PAKE in the UC-Framework

Adaptive Security

CANS '2007

Singapore

Sunday, December 9th, 2007

David Pointcheval
CNRS-ENS-INRIA
Paris - France

Universal Composability

- **Universal Composability**
- Password-Based AKE
- UC Password-Based AKE

Provable Security

Security proofs give the guarantee that an assumption is **enough** for security:

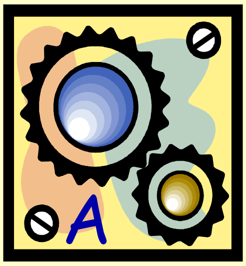
- if an adversary can break the system
- one can break the assumption

$\Rightarrow$ "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**

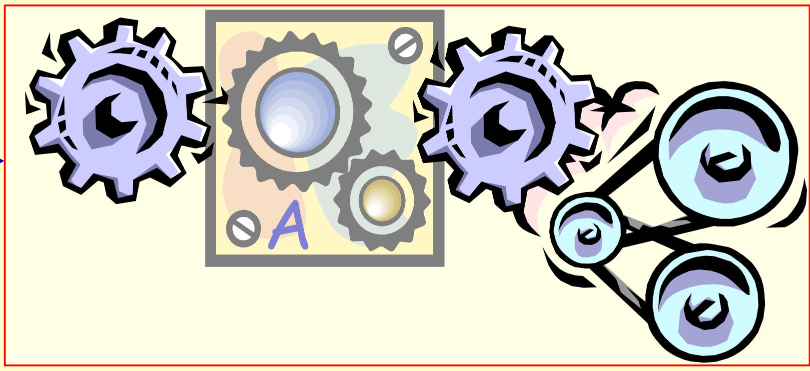


Proof by Reduction

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**

Instance **I** of **P** →



→ Solution of **I**

P intractable ⇒ scheme unbreakable

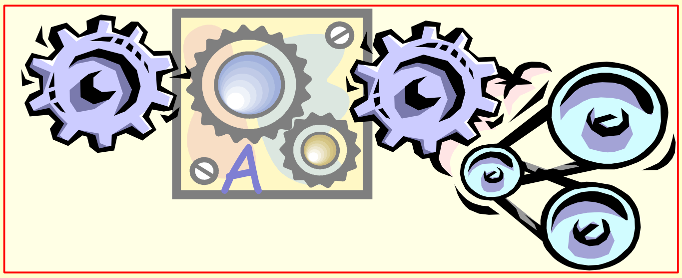
Provably Secure Scheme

To prove the security of a cryptographic scheme, one has to make precise

- the algorithmic assumptions
- the security notions to be guaranteed
- a reduction: an adversary can help to break the assumption

Simulation

In such a reduction,
our simulator tries to emulate the environment, until the adversary may win the attack game



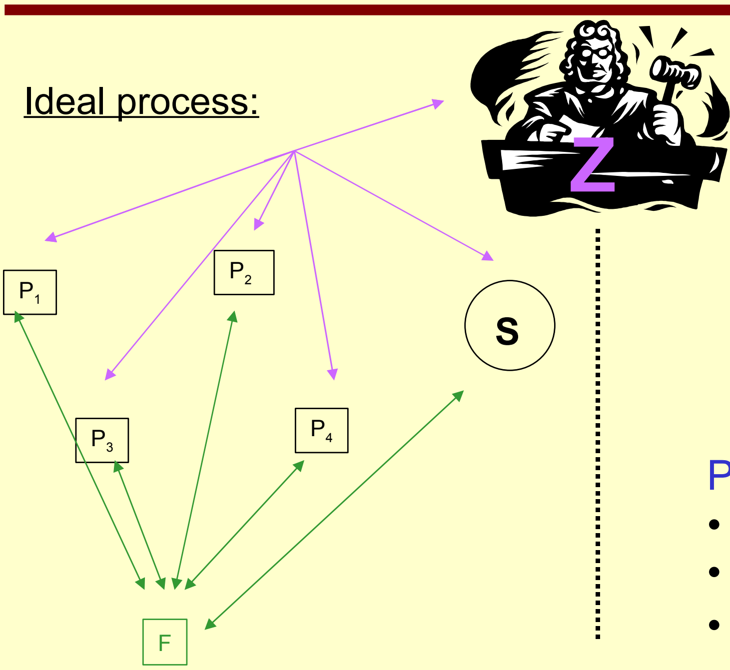
What about the composition of multiple protocols?

