PAKE in the UC-Framework
Adaptive Security

CANS '2007
Singapore
Sunday, December 9th, 2007

Universal Composability
Provable Security

Security proofs give the guarantee that an assumption is enough for security: if an adversary can break the system one can break the assumption \( \Rightarrow \) "reductionist" proof

Proof by Reduction

Reduction of a problem \( P \) to an attack \( Atk \):

Let \( A \) be an adversary that breaks the scheme

Then \( A \) can be used to solve \( P \)

- \( \neg \) \( A \) is a reduction from \( P \) to \( Atk \)
Proof by Reduction

- Let $A$ be an adversary that breaks the scheme $P$.
- Then $A$ can be used to solve the attack $Atk$.

Provably Secure Scheme

- One has to make precise the algorithmic assumptions.
- Define the security notions to be guaranteed.
- Use a reduction: an adversary can help to break the assumption.
In such a reduction, our simulator tries to emulate the environment, until the adversary may win the attack game. What about the composition of multiple protocols? The simulation fails as soon as an adversary may break one part of the global system, whereas other executing protocols may provide additional information to the adversary. \[ \Rightarrow \]

\[\text{either we re-prove the global system,} \]
\[\text{or we prove each component in the UC Framework.}\]

Universal Composability

[Canetti - FOCS '01]
Real vs. Ideal

Definition of security

Protocol $\pi$ emulates the ideal process for $F$ if for any adversary $A$, there exists a simulator $S$ such that for all $Z$, $IDEAL_F S, Z \approx EXEC\pi, A, Z$. Hence, we say that protocol $\pi$ securely realizes $F$.

Equivalently:

$\exists S_d \forall Z \ \ IDEAL_F S, Z \approx EXEC\pi, A_d, Z$

UC Theorem: Composition

Modular composition

$C \approx IDEAL_F S, D \approx IDEAL_F D, F_{S,D} \approx IDEAL_F S,D$
UC Theorem: Idea

UC Theorem: Idea
UC Theorem: Idea

Implications of UC

Now can be done concurrently and in parallel.)
Key Exchange

- "key exchange" (a two-party protocol to generate a common random key that is "secret" for external adversaries.

Assuming authenticated communication (Diffie-Hellman model)

Unauthenticated communication (AKE)

Different ways to authenticate the exchange:

- Long-term public keys for signature or encryption plus "public-key infrastructure".
- Long-term pre-shared keys
- Trusted third parties (The Kerberos model)
- Passwords
Analysis of AKE

PAKE in the UC-Framework - 17

AKE has been studied extensively: Protocols were proposed, and later broken. First complexity-based notion: [Bellare-Rogaway - Crypto '93] Based on a "distinguishing game" for the adversary (FtG) Explicitly handles multiple concurrent sessions Treatments that argue usability for secure sessions: Bellare-Canetti-Krawczyk - STOC '98 simulation based (but has problems) Canetti-Krawczyk – EC '01: based on BR93 with a different system model, defines and obtains "secure sessions". Canetti-Krawczyk – EC '02: A UC treatment of AKE

Ideal Functionality: KE

Functionality $\mathcal{F}_{KE}$

$\mathcal{F}_{KE}$ is parameterized by a security parameter $k$. It interacts with an adversary $S$ and a set of (dummy) parties via the following queries:

Upon receiving a query (NewSession, $sid$, $P_i$, $P_j$, role) from party $P_i$:

Send (NewSession, $sid$, $P_i$, $P_j$, role) to $S$. In addition, if this is the first NewSession query, or if this is the second NewSession query and there is a record ($P_j$, $P_i$), then record ($P_i$, $P_j$).

Upon receiving a query (NewKey, $sid$, $P_i$, $sk$) from $S$, where $|sk| = k$:

If there is a record ($P_i$, $P_j$), and this is the first NewKey query for $P_i$, then:

- If either $P_i$ or $P_j$ is corrupted, then output ($sid$, $sk$) to player $P_i$.
- If there is also a record ($P_j$, $P_i$), and a key $sk'$ was sent to $P_j$, output ($sid$, $sk'$) to $P_i$.
- In any other case, pick a new random key $sk'$ of length $k$ and send ($sid$, $sk'$) to $P_i$.

