrupted players, same passwords
  $\Rightarrow$ same key sk uniformly chosen
- no corrupted players, different passwords
  $\Rightarrow$ independent keys uniformly chosen
- a corrupted player
  $\Rightarrow$ key chosen by the adversary
- correct password guess
  $\Rightarrow$ key chosen by the adversary
- incorrect password guess
  $\Rightarrow$ independent keys uniformly chosen

Ideal Functionality of PAKE

Session key:

- no corrupted players, same passwords
  ⇒ same key *sk* uniformly chosen
- no corrupted players, different passwords
  ⇒ independent keys uniformly chosen
- a corrupted player
  ⇒ key chosen by the adversary
- correct password guess
  ⇒ key chosen by the adversary
- incorrect password guess
  ⇒ independent keys uniformly chosen

Ideal Functionality of PAKE

Session key:

- no corrupted players, same passwords
  $\Rightarrow$ same key *sk* uniformly chosen
- no corrupted players, different passwords
  $\Rightarrow$ independent keys uniformly chosen
- a corrupted player
  $\Rightarrow$ key chosen by the adversary
- correct password guess
  $\Rightarrow$ key chosen by the adversary
- incorrect password guess
  $\Rightarrow$ independent keys uniformly chosen

Ideal Functionality of PAKE

Session key:

- no corrupted players, same passwords
  $\Rightarrow$ same key *sk* uniformly chosen
- no corrupted players, different passwords
  $\Rightarrow$ independent keys uniformly chosen
- a corrupted player
  $\Rightarrow$ key chosen by the adversary
- correct password guess
  $\Rightarrow$ key chosen by the adversary
- incorrect password guess
  $\Rightarrow$ independent keys uniformly chosen

Ideal Functionality of PAKE

Session key:

- no corrupted players, same passwords
  $\Rightarrow$ same key *sk* uniformly chosen
- no corrupted players, different passwords
  $\Rightarrow$ independent keys uniformly chosen
- a corrupted player
  $\Rightarrow$ key chosen by the adversary
- correct password guess
  $\Rightarrow$ key chosen by the adversary
- incorrect password guess
  $\Rightarrow$ independent keys uniformly chosen

Ideal Functionality of PAKE

Queries

- `NewSession` = a player initializes the protocol
  The passwords are chosen by the environment.
- `TestPwd` = $\mathcal{A}$ attempts to guess a password (one per session)
  In case of correct guess, the adversary is allowed to choose the session key.
  $\Rightarrow$ models the on-line dictionary attacks
- `NewKey` = $\mathcal{A}$ asks for the key *sk* to be delivered to a player
  The key *sk* is ignored except in case of corruption or correct password guess.

Ideal Functionality of PAKE

Queries

- `NewSession` = a player initializes the protocol
  The passwords are chosen by the environment.
- `TestPwd` = $\mathcal{A}$ attempts to guess a password (one per session)
  In case of correct guess, the adversary is allowed to choose the session key.
  $\Rightarrow$ models the on-line dictionary attacks
- `NewKey` = $\mathcal{A}$ asks for the key *sk* to be delivered to a player
  The key *sk* is ignored except in case of corruption or correct password guess.

Ideal Functionality of PAKE

Queries

- `NewSession` = a player initializes the protocol
  The passwords are chosen by the environment.
- `TestPwd` = $\mathcal{A}$ attempts to guess a password (one per session)
  In case of correct guess, the adversary is allowed to choose the session key.
  $\Rightarrow$ models the on-line dictionary attacks
- `NewKey` = $\mathcal{A}$ asks for the key *sk* to be delivered to a player
  The key *sk* is ignored except in case of corruption or correct password guess.