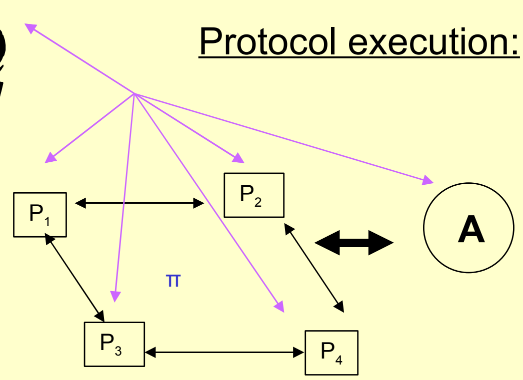
- the simulation fails as soon as an adversary may break one part of the global system, whereas other parts may provide a protection
- other executing protocols may provide additional information to the adversary

⇒ either we re-prove the global system,
or we prove each component in the **UC Framework**

Universal Composability

[Canetti - FOCS '01]





Protocol π securely realizes F if:

- For any adversary **A**
- There exists a simulator **S**
- Such that no environment **Z** can tell whether it interacts with:
  - a run of π with **A**
  - an ideal run with F and **S**

Real vs. Ideal

Definition of security

Protocol π emulates the ideal process for F if

- for any adversary **A**
- there exists a simulator **S**
- such that for all **Z**

$$\mathrm{IDEAL}^{F}_{\mathbf{S},\mathbf{Z}} \sim \mathrm{EXEC}_{\pi,\mathbf{A},\mathbf{Z}} .$$

⇒ we say that protocol π securely realizes F.

$$(\forall\ \mathbf{A})\ (\exists\ \mathbf{S})\ (\forall\ \mathbf{Z})\ \mathrm{IDEAL}^{F}_{\mathbf{S},\mathbf{Z}} \sim \mathrm{EXEC}_{\pi,\mathbf{A},\mathbf{Z}}.$$

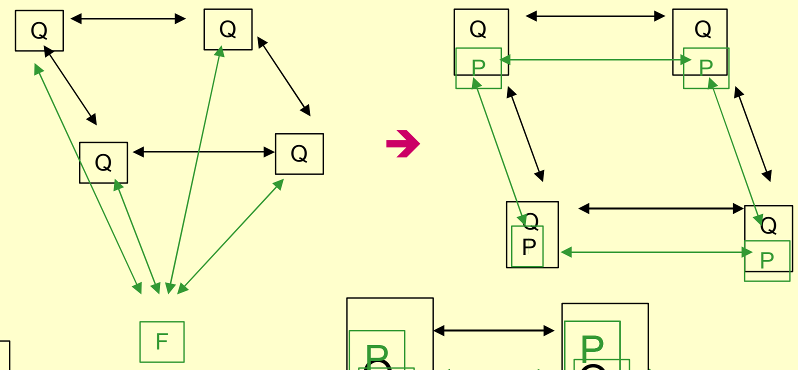
Equivalently:

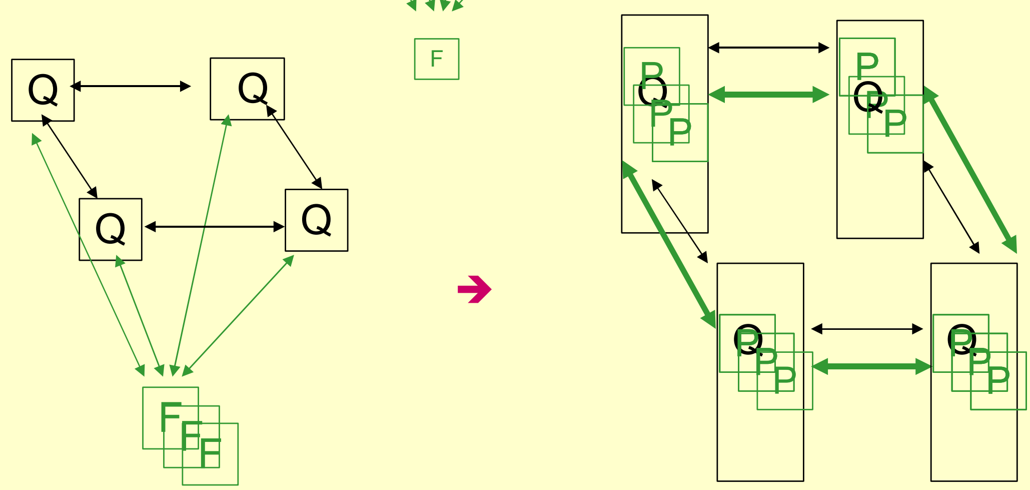
$$(\exists\ \mathbf{S}_d)\ (\forall\ \mathbf{Z})\ \mathrm{IDEAL}^{F}_{\mathbf{S},\mathbf{Z}} \sim \mathrm{EXEC}_{\pi,\mathbf{A}_d,\mathbf{Z}}$$

