**Provable Security in the Computational Model**

**IV – Protocols**

David Pointcheval

Ecole normale supérieure, CNRS & INRIA

MPRI – Paris

---

**Outline**

1. **Game-based Security**
   - Key Exchange
   - Authenticated Key Exchange
   - Explicit Authentication

2. **Simulation-based Security**
   - Simulation-based Security
   - Universal Composability
   - Password-based Key Exchange

3. **Encrypted Key Exchange**
   - Description
   - Semantic Security
   - Simulation-based Security

4. **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? → Key-exchange protocols
**Diffie-Hellman Key-Exchange**

\[ G = \langle g \rangle \text{ a group, of prime order } q, \text{ in which the CDH problem is hard} \]

\[ \begin{align*}
\text{Alice} & \quad \text{Bob} \\
\quad x & \quad \quad y \\
\quad X = g^x & \quad Y = g^y \\
\quad \quad \leftarrow & \quad \quad \rightarrow
\end{align*} \]

\[ Y^x = g^{xy} = X^y \]

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

**Security Game: Oracle Accesses**

The adversary has access to the oracles:

- **Execute** \((C^i, S^j)\)
  - \(A\) gets the transcript of an execution between \(C\) and \(S\)
  - It models passive attacks (*eavesdropping*)

- **Send** \((U^i, m)\)
  - \(A\) sends the message \(m\) to the instance \(U^i\)
  - It models active attacks against \(U^i\)

- **Reveal** \((U^i)\)
  - \(A\) 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 \(b = 0\), \(A\) gets the session key \((\text{Reveal}(U^i))\)
  - 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: Find-then-Guess

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

\(A\) 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

\[ \text{Adv}^\text{ftg}(A) = 2 \times \text{Succ}(A) - 1 \]

\[ \text{Adv}^\text{ftg}(t, q_{\text{execute}}, q_{\text{send}}, q_{\text{reveal}}) = \max\{\text{Adv}^\text{ftg}(A)\} \]

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

Definition

A Key Exchange Scheme is **FtG-Semantically Secure** if

\[ \text{Adv}^\text{ftg}(t) \leq \text{negl}() \]

Security Game: Real-or-Random

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

\(A\) 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
- \(\text{Execute}(C, S)\)
  - \(A\) gets the transcript of an execution between \(C\) and \(S\)
  - It models passive attacks (*eavesdropping*)
- \(\text{Send}(U, m)\)
  - \(A\) sends the message \(m\) to the instance \(U\)
  - It models active attacks against \(U\)
- \(\text{Test}(U)\)
  - If \(U\) is not fresh: same answer as for its partner
  - Otherwise
    - If \(b = 0\), \(A\) gets the session key
    - If \(b = 1\), it gets a random key
Semantic Security: Real-or-Random

The semantic security is characterized by

\[ \text{Adv}^{\text{ror}}(A) = 2 \times \text{Succ}(A) - 1 \]

\[ \text{Adv}^{\text{ror}}(t, q_{\text{execute}}, q_{\text{send}}, q_{\text{test}}) = \max \{ \text{Adv}^{\text{ror}}(A) \} \]

Definition

A Key Exchange Scheme is **RoR-Semantically Secure** if

\[ \text{Adv}^{\text{ror}}(t, q_{\text{execute}}, q_{\text{send}}, q_{\text{test}}) \leq \text{negl}(\cdot) \]

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

Theorem

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

Let \( A \) be a FtG-adversary

We build an adversary \( B \) against the RoR security game:

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

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

If \( b \) is the Real choice, then the view of \( A \) is

- Execute(\( C^i, S^i \)) 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 \( \beta \) 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