Ideal Functionality of PAKE

Improvements

- No assumption on the relations between the passwords of the different players (can be different, identical, or the same for different protocols)
- It provides forward secrecy, since corruption of players is available

Ideal Functionality of PAKE

Improvements

- No assumption on the relations between the passwords of the different players (can be different, identical, or the same for different protocols)
- It provides forward secrecy, since corruption of players is available

Encrypted Key Exchange

Outline

Setup

- The arithmetic is in a finite cyclic group $\mathbb{G} = \langle g \rangle$
- of order a ℓ-bit prime number q
- Hash functions

$$\mathcal{H}_0 : \{0,1\}^\star \to \{0,1\}^{\ell_0} \qquad \mathcal{H}_1 : \{0,1\}^\star \to \{0,1\}^{\ell_1}$$

- A block cipher $(\mathcal{E}_k, \mathcal{D}_k)$ where $k \in$ Password, onto $\mathbb{G}$.
- $\bar{\mathbb{G}} = \mathbb{G} \backslash \{1\}$, thus $\bar{\mathbb{G}} = \{g^x \mid x \in \mathbb{Z}_q^\star\}$.

Client and server initially share a low-quality password *pw*, uniformly drawn from the dictionary Password.

The session-key space **SK** is $\{0,1\}^{\ell_0}$ equipped with a uniform distribution.

Setup

- The arithmetic is in a finite cyclic group $\mathbb{G} = \langle g \rangle$
- of order a ℓ-bit prime number q
- Hash functions

$$\mathcal{H}_0 : \{0,1\}^\star \to \{0,1\}^{\ell_0} \qquad \mathcal{H}_1 : \{0,1\}^\star \to \{0,1\}^{\ell_1}$$

- A block cipher $(\mathcal{E}_k, \mathcal{D}_k)$ where $k \in$ Password, onto $\mathbb{G}$.
- $\bar{\mathbb{G}} = \mathbb{G} \backslash \{1\}$, thus $\bar{\mathbb{G}} = \{g^x \mid x \in \mathbb{Z}_q^\star\}$.

Client and server initially share a low-quality password *pw*, uniformly drawn from the dictionary Password.

The session-key space **SK** is $\{0,1\}^{\ell_0}$ equipped with a uniform distribution.

Setup

- The arithmetic is in a finite cyclic group $\mathbb{G} = \langle g \rangle$
- of order a ℓ-bit prime number q
- Hash functions

$$\mathcal{H}_0 : \{0,1\}^\star \to \{0,1\}^{\ell_0} \qquad \mathcal{H}_1 : \{0,1\}^\star \to \{0,1\}^{\ell_1}$$

- A block cipher $(\mathcal{E}_k, \mathcal{D}_k)$ where $k \in$ Password, onto $\mathbb{G}$.
- $\bar{\mathbb{G}} = \mathbb{G} \backslash \{1\}$, thus $\bar{\mathbb{G}} = \{g^x \mid x \in \mathbb{Z}_q^\star\}$.

Client and server initially share a low-quality password *pw*, uniformly drawn from the dictionary Password.

The session-key space **SK** is $\{0,1\}^{\ell_0}$ equipped with a uniform distribution.

Setup

- The arithmetic is in a finite cyclic group $\mathbb{G} = \langle g \rangle$
- of order a ℓ-bit prime number q
- Hash functions

$$\mathcal{H}_0 : \{0,1\}^\star \rightarrow \{0,1\}^{\ell_0} \qquad \mathcal{H}_1 : \{0,1\}^\star \rightarrow \{0,1\}^{\ell_1}$$

- A block cipher $(\mathcal{E}_k, \mathcal{D}_k)$ where $k \in$ Password, onto $\mathbb{G}$.
- $\bar{\mathbb{G}} = \mathbb{G} \backslash \{1\}$, thus $\bar{\mathbb{G}} = \{g^x \mid x \in \mathbb{Z}_q^\star\}$.

Client and server initially share a low-quality password *pw*, uniformly drawn from the dictionary Password.

The session-key space **SK** is $\{0,1\}^{\ell_0}$ equipped with a uniform distribution.

Setup

- The arithmetic is in a finite cyclic group $\mathbb{G} = \langle g \rangle$
- of order a ℓ-bit prime number q
- Hash functions

$$\mathcal{H}_0 : \{0,1\}^\star \rightarrow \{0,1\}^{\ell_0} \qquad \mathcal{H}_1 : \{0,1\}^\star \rightarrow \{0,1\}^{\ell_1}$$

- A block cipher $(\mathcal{E}_k, \mathcal{D}_k)$ where $k \in$ Password, onto $\mathbb{G}$.
- $\bar{\mathbb{G}} = \mathbb{G} \backslash \{1\}$, thus $\bar{\mathbb{G}} = \{g^x \mid x \in \mathbb{Z}_q^\star\}$.