$$(\forall\ \mathbf{A})\ (\forall\ \mathbf{Z})\ (\exists\ \mathbf{S})\ \mathrm{IDEAL}^{F}_{\mathbf{S},\mathbf{Z}} \sim \mathrm{EXEC}_{\pi,\mathbf{A},\mathbf{Z}}$$

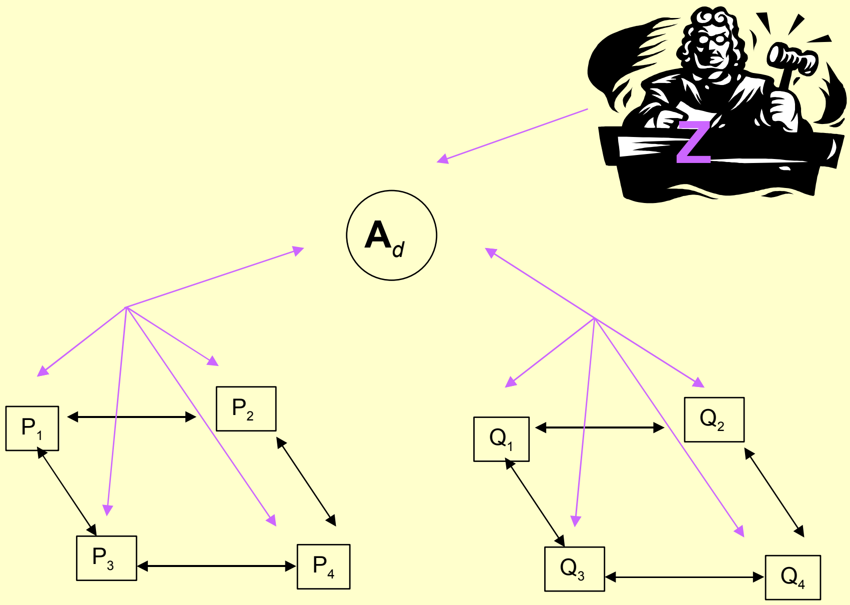
UC Theorem: Composition

Modular composition

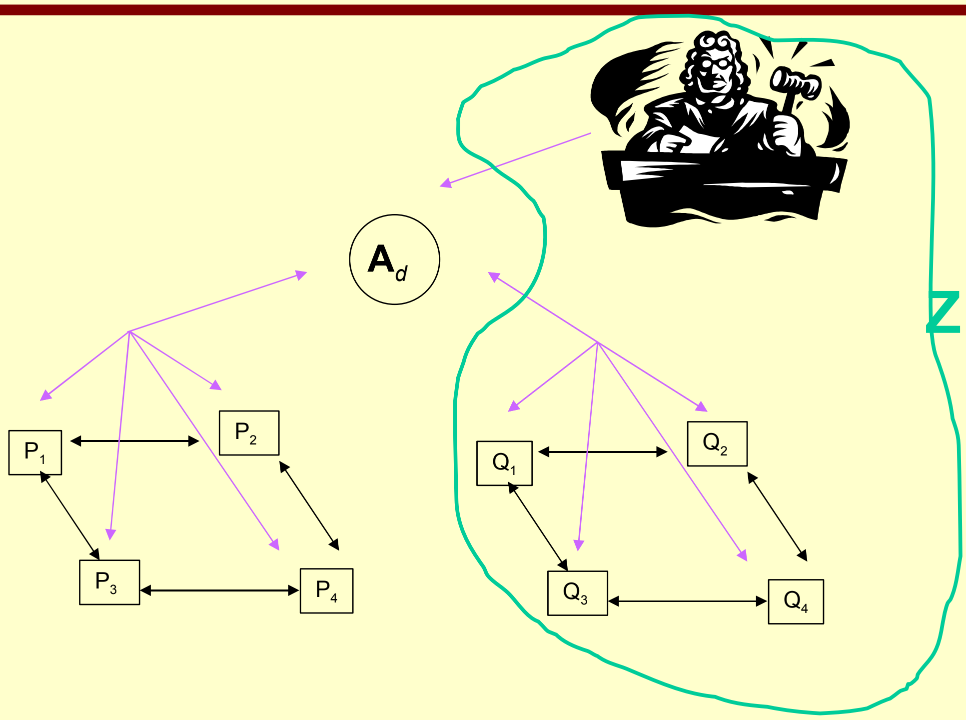




UC Theorem: Idea

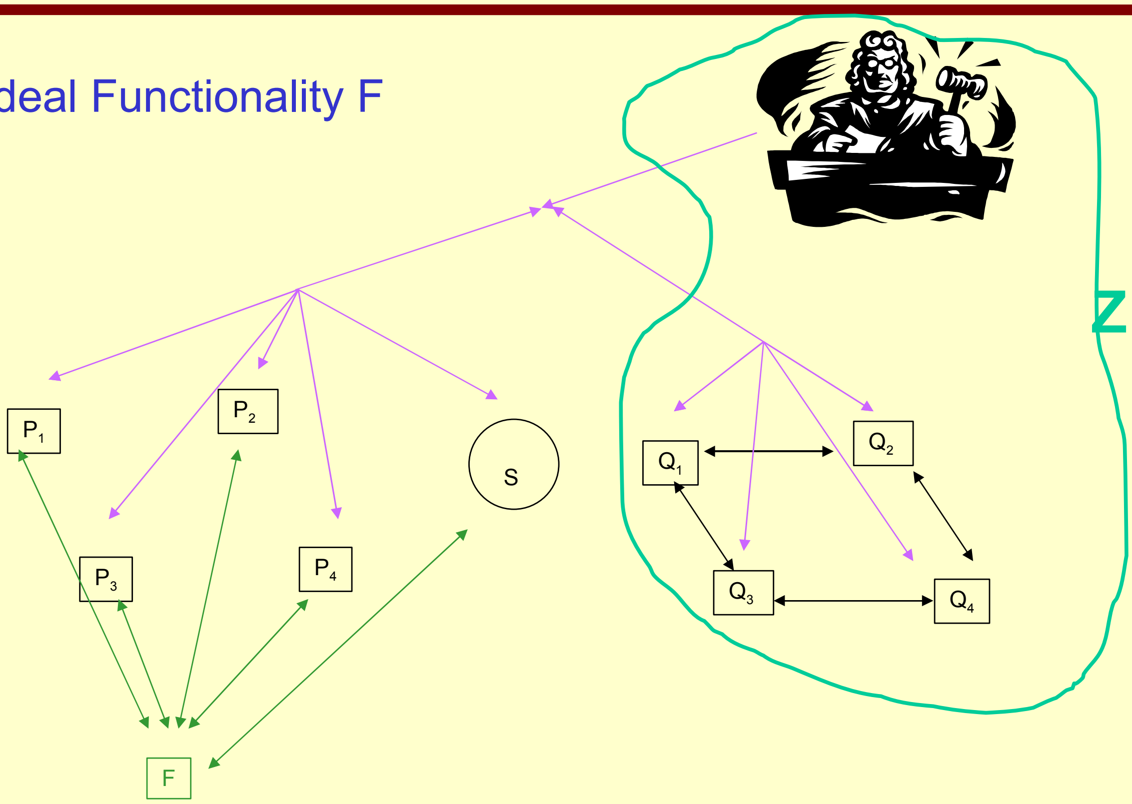


UC Theorem: Idea



UC Theorem: Idea

Ideal Functionality F



Implications of UC

Can design and analyze protocols in a modular way:

- Partition a given task T to simpler sub-tasks $T_1 \ldots T_k$
- Construct protocols for realizing $T_1 \ldots T_k$.
- Construct a protocol for T assuming ideal access to $T_1 \ldots T_k$.
- Use the composition theorem to obtain a protocol for T from scratch.

