Introduction

Key Exchange Protocols

A fundamental problem in cryptography:
Enable secure communication over insecure channels

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

Diffie-Hellman Key Exchange

The classical Diffie-Hellman protocol allows such a key exchange:
in a finite cyclic group $G$, of prime order $p$, with a generator $g$

$$x \leftarrow \mathbb{Z}_p, X \leftarrow g^x \quad \Rightarrow \quad X \quad \Rightarrow \quad y \leftarrow \mathbb{Z}_p, Y \leftarrow g^y$$

$$K \leftarrow Y^x = g^{xy} \quad \Leftarrow \quad Y \quad \Leftarrow \quad K \leftarrow X^y = g^{xy}$$

No authentication provided

Authenticated Key Exchange

Semantic security / Implicit Authentication:
the session key should be indistinguishable from a random string
to all except the expected players

How to obtain such a shared secret key? —> Key exchange protocols
Authentication Techniques

Asymmetric technique
- Assume the existence of a public-key infrastructure
- Each party holds a pair of secret and public keys
- 2-party and group settings

Symmetric technique
- Users share a random secret key
- 2-party or server-based settings

Password-based technique
- Users share a random low-entropy secret: password
- 2-party and group settings

Electronic Passport

Since 1998, some passports contain digital information on a chip. Standards are specified by ICAO (International Civil Aviation Organization)

In 2004, security introduced:
- encrypted communication between the chip and the reader
- access control: BAC (Basic Access Control)

The shared secret is on the MRZ (Machine Readable Zone)
It has low entropy:
- at most 72 bits,
- but actually approx. 40

⇒ low-entropy shared secret: a password \( pw \)

BAC: Basic Access Control

The symmetric encryption and MAC keys are derived from \( pw \)

\[
\begin{align*}
\text{Passport} & \quad r_P, k_P \xleftarrow{} \{0,1\}^{64} \\
\text{Reader} & \quad r_R, k_R \xleftarrow{} \{0,1\}^{64} \\
C_R & \xleftarrow{} \text{Enc}_{pw}(r_R, r_P, k_P) \\
M_R & \xleftarrow{} \text{Mac}_{pw}(C_R)
\end{align*}
\]

\[
\begin{align*}
C_P & \leftarrow \text{Enc}_{pw}(r_P, r_R, k_P) \\
M_P & \leftarrow \text{Mac}_{pw}(C_P) \\
K & \leftarrow k_P \oplus k_R
\end{align*}
\]

From a pair \((C_R, M_R)\), one can make an exhaustive search on the password \( pw \) to check the validity of the Mac \( M_R \)

After a few eavesdroppings only: password recovery

What can we expect from a low-entropy secret?
**On-line Dictionary Attacks**

- The adversary interacts with a player, trying a password
- In case of success: it has guessed the password
- In case of failure: it tries again with another password
- If the dictionary has a size $N$, the adversary wins after $N/2$ attempts

**In Practice**
- This attack is unavoidable
- If the failures for a target user can be detected: the impact can be limited by various techniques (limited number of failures, delays between attempts, ...)
- If the failures cannot be detected (anonymity, no check, ...) the impact can be dramatic

---

**The Most Famous Examples**

- **EKE: Encrypted Key Exchange** [Bellare–P.–Rogaway, 2000]
- **SPEKE: Simple Password Exponential Key Exchange** [Jablon, 1996]

**Security Models**

- **Game-based Security**
- **Find-then-Guess**
- **Real-or-Random**
- **Simulation-based Security**
- **Universal Composability**

**What does security really mean?**

\[ q_S = \frac{\# \text{Active Sessions}}{N} \]

\[ q_S = \frac{\# \text{Active Sessions}}{N} \]

\[ q_S = \frac{\# \text{Active Sessions}}{N} \]

\[ q_S = \frac{\# \text{Active Sessions}}{N} \]

\[ q_S = \frac{\# \text{Active Sessions}}{N} \]

\[ q_S = \frac{\# \text{Active Sessions}}{N} \]
Game-based Security

Computational Security Proofs

- a formal security model (security notions)
- a reduction: if one (Adversary) can break the security notions, then one (Simulator + Adversary) can break a hard problem
- acceptable computational assumptions (hard problems)

The adversary $A$ interacts with oracles:

- **Execute**($A^i, B^j$)
  - $A$ gets the transcript of an execution between $A$ and $B$
  - It models passive attacks (eavesdropping)
- **Send**($U^i, m$)
  - $A$ sends the message $m$ to the instance $U^i$
  - It models active attacks against $U^i$ (active sessions)
- **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 (i.e. Reveal($U^i$))
  - If $b = 1$, $A$ gets a random key

Security Game: Find-then-Guess

- Secrecy of the key: guess $b'$ of the bit $b$ involved in the Test-query
- Is the obtained key real or random?
- Constraint: no Test-query on a trivially known key
  - *i.e.* key already revealed thought the instance or its partner

- **Semantic Security**
  - The Find-then-Guess game models the **secrecy** of the key
  - **⇒** the session key is unknown to the other players
    - What about this secrecy after the corruption of a player?
    - What about the knowledge of the two players?