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

If \( b \) is the Random choice, then the view of \( A \) is

- Execute(\( C^i, S^i \)) 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 \( \beta \)

\[ 2 \times \text{Pr}[\beta^0 = \beta \mid b = 1] - 1 = 0 \]

\[ \text{Adv}^{\text{ror}}(B) = 2 \times \text{Pr}[\beta^0 = \beta] - 1 = \text{Adv}^{\text{ftg}}(A) / 2 \leq \text{Adv}^{\text{ror}}(t, q_{\text{execute}}, q_{\text{send}}, q_{\text{reveal}} + 1) \]

\[ \text{Adv}^{\text{ftg}}(t, q_{\text{execute}}, q_{\text{send}}, q_{\text{reveal}}) \leq 2 \times \text{Adv}^{\text{ror}}(t, q_{\text{execute}}, q_{\text{send}}, q_{\text{reveal}} + 1) \]
Real-or-Random vs. Find-then-Guess

Theorem

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

Let $A$ be a RoR-adversary.
We build an adversary $B$ against the FtG security game:
- A random bit $b$ is chosen by the FtG challenger.
- $B$ chooses a random index $J$.
- Execute($C^i, S^j$) and Send($U^i, m$) queries are forwarded by $B$.
- 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 $A$’s answer $\beta^0$, $B$ outputs $(\beta = \beta^0)$.

Outline

1. Game-based Security
   - Key Exchange
   - Authenticated Key Exchange
   - Explicit Authentication
2. Simulation-based Security
3. Encrypted Key Exchange
4. Conclusion

Man-in-the-Middle Attacks

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

Client $C$

$\begin{align*}
  x &\leftarrow \mathbb{Z}_q \\
  X &\leftarrow g^x \\
  sk_C &\leftarrow Y^x
\end{align*}$

Server $S$

$\begin{align*}
  y &\leftarrow \mathbb{Z}_q \\
  Y &\leftarrow g^y \\
  sk_S &\leftarrow (Xg)^y
\end{align*}$

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

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

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:

\[ G = \langle g \rangle \text{ a group, of prime order } q, \text{ in which the DDH problem is hard} \]

\[
\begin{align*}
Alice & \quad x \overset{R}{\leftarrow} \mathbb{Z}_q \\
X &= g^x \\
& \quad \text{Sign}_A(B, X) \\
& \quad \text{Sign}_B(A, X, Y) \\
Y &= g^y \\
Y^x &= g^{xy} = X^y
\end{align*}
\]

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

\[
\text{Adv}^\text{ror}(t, q_{\text{user}}, q_{\text{execute}}, q_{\text{send}}, q_{\text{test}}) \\
\leq q_{\text{user}} \times \text{Succ}^\text{euf−cma}\left( t + (3q_{\text{execute}} + q_{\text{send}} + q_{\text{test}})^\tau_{\text{exp}}, \right) \\
+ \text{Adv}^\text{ddh}(t + (7q_{\text{execute}} + 2q_{\text{send}} + 4q_{\text{test}})^\tau_{\text{exp}})
\]

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

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 \( A \)
- \( \text{Execute}(A, B^i) \) or \( \text{Execute}(B^i, A) \): we use the signing oracle
- \( \text{Send}(A, m) \): we use the signing oracle
- \( \text{Send}(B, \text{Sign}_A(m)) \): if not signed by the signing oracle, we reject
- \( \text{Test}(U) \): as usual

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

\[
q_{\text{user}} \times \text{Succ}^\text{euf−cma}\left( t + (3q_{\text{execute}} + q_{\text{send}} + q_{\text{test}})^\tau_{\text{exp}}, \right) \\
+ \text{Adv}^\text{prompt}(t + (7q_{\text{execute}} + 2q_{\text{send}} + 4q_{\text{test}})^\tau_{\text{exp}})
\]
Signed Diffie-Hellman: DDH

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

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

We thus have

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

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 \((X = g^x, Y = g^y, Z = g^z)\)

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

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

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

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\((A', B')\): we use a fresh \(X_i\) but \(Y^0 = g^y\) for a known \(y^0\)
  We can compute \(Z^0\)
- Send\((A, \text{Start})\): we use a fresh \(X_i\)
- Send\((B, \text{Sign}_A(B, X))\): if valid, we look for \(X_i = X\), use a fresh \(Y_{i,j}\)
  The associated key is \(Z_{i,j}\)
- Send\((A, \text{Sign}_B(A, X, Y))\): 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