(Now can be done concurrently and in parallel.)

Password-Based AKE

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

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

$\mathcal{F}_{\mathsf{KE}}$ 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 $(\mathsf{NewSession}, sid, P_i, P_j, \text{role})$ **from party** P_i**:**

Send $(\mathsf{NewSession}, sid, P_i, P_j, \text{role})$ 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 $(\mathsf{NewKey}, sid, P_i, sk)$ **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 $\mathcal{F}_{\mathsf{KE}}$

Password-Based Authentication

- **Asymmetric**: (sk_A, pk_A) and possibly (sk_B, pk_B)
  - they authenticate to each other using the knowledge of the private key associated to the certified public key
- **Symmetric**: common (long – high-entropy) secret
  - they use the long term secret to derive a secure and authenticated ephemeral key sk
- **Password**: common (short - low-entropy) secret
  - let us assume a **20-bit** password
  - ⇒ it is possible to win with non-negligible advantage

Ideal Functionality: pwKE

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

Functionality $\mathcal{F}_{\mathsf{pwKE}}$

The functionality $\mathcal{F}_{\mathsf{pwKE}}$ 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 $(\mathsf{NewSession}, sid, P_i, P_j, pw, \mathrm{role})$ **from party** P_i**:**

Send $(\mathsf{NewSession}, sid, P_i, P_j, \mathrm{role})$ 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, pw'), then record (P_i, P_j, pw) and mark this record fresh.

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

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

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

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

- If this record is compromised, or either P_i or P_j is corrupted, then output (sid, sk) to player P_i.
- If this record is fresh, and there is a record (P_j, P_i, pw') with $pw' = pw$, and a key sk' was sent to P_j, and (P_j, P_i, pw) was fresh at the time, then 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.

Either way, mark the record (P_i, P_j, pw) as completed.

Figure 2: The password-based key-exchange functionality $\mathcal{F}_{\mathsf{pwKE}}$

Concurrent Executions

In this ideal functionality:

- **TestPwd** query, which gives the authorization to the adversary to test **one** password per session
- In case of correct password guess, the adversary can choose the 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

KOY/GL Protocol

P_i (client)		P_j (server)
	CRS: pke	
		$(sk, vk) \leftarrow \text{sigKey}(\$)$
	$\xleftarrow{\quad c_1, vk \quad}$	$c_1 \leftarrow E_{pke}(pw; r_1)$
$c_2 \leftarrow E_{pke}(pw, r_2)$ $hk \leftarrow \mathcal{H}$ $hp \leftarrow \alpha(hk; c_1)$	$\xrightarrow{\quad c_2, hp \quad}$	
		$hk' \leftarrow \mathcal{H}$ $hp' \leftarrow \alpha(hk'; c_2)$
	$\xleftarrow{\quad hp', \sigma \quad}$	$\sigma \leftarrow \mathsf{Sign}_{sk}(c_2, hp, hp')$
if $(\mathsf{Verify}_{vk}((c_2, hp, hp'), \sigma) = 1)$ session-key $\leftarrow H_{hk}(c_1, pw)$ $+ h_{hp'}(c_2, pw; r_2)$		session-key $\leftarrow h_{hp}(c_1, pw; r_1)$ $+ H_{hk'}(c_2, pw)$

KOY/GL: Security Analysis

- Commitment:
  - c = Commit(pw,r) = Encrypt(pke, pw,r)
  - IND-CCA ⇒ NM for multiple commitments
- Smooth Projective Hash Functions:

  H(c,pw) = Hash(hk;c,pw) = ProjHash(hp;c,pw;r)
  - No information about H(c,pw) if $pw \neq$ Decrypt(ske,c)
  - Hard to compute H(c,pw') without
    either the hash-key hk or the witness r
- Session Key:

  c_1= Encrypt(pke, pw, r_1) c_2= Encrypt(pke, pw, r_2)

  sk = Hash(hk_2;c_1,pw) + ProjHash(hp_1;c_2,pw;r_2)

  = ProjHash(hp_2;c_1,pw;r_1) + Hash(hk_1;c_2,pw)

KOY/GL: Security Analysis

- Passive Adversary:
  - Pseudo-randomness without the witness
    ⇒ indistinguishability of the session key
- Active Adversary:
  - NM for multiple commitments
    ⇒ no new valid commitment (except chance with pw)
  - Invalid commitment
    ⇒ indistinguishability of sk (statistic)
  - Replay of commitment: does not know the witness
    ⇒ indistinguishability of sk (computational)