- **Forward Secrecy**
  - An additional oracle: Corrupt($U$) provides the password pw of the player $U$ to the adversary
  - A new constraint: For any Test($U^i$), player $U$ was not corrupted when $U^i$ was involved in its session

- **Explicit Authentication**
  - **⇒** the session key is **really** known to the two expected players
  - The attacker wins the Explicit Authentication Game if
    - an instance terminates with a key
    - without exactly one partner having the material to compute the key

- **AdvFG($A$) = $2 \times Pr[b' = b] - 1 \leq \frac{O(qS)}{N} + \text{negl()}**
### Secure Protocols: EKE-like

With both Random Oracles and an Ideal Cipher
- **EKE (ROM+ICM)**
  - with Forward-Secrecy
  - [Bellare–P.-Rogaway, 2000]
- **OEKE (ROM+ICM)**
  - with Forward-Secrecy and Client-Authentication
  - Formally verified with CryptoVerif
  - [Blanchet, 2012]

With Random Oracles (and One-time Pad)
- **OMDHKE (ROM)**
  - with Forward-Secrecy and Server-Authentication
  - [Bresson–Chevassut–P., 2004]
- **PAKE (ROM)**
  - [Abdalla–P., 2005]

#### Quite Simple Scheme

\[
\begin{align*}
x & \xleftarrow{\$} \mathcal{P} \\
x^p & \leftarrow x^a \\
x^y & \leftarrow x^{hk} \\
 Y & \leftarrow \mathcal{D}
\end{align*}
\]

\[
\begin{align*}
h & \leftarrow \mathcal{H}((A \ B \ X \ Y \ p \ w \ k)) \\
\end{align*}
\]

### Smooth Projective Hash Functions

**Definition**

Let \( \{H\} \) be a family of functions from \( X \) to \( G \) and \( L \) a subset (language) of this domain \( X \) such that, for any point \( x \in L \), and a witness \( w \),
- \( H(x) = \text{Hash}_L(hk; x) \), with the secret hashing key \( hk \)
- \( H(x) = \text{ProjHash}_L(hp; x, w) \), with the public projected key \( hp \)

- Hard-Partitioned Subset: \( L \) and \( X \) hard to distinguish
- Smoothness: if \( x \not\in L \), \( H(x) \) and \( hp \) are independent
- Pseudo-Randomness: if \( x \in L \), \( H(x) \) is pseudo-random, with \( hp \) but without a witness \( w \)

### Security Game: Real-or-Random

Secrecy/independence of all the keys:
- many Test-queries on any \( U \) with the same bit \( b \)
  - If no key defined by the protocol yet: output \( \bot \)
  - If dishonest/corrupted partner: output the real key
  - If player/partner already tested: output the same key
  - If \( b = 0 \): output the real key
  - If \( b = 1 \): output a random key

\[
\text{Adv}^{RoR}(A) = 2 \times \Pr[b' = b] - 1
\]
Security Game: Real-or-Random


Find-then-Guess and Real-or-Random are polynomially equivalent

\[ \text{Adv}_{\text{RoR}}(t, q_T) \leq q_T \times \text{Adv}_{\text{FG}}(t) \]

where \( q_T \) is the number of Test-queries

For Password-based Authenticated Key Exchange:

\[ \text{Adv}_{\text{FG}}(t) \leq O\left(\frac{q_S}{N}\right) \]

\( \Rightarrow \) **Much stronger notion**

No need of Reveal-queries [Abdalla–Fouque–P., 2005]

\( \Rightarrow \) **Much simpler security notion**

Game-based Security: Limitations

- Proven bound: \( O(q_S)/N \), but almost never \( q_S/N \)
  \( \Rightarrow \) hard to get optimal bound!
  Maybe several passwords can be excluded by each active attack
- Passwords chosen from pre-determined, known distributions
- Different passwords are assumed to be independent
- No security guarantees under arbitrary composition

\( \Rightarrow \) **Universal Composability more appropriate?** [Canetti, 2001]


---

**Introduction**

**Definition**

**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**

\( \mathcal{P} \) emulates \( \mathcal{F} \) if, for any environment \( Z \), for any adversary \( \mathcal{A} \), there exists a simulator \( S \) so that, the view of \( Z \) is the same for

- \( \mathcal{A} \) attacking the real protocol \( \mathcal{P} \)
- \( S \) attacking the ideal functionality \( \mathcal{F} \)
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 $P$.