Client and server initially share a low-quality password *pw*,
uniformly drawn from the dictionary Password.

The session-key space **SK** is $\{0,1\}^{\ell_0}$
equipped with a uniform distribution.

Setup

- The arithmetic is in a finite cyclic group $\mathbb{G} = \langle g \rangle$
- of order a ℓ-bit prime number q
- Hash functions

$$\mathcal{H}_0 : \{0,1\}^\star \to \{0,1\}^{\ell_0} \qquad \mathcal{H}_1 : \{0,1\}^\star \to \{0,1\}^{\ell_1}$$

- A block cipher $(\mathcal{E}_k, \mathcal{D}_k)$ where $k \in$ Password, onto $\mathbb{G}$.
- $\bar{\mathbb{G}} = \mathbb{G} \backslash \{1\}$, thus $\bar{\mathbb{G}} = \{g^x \mid x \in \mathbb{Z}_q^\star\}$.

Client and server initially share a low-quality password *pw*, uniformly drawn from the dictionary Password.

The session-key space **SK** is $\{0,1\}^{\ell_0}$ equipped with a uniform distribution.

Setup

- The arithmetic is in a finite cyclic group $\mathbb{G} = \langle g \rangle$
- of order a ℓ-bit prime number q
- Hash functions

$$\mathcal{H}_0 : \{0,1\}^\star \rightarrow \{0,1\}^{\ell_0} \qquad \mathcal{H}_1 : \{0,1\}^\star \rightarrow \{0,1\}^{\ell_1}$$

- A block cipher $(\mathcal{E}_k, \mathcal{D}_k)$ where $k \in$ Password, onto $\mathbb{G}$.
- $\bar{\mathbb{G}} = \mathbb{G} \backslash \{1\}$, thus $\bar{\mathbb{G}} = \{g^x \mid x \in \mathbb{Z}_q^\star\}$.

Client and server initially share a low-quality password *pw*, uniformly drawn from the dictionary Password.

The session-key space **SK** is $\{0,1\}^{\ell_0}$ equipped with a uniform distribution.

Setup

- The arithmetic is in a finite cyclic group $\mathbb{G} = \langle g \rangle$
- of order a ℓ-bit prime number q
- Hash functions

$$\mathcal{H}_0 : \{0,1\}^\star \rightarrow \{0,1\}^{\ell_0} \qquad \mathcal{H}_1 : \{0,1\}^\star \rightarrow \{0,1\}^{\ell_1}$$

- A block cipher $(\mathcal{E}_k, \mathcal{D}_k)$ where $k \in$ Password, onto $\mathbb{G}$.
- $\bar{\mathbb{G}} = \mathbb{G} \backslash \{1\}$, thus $\bar{\mathbb{G}} = \{g^x \mid x \in \mathbb{Z}_q^\star\}$.

Client and server initially share a low-quality password *pw*, uniformly drawn from the dictionary Password.

The session-key space **SK** is $\{0,1\}^{\ell_0}$ equipped with a uniform distribution.

(One) Encrypted Key Exchange

$$\begin{array}{rcl}
\underline{\textit{Client } U}\ (pw) & & \underline{\textit{Server } S}\ (pw) \\
\\
\text{accept} \leftarrow \text{false} & & \text{accept} \leftarrow \text{false} \\
\text{terminate} \leftarrow \text{false} & & \text{terminate} \leftarrow \text{false} \\
x \stackrel{R}{\leftarrow} [1, q-1] & & y \stackrel{R}{\leftarrow} [1, q-1] \\
X \leftarrow g^x & \xrightarrow{\quad U, X \quad} & Y \leftarrow g^y \\
Y \leftarrow \mathcal{D}_{pw}(Y^\star) & \xleftarrow{\quad S, Y^\star \quad} & Y^\star \leftarrow \mathcal{E}_{pw}(Y) \\
\\
K_U \leftarrow Y^x & & K_S \leftarrow X^y \\
Auth \leftarrow \mathcal{H}_1(U\|S\|X\|Y\|K_U) & & \\
sk_U \leftarrow \mathcal{H}_0(U\|S\|X\|Y\|K_U) & & \\
\text{accept} \leftarrow \text{true} & \xrightarrow{\quad Auth \quad} & Auth \stackrel{?}{=} \mathcal{H}_1(U\|S\|X\|Y\|K_S) \\
 & & \qquad \text{if true, accept} \leftarrow \text{true} \\
 & & sk_S \leftarrow \mathcal{H}_0(U\|S\|X\|Y\|K_S) \\
\text{terminate} \leftarrow \text{true} & & \text{terminate} \leftarrow \text{true}
\end{array}$$