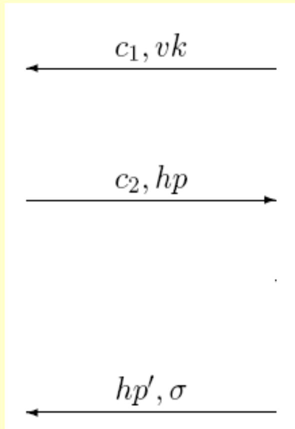
KOY/GL: Security Analysis

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*!
  - adversary sends c_2: extract and check
    - wrong ⇒ random key
    - correct ⇒ we get stuck

  Wrong simulation if adversary has guessed *pw*

  Not negligible and thus not UC secure



UC PAKE

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

P_i (client)	CRS: pke	P_j (server)
$c_0 \leftarrow E_{pke}(pw; r_0)$	c_0 →	
	← c_1, vk	$(sk, vk) \leftarrow \text{sigKey}(\$)$ $c_1 \leftarrow E_{pke}(pw; r_1)$
$c_2 \leftarrow E_{pke}(pw, r_2)$ $hk \leftarrow \mathcal{H}$ $hp \leftarrow \alpha(hk; c_1)$	c_2, hp →	
	$\text{ZKP}(c_0 \approx c_2)$ →	
	← hp', σ	$hk' \leftarrow \mathcal{H}$ $hp' \leftarrow \alpha(hk'; c_2)$ $\sigma \leftarrow \mathsf{Sign}_{sk}(c_2, hp, hp')$
if $(\mathsf{Verify}_{vk}((c_2, hp, hp'), \sigma) = 1)$ session-key $\leftarrow H_{hk}(c_1, pw) + h_{hp'}(c_2, pw; r_2)$		session-key $\leftarrow h_{hp}(c_1, pw; r_1) + H_{hk'}(c_2, pw)$

CHKLMK: Idea

UC Proof: with an extractable commitment

- Adversary sends c_0: we can extract the password, and check whether it is correct or not
- Simulator sends c_0: with a random/dummy *pw*!
  - adversary sends c_1: extract and check *pw*
    - wrong ⇒ random key
    - correct ⇒ we commit the correct password in c_2 and simulate a fake ZKP

c_0	→
c_1, vk	←
c_2, hp	→
$\text{ZKP}(c_0 \approx c_2)$	→

Adaptive Adversary

An adaptive adversary can corrupt players
at any time and receive the internal state

- in KOY/GL-like scheme: not secure
  - in the simulation, use of "dummy password" for c_0
  - if corruption right after that: how to simulate r_0?
- in EKE-like scheme: secure
  - granted the Programmability
    of the Ideal-Cipher and the Random Oracle

  ⇒ Adaptive adversaries and strong corruption

  [Abdalla-Catalano-Chevalier-Pointcheval – CT-RSA '08]

EKE Scheme

Client U		Server S
$x \stackrel{R}{\leftarrow} [\![1 ; q-1]\!]$		$y \stackrel{R}{\leftarrow} [\![1 ; q-1]\!]$
(U1) $X \leftarrow g^x$	$\xrightarrow{U,X}$	
		(S2) $Y \leftarrow g^y$
		$Y^* \leftarrow \mathcal{E}_{pw}(Y)$
	$\xleftarrow{S,Y^*}$	$K_S \leftarrow X^y$
(U3) $Y = \mathcal{D}_{pw}(Y^*)$		
$K_U \leftarrow Y^x$		
$Auth \leftarrow \mathcal{H}_1(ssid \Vert U \Vert S \Vert X \Vert Y \Vert K_U)$		
$sk_U \leftarrow \mathcal{H}_0(ssid \Vert U \Vert S \Vert X \Vert Y \Vert K_U)$		
completed	$\xrightarrow{Auth}$	
		(S4) if $(Auth = \mathcal{H}_1(ssid \Vert U \Vert S \Vert X \Vert Y \Vert K_S))$
		then completed
		$sk_S \leftarrow \mathcal{H}_0(ssid \Vert U \Vert S \Vert X \Vert Y \Vert K_S)$
		else error