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.

\[ Alice \]
\[ x \overset{R}{\leftarrow} \mathbb{Z}_q \]
\[ X = g^x \]
\[ MAC_k(A, B, X) \]

\[ Bob \]
\[ y \overset{R}{\leftarrow} \mathbb{Z}_q \]
\[ Y = g^y \]
\[ MAC_k(B, A, X, Y) \]
\[ Y^x = g^{xy} = X^y \]

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

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

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

\[
\text{Adv}^{ftg}(t) \leq q_{send}/|D| + \text{negl}(t)
\]

or even better

\[
\text{Adv}^{ror}(t) \leq q_{send}/|D| + \text{negl}(t)
\]

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

Outline

1. **Game-based Security**
   - Key Exchange
   - Authenticated Key Exchange
   - Explicit Authentication

2. Simulation-based Security

3. Encrypted Key Exchange

4. Conclusion
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 partner has accepted

Corruption

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

Outline

1. Game-based Security
2. Simulation-based Security
   - Universal Composability
   - Password-based Key Exchange
3. Encrypted Key Exchange
4. 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 player,
- 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 for every environment $\mathcal{Z}$

the views are indistinguishable:

$$\forall \mathcal{A}, \exists \mathcal{S}, \forall \mathcal{Z}, EXEC_{\mathcal{F},\mathcal{S},\mathcal{Z}} \approx EXEC_{\mathcal{P},\mathcal{A},\mathcal{Z}}$$
Equivalent Formulations

\[ \forall A, \exists S, \forall Z, \text{EXEC}_F, S, Z \approx \text{EXEC}_P, A, Z \]

\[ \forall A, \forall Z, \exists S, \text{EXEC}_F, S, Z \approx \text{EXEC}_P, A, Z \]

\[ \exists S, \forall Z, \text{EXEC}_F, S, Z \approx \text{EXEC}_P, A_d, Z \]

where \( A_d \) is the dummy adversary: under the control of the environment (forwards every input/output).

Security

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

Then, nothing can be done against \( P \)

Outline

1. Game-based Security
2. Simulation-based Security
   - Simulation-based Security
   - Universal Composability
   - Password-based Key Exchange
3. Encrypted Key Exchange
4. Conclusion

Implications of UC

Can design and analyze protocols in a modular way:

- Divide a given task \( F \) into sub-tasks \( F_1, \ldots, F_n \)
  \( F \) is equivalent to \( F_1 \cup F_2 \cup F_3 \cup F_4 \)
- Construct protocols \( \pi_1, \ldots, \pi_n \) emulating \( F_1, \ldots, F_n \)
- Combine them into a protocol \( \pi \)
- Composition theorem: \( \pi \) emulates \( F \)

Can be done concurrently and in parallel
Theorem (Universal Composition)

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

**Queries**
- **NewSession** = a player initializes the protocol
  The passwords are chosen by the environment.
- **TestPwd** = $A$ attempts to guess a password (one per session)
  In case of correct guess, the adversary is allowed to choose the session key.
  ⇒ models the on-line dictionary attacks
- **NewKey** = $A$ 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

**Outline**
1. Game-based Security
2. Simulation-based Security
3. Encrypted Key Exchange
   - Description
     - Semantic Security
     - Simulation-based Security
4. Conclusion
Setup

- The arithmetic is in a finite cyclic group $G = \langle g \rangle$ of order a $\ell$-bit prime number $q$.
- Hash functions
  - $H_0 : \{0, 1\}^* \rightarrow \{0, 1\}^{\ell_0}$
  - $H_1 : \{0, 1\}^* \rightarrow \{0, 1\}^{\ell_1}$
- A block cipher $(E_k, D_k)$ where $k \in \text{Password}$, onto $G$.
- $\overline{G} = G \setminus \{1\}$, thus $\overline{G} = \{g^x \mid x \in \mathbb{Z}_q^*\}$.