Figure 1: The authenticated key-exchange functionality $\mathcal{F}_{KE}$
Password-Based Authentication

- **Asymmetric**: \((\sk_A, \pk_A) \leftrightarrow \cdots \leftrightarrow (\sk_B, \pk_B)\)

- **Symmetric**: \(\cdots \leftrightarrow (\sk, \pk)\)

- **Password**: \(\cdots \leftrightarrow (\text{20-bit})\)

Let us assume a 20-bit password \(\Rightarrow \) it is possible to win with non-negligible advantage

---

**Ideal Functionality: pwKE**


The functionality \(\mathcal{F}_{pwKE}\) is parameterized by a security parameter \(k\). It interacts with an adversary \(S\) and a set of parties via the following queries:

**Upon receiving a query** \((\text{NewSession}, \text{id}, P_i, P_j, \text{pw}, \text{role})\) **from party** \(P_i\):

Send \((\text{NewSession}, \text{id}, P_i, P_j, \text{role})\) to \(S\). In addition, if this is the first \text{NewSession} query, or if this is the second \text{NewSession} query and there is a record \((P_j, P_i, \text{pw}')\), then record \((P_i, P_j, \text{pw})\) and mark this record fresh.

**Upon receiving a query** \((\text{TestPwd}, \text{id}, P_i, \text{pw}')\) **from the adversary** \(S\):

If there is a record of the form \((P_i, P_j, \text{pw})\) which is fresh, then do:
- If \(\text{pw} = \text{pw}'\), mark the record compromised and reply to \(S\) with “correct guess”.
- If \(\text{pw} \neq \text{pw}'\), mark the record interrupted and reply with “wrong guess”.

**Upon receiving a query** \((\text{NewKey}, \text{id}, P_i, \text{sk})\) **from** \(S\), **where** \(|\text{sk}| = k\):

If there is a record of the form \((P_i, P_j, \text{pw})\), and this is the first \text{NewKey} query for \(P_i\), then:
- If this record is compromised, or either \(P_i\) or \(P_j\) is corrupted, then output \((\text{id}, \text{sk})\) to player \(P_i\).
- If this record is fresh, and there is a record \((P_j, P_i, \text{pw}')\) with \(\text{pw}' = \text{pw}\), and a key \(\text{sk}'\) was sent to \(P_j\), and \((P_j, P_i, \text{pw})\) was fresh at the time, then output \((\text{id}, \text{sk}')\) to \(P_i\).
- In any other case, pick a new random key \(\text{sk}'\) of length \(k\) and send \((\text{id}, \text{sk}')\) to \(P_j\).

Either way, mark the record \((P_i, P_j, \text{pw})\) as completed.
In this ideal functionality:

- **TestPwd** query, which gives the adversary the authorization to test one password per session.

If the adversary correctly guesses a password, the adversary can choose the corresponding key.

Passwords:
The environment chooses the passwords. Can thus make players run with different passwords, or related passwords.

⇒ Passwords are not in an internal state of the functionality: no need of joint-state UC.

Concurrent Executions

**KOY/GL Protocol**

\[
\begin{align*}
P_i \text{ (client)} & \quad \text{CRS: pke} \quad P_j \text{ (server)} \\
& \\
& c_2 \leftarrow E_{pke}(pw, r_2) \\
hk & \leftarrow H \\
hp & \leftarrow \alpha(hk; c_1) \\
& \\
& \text{if } (\text{Verify}_{\alpha}(hc, hp', \sigma) = 1) \\
& \quad \text{session-key} \leftarrow H_{hk}(c_1, pw) + h_{hp'}(c_2, pw; r_2) \\
& \quad \sigma \leftarrow \text{Sign}_{sk}(c_2, hp, hp') \\
& \\
& (sk, vk) \leftarrow \text{sigKey}(\$) \\
& c_1 \leftarrow E_{pke}(pw; r_1) \\
& \text{session-key} \leftarrow h_{hp}(c_1, pw; r_1) + H_{hk'}(c_2, pw) \\
& \\
& c_1, vk \\
& \\
& c_2, hp \\
& \\
& h_{hp}, \sigma
\end{align*}
\]
KOY/GL: Security Analysis