Password-based Authenticated Key Exchange

**PAKE Ideal Functionality**

**Queries**

- **NewSession** = a player joins the system with a password
- **TestPwd** = $A$ attempts to guess a password (one per session).
  The adversary learns whether the guess was correct or not.
- **NewKey** = $A$ asks for the session key to be computed and delivered to the player.

**Corruption-Query**

- $A$ gets the long-term secrets (pw) and the internal state.
- $A$ takes the entire control on the player and plays on its behalf.

Corruptions can occur before the execution: Static Corruptions.
Corruptions can occur at any moment: Adaptive Corruptions.

**Session Key**


- no corrupted players, same passwords
  $\Rightarrow$ same key, randomly chosen
- no corrupted players, different passwords
  $\Rightarrow$ independent keys, randomly chosen
- a corrupted player (with the secret from the environment)
  $\Rightarrow$ key chosen by the adversary
- correct password guess (TestPwd-query)
  $\Rightarrow$ key chosen by the adversary
- incorrect password guess (TestPwd-query)
  $\Rightarrow$ independent keys, randomly chosen

**Properties**

- The TestPwd-query models the on-line dictionary attacks
- The Corruption-query includes forward-secrecy

**Advantages wrt Game-based Security**

- No assumption on the distribution of passwords
- Passwords can be related (it models mistyping)
- Security under arbitrary compositions $\Rightarrow$ secure channels
**Game-based Security vs. Universal Composability**

**Game-based Security**

In the reduction, the simulator has to emulate the protocol execution only up to an evidence the adversary has won \( (pw \Rightarrow \text{not negl.}) \).

In a global system, the simulation may thus fail as soon as an adversary breaks one of the components whereas other parts could provide protection \( (pw \Rightarrow \text{weak proof!}) \).

**UC Security**

Handles compositions, but proofs are more complex: the simulator must have an indistinguishable behavior, even when the adversary wins!

In the case of password-based cryptography: the adversary can win with non-negligible probability!

**Secure Protocols**

In the standard model, with CRS:

- **GL\(^+\)** *(with ZK proofs)*
  \[ \Rightarrow \text{Static Corruptions} \]

- With an equivocable/extractable commitment *(bit-by-bit)*
  \[ \Rightarrow \text{GL secure against Adaptive Corruptions} \] [Abdalla–Chevalier–P., 2009]

- With hp independent of the commitment *(with NIZK)*
  \[ \Rightarrow \text{one-round only} \]

With random oracles and an ideal cipher:

- **OKE**
  \[ [\text{Abdalla–Catalano–Chevalier–P., 2008}] \]

  \[ \Rightarrow \text{First efficient scheme secure against Adaptive Corruptions} \]

**Weak Authentication: Split Functionality**

[Barak–Canetti–Lindell–Pass–Rabin, 2005]

No initial authentication: anybody can join the protocol.

In a multi-party protocol, the adversary can emulate all the other players against one victim, and can do it \( n \) times, against the \( n \) real players.

**Session Key: NewKey-Query**

... a corrupted player \( \Rightarrow \text{key chosen by the adversary} \)

... correct password guess \( \Rightarrow \text{key chosen by the adversary} \)

The NewKey-query is weak

- A lot of control by the adversary:
  as soon as it controls a player, it controls the key
  Key Distribution vs. Key Agreement: Contributiveness

- Not much information leaked to the adversary:
  whether the protocol succeeds or not
  In practice, the communication continues or stops \[ \Rightarrow \text{some information leaks!} \]

**Limitations of the NewKey-Query**

- GPAKE: Each sub-session allows to test one password
Contributiveness

Initial Definition of the Session Key
- no corrupted players, same passwords ⇒ same random key
- corrupted player or correct TestPwd ⇒ key chosen by \( A \)
- otherwise ⇒ independent random keys

With Contributiveness
- at least one non-corrupted player, same passwords ⇒ same random key
- all players corrupted ⇒ key chosen by \( A \)
- otherwise ⇒ independent random keys

It extends to Group protocols, with threshold: \((t,n)\)-Contributiveness
No player more important than others: \( \neq \) key distribution
Prevents from weak random coins or Trojan horses

Success Information

The players could learn whether the authentication succeeded

Explicit Authentication
At the Key Delivery time, the player learns: Success or Failure

Together with the Split Functionality:
the adversary makes a user try a password
it then learns whether it is correct \( \Rightarrow \) similar to TestPwd

The adversary should learn this information too (available in practice!)

Successful Agreement
At the Key Computation time, the adversary learns: OK or NOK

In both cases, one can remove the TestPwd-query
allowing the adversary to join a session with a NewSession-query!