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

Client $U$ (pw)
- $x \leftarrow [1, q - 1]$ \[ U, X \]
- $Y \leftarrow D_{pw}(Y^*) \quad \rightarrow \quad Y^* \leftarrow E_{pw}(Y)$
- $K_U \leftarrow Y^x
- Auth \leftarrow H_1(UkSkXkYkK_U)$
- $sk_U \leftarrow H_0(UkSkXkYkK_U)$
- accept $\leftarrow$ true
- terminate $\leftarrow$ true

Server $S$ (pw)
- $y \leftarrow [1, q - 1]$ \[ Y \]
- $X \leftarrow g^x \quad \rightarrow \quad S, Y^*$
- $Y^* \leftarrow E_{pw}(Y)$
- $K_S \leftarrow X^y
- Auth \leftarrow H_1(UkSkXkYkK_S)$
- if true, accept $\leftarrow$ true
- $sk_S \leftarrow H_0(UkSkXkYkK_S)$
- terminate $\leftarrow$ true

Outline

1. Game-based Security
2. Simulation-based Security
3. Encrypted Key Exchange
   - Description
   - Semantic Security
   - Simulation-based Security
4. Conclusion

Security Result

[Bresson–Chevassut–Pointcheval – ACM CCS 2003]

Theorem

Let $A$ 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

$$\text{Adv}^{\text{cor}}(A) \leq 3 \times \frac{q_s}{N} + 8q_h \times \text{Succ}_{cdh}^{\text{cor}}(t_0) + \frac{(2q_e + 3q_s + 3q_p)^2}{q - 1} \times \frac{q_h^2}{2^{\ell_1}} + 4q_s,$$

where $t_0 \leq t + (q_s + q_p + q_e + 1) \cdot \tau_e$, with $\tau_e$ the computational time for an exponentiation in $G$. 

---

[One] Encrypted Key Exchange

Client $U$ (pw)

- $x \leftarrow [1, q - 1]$ \[ U, X \]
- $Y \leftarrow D_{pw}(Y^*) \quad \rightarrow \quad Y^* \leftarrow E_{pw}(Y)$
- $K_U \leftarrow Y^x
- Auth \leftarrow H_1(UkSkXkYkK_U)$
- $sk_U \leftarrow H_0(UkSkXkYkK_U)$
- accept $\leftarrow$ true
- terminate $\leftarrow$ true

Server $S$ (pw)

- $y \leftarrow [1, q - 1]$ \[ Y \]
- $X \leftarrow g^x \quad \rightarrow \quad S, Y^*$
- $Y^* \leftarrow E_{pw}(Y)$
- $K_S \leftarrow X^y
- Auth \leftarrow H_1(UkSkXkYkK_S)$
- if true, accept $\leftarrow$ true
- $sk_S \leftarrow H_0(UkSkXkYkK_S)$
- terminate $\leftarrow$ true

Outline

1. Game-based Security
2. Simulation-based Security
3. Encrypted Key Exchange
   - Description
   - Semantic Security
   - Simulation-based Security
4. Conclusion
```plaintext
Client U

x ← Z_q
(U1) X ← g^x

S

y ← Z_q^*
(S2) Y ← g^y

Y^* ← E_{ssid∥pw}(Y)

K_U ← Y^x

Auth ← H_1(ssidUkSkXkYkK_U)

sk_U ← H_0(ssidUkSkXkYkK_U)

completed

K_S ← X^y

if Auth = H_1(ssidUkSkXkYkK_S) then completed

else error

ENS/CNRS/INRIA Cascade
David Pointcheval
57/62
ENS/CNRS/INRIA Cascade
David Pointcheval
58/62

Theorem

The above protocol securely realizes \( F \) 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 \( F \), 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.

G_0: real game
G_1: \( S \) simulates the ideal cipher and the random oracle
G_2: we get rid off such a situation in which the adversary wins by chance
G_3: passive case, in which no corruption occurs before the end of the protocol
G_4: complete simulation of the client, whatever corruption may occur
G_5: simulation of the server, in the last step of the protocol
G_6: complete simulation of the server

These games are sequential and built on each other.
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