- Commitment:
  \[ c = \text{Commit}(pw) = \text{Encrypt}(pke, pw) \]

- \[ \text{IND-CCA} \Rightarrow \text{NM for multiple commitments} \]

- Smooth Projective Hash Functions:
  \[ H(c, pw) = \text{Hash}(hk; c, pw) = \text{ProjHash}(hp; c, pw) \]

- No information about \( H(c, pw) \) if \( pw \neq \text{Decrypt}(sk, c) \)

- Hard to compute \( H(c, pw') \) without either the hash-key \( hk \) or the witness \( r \)

- Session Key:
  \[ c_1 = \text{Encrypt}(pke, pw, r_1) \quad c_2 = \text{Encrypt}(pke, pw, r_2) \]
  \[ sk = \text{Hash}(hk; c_1, pw) + \text{ProjHash}(hp; c_2, pw, r_2) = \text{ProjHash}(hp; c_1, pw, r_1) + \text{Hash}(hk; c_2, pw) \]

KOY/GL: Security Analysis

- Passive Adversary:
  Pseudo-randomness without the witness \( \Rightarrow \) indistinguishability of the session key

- Active Adversary:
  \[ \text{NM for multiple commitments} \Rightarrow \text{no new valid commitment (except chance with } pw \text{)} \]

- Invalid commitment \( \Rightarrow \) indistinguishability of \( sk \) (statistic)

- Replay of commitment: does not know the witness \( \Rightarrow \) indistinguishability of \( sk \) (computational)
KOY/GL: Security Analysis

**Proof: with an extractable commitment**

Adversary sends $c_1$: we can extract the password, and check whether it is correct or not

Simulator sends $c_1$: with a random/dummy pw!

Wrong simulation if adversary has guessed pw

Not negligible and thus not UC secure

---

**UC Password-Based AKE**

- $c_1, vk$
- $c_2, hp$
- $hp', \sigma$

---

- $\text{Some encryption and decryption}$
- $\text{Some hashing and verification}$
- $\text{Some key derivation}$
**CHKLMK: Idea**

- **Idea**: CHKLMK introduces a new approach to PAKE in the UC framework.
- **Proof**: Utilizes an extractable commitment scheme.
- **Adversary**: Sends a commitment $c_0$.
- **Simulator**: Sends a random/dummy password $pw'$.
- **Session Key**: Extracts and checks the password $pw$.
  - If $pw$ is incorrect, a random key is used.
  - If $pw$ is correct, the correct password is committed in $c_2$ and a fake ZKP is simulated.

---

**UC PAKE**

[Canetti-Halevi-Katz-Lindell-MacKenzie – EC '05]
Adaptive Adversary

- An adaptive adversary can corrupt players at any time and receive the internal state in a KOY/GL-like scheme: not secure.
- If corruption right after that: how to simulate?

EKE Scheme

<table>
<thead>
<tr>
<th>Client U</th>
<th>Server S</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x \leftarrow [1; q-1]$</td>
<td>$y \overset{R}{\leftarrow} [1; q-1]$</td>
</tr>
<tr>
<td>(U1) $X \leftarrow g^x$</td>
<td>$U, X$</td>
</tr>
<tr>
<td>(S2) $Y \leftarrow g^y$</td>
<td>$Y^* \leftarrow E_{pw}(Y)$</td>
</tr>
<tr>
<td>$K_U \leftarrow Y^*$</td>
<td>$S, Y^*$</td>
</tr>
<tr>
<td>$K_S \leftarrow X^y$</td>
<td>$K_S$</td>
</tr>
<tr>
<td>$\text{Auth} \leftarrow H_1(\text{ssid}</td>
<td></td>
</tr>
<tr>
<td>$sk_U \leftarrow H_0(\text{ssid}</td>
<td></td>
</tr>
<tr>
<td>completed</td>
<td>(S4) if (Auth = $H_1(\text{ssid}</td>
</tr>
<tr>
<td>$sk_S \leftarrow H_0(\text{ssid}</td>
<td></td>
</tr>
</tbody>
</table>