Simpler (but Stronger) Functionality

Queries
- NewSession = a player joins the protocol with a password
  or \( A \) joins the protocol with a password on behalf of a player
  \( \Rightarrow A \) impersonates \( P_i \): it receives the messages for it
- NewKey = \( A \) asks for the session key to be generated
- SendKey = \( A \) asks for the session key to be delivered

NewKey-Query
- the two players are controlled by the adversary
  \( \Rightarrow \) No need to inform anybody: the adversary plays alone!
- Same passwords ⇒ same random key – \( A \) informed: OK
- otherwise ⇒ \( \perp – A \) informed: NOK

More general \( \Rightarrow \) not limited to passwords: Consistent Inputs?

Generalized Functionality: LAKE

Language-based Authenticated Key Exchange
[Blazy–Chevalier–Pointcheval–Vergnaud, 2012]

Two players want to agree on a common secret key,
IFF their partner actually knows a word in an appropriate language:
- Alice owns a word \( w_a \) in a language \( L_a(Pub_a, Priv_a) \);
- Bob owns a word \( w_b \) in a language \( L_b(Pub_b, Priv_b) \);
- If Alice and Bob implicitly agree on the languages,
  and own valid words (implicit authentication),
  \( \Rightarrow \) they agree on a common session key (semantic security)

E.g. \( Pub = M, Priv = vk \): the language \( L(Pub, Priv) \) contains
the valid signatures of \( M \) under the verification key \( vk \),
where \( M = \) public message, but \( vk = \) implicit verification key
LAKE: Ideal Functionality

Queries
- **NewSession** = a player or \( A \) (for a player) joins the protocol with
  - its own language parameters: \( \text{Pub} \) and \( \text{Priv} \)
  - its partner’s language parameters: \( \text{Pub}' \) and \( \text{Priv}' \)
  - its word \( w \)
- **NewKey** = \( A \) asks for the session key to be generated
- **SendKey** = \( A \) asks for the session key to be delivered

Consistent Inputs
The protocol succeeds with the same key if and only if

\[
(\text{Pub}_a, \text{Priv}_a) = (\text{Pub}'_b, \text{Priv}'_b), \quad (\text{Pub}_b, \text{Priv}_b) = (\text{Pub}'_a, \text{Priv}'_a),
\]

\( w_a \in L_a(\text{Pub}_a, \text{Priv}_a), \quad w_b \in L_b(\text{Pub}_b, \text{Priv}_b) \)

LAKE: General Approach

Verification
- \( \text{Pub}_a = \text{Pub}'_b \) & \( \text{Pub}_b = \text{Pub}'_a \): public matching verification
- \( \text{Priv}_a = \text{Priv}'_b \) & \( \text{Priv}_b = \text{Priv}'_a \): implicit matching verification

\[ \implies \text{as in PAKE} \]
- \( w_a \in L_a(\text{Pub}_a, \text{Priv}_a) \) & \( w_b \in L_b(\text{Pub}_b, \text{Priv}_b) \): implicit verification

\[ \implies \text{much more complex check!} \]

The GL approach, with advanced Smooth Projective Hash Functions, allows to implement all these private/implicit checks.

Can be instantiated under the DLIn assumption or the DDH assumption

LAKE: Applications

The improved **NewKey**-query is more powerful/general than the **TestPwd**-query!

LAKE is a quite general framework that includes all the AKE variants:

Particular Instantiations
- \( \text{Pub} = \{\} \), \( \text{Priv} = \text{pw} \) and \( L(\text{Pub}, \text{Priv}) = \{\text{Priv}\} \)
  \[ \implies \text{PAKE (15 group elements exchanged)} \]
  With \( \text{Priv} = (g^{\text{pw}}, h^{\text{pw}}) \): verifier-based PAKE (29 group elements)
  - \( \text{Pub} = \text{M}, \text{Priv} = \text{vk} \), \( L(\text{Pub}, \text{Priv}) = \{\sigma, \text{Verif}(\text{Priv}, \text{Pub}, \sigma) = 1\} \)
  \[ \implies \text{Secret Handshake} \quad [\text{Balfanz–Durfee–Shankar–Smetters–Staddon–Wong, 2003}] \]
  \[ \quad (43 \text{ group elements for Waters Signatures}) \]

Admits efficient instantiations!

Conclusion

Theoretical Aspects
- Many security models for AKE and PAKE: Mature Topic
- Many PAKE candidates:
  - EKE-like protocols are quite efficient, but ideal models
  - GL approach is quite powerful, and reasonably efficient
- LAKE: more general applications, and efficient instantiations

PAKE in Practice
- While appealing, PAKE not really used in practice:
  - IETF RFC 2945 for SRP (no security analysis!)
  - EKE-like: quite efficient but patented \[ \implies \text{not used so far} \]
  - EKE Patent expired late 2011 \[ \implies \text{recent IETF RFC 6124} \]

With EKE-like (efficient) or GL-based (fine-grained authentication) approaches, any situation should find an AKE solution!