Outline

Theorem

Let $\mathcal{A}$ be an adversary against the RoR security within a time bound t, with less than q_s interactions with the parties and q_p passive eavesdroppings, and, asking q_h hash-queries and q_e encryption/decryption queries. Then we have

$$\begin{aligned}\mathsf{Adv}^{ror}(\mathcal{A}) &\leq 3 \times \frac{q_s}{N} + 8q_h \times \mathsf{Succ}_{\mathbb{G}}^{\mathsf{cdh}}(t') \\ &\quad + \frac{(2q_e + 3q_s + 3q_p)^2}{q-1} + \frac{q_h^2 + 4q_s}{2^{\ell_1}}.\end{aligned}$$

where $t' \leq t + (q_s + q_p + q_e + 1) \cdot \tau_e$,
with τ_e the computational time for an exponentiation in $\mathbb{G}$.

Outline

(One) Encrypted Key Exchange

$$
\begin{array}{lcl}
\text{Client U} & & \text{Server S} \\
x \xleftarrow{R} \mathbb{Z}_q^\star & & y \xleftarrow{R} \mathbb{Z}_q^\star \\
(U1)\ X \leftarrow g^x & \xrightarrow{U,X} & (S2)\ Y \leftarrow g^y \\
 & & \quad Y^* \leftarrow \mathcal{E}_{ssid\|pw}(Y) \\
(U3)\ Y = \mathcal{D}_{ssid\|pw}(Y^*) & \xleftarrow{S,Y^*} & \quad K_S \leftarrow X^y \\
\quad K_U \leftarrow Y^x & & \\
\quad Auth \leftarrow \mathcal{H}_1(ssid\|U\|S\|X\|Y\|K_U) & & \\
\quad sk_U \leftarrow \mathcal{H}_0(ssid\|U\|S\|X\|Y\|K_U) & & \\
\quad completed & \xrightarrow{Auth} & (S4)\ \text{if } (Auth = \mathcal{H}_1(ssid\|U\|S\|X\|Y\|K_S)) \\
 & & \quad \text{then } completed \\
 & & \qquad sk_S \leftarrow \mathcal{H}_0(ssid\|U\|S\|X\|Y\|K_S) \\
 & & \quad \text{else } error
\end{array}
$$

Security Result

[Abdalla–Catalano–Chevalier–Pointcheval – CT-RSA 2008]

Theorem

The above protocol securely realizes $\mathcal{F}$ in the random oracle and ideal cipher models (in the presence of adaptive adversaries).

In order to show that the protocol UC-realizes the functionality $\mathcal{F}$, we need to show that for all environments and all adversaries, we can construct a simulator such that the interactions,

- between the environment, the players (say, Alice and Bob) and the adversary (the real world);
- and between the environment, the ideal functionality and the simulator (the ideal world)

are indistinguishable for the environment.

Security Proof

- $\mathbf{G}_0$: real game
- $\mathbf{G}_1$: $\mathcal{S}$ simulates the ideal cipher and the random oracle
- $\mathbf{G}_2$: we get rid off such a situation in which the adversary wins by chance
- $\mathbf{G}_3$: passive case, in which no corruption occurs before the end of the protocol
- $\mathbf{G}_4$: complete simulation of the client, whatever corruption may occur
- $\mathbf{G}_5$: simulation of the server, in the last step of the protocol
- $\mathbf{G}_6$: complete simulation of the server

These games are sequential and built on each other

Conclusion

Outline

Conclusion

Simulation-based Methodology:

Conclusion

Simulation-based Methodology:

- Universal Composability introduced by [Canetti – FOCS 2001]
- allows to define the security properties of one functionality
- a unique proof is enough
- the protocol can then be composed