Password-based Authenticated Key Exchange

David Pointcheval

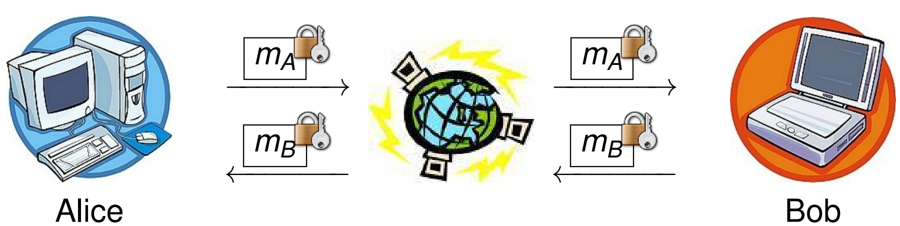
Ecole Normale Supérieure

PKC 2012
Darmstadt, Germany
May 22nd, 2012

Outline

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



How to obtain such a shared secret key? $\longrightarrow$ Key exchange protocols

Diffie-Hellman Key Exchange

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

$$\begin{array}{ccc} x \stackrel{\$}{\leftarrow} \mathbb{Z}_p, X \leftarrow g^x & \xrightarrow{\quad X \quad} & y \stackrel{\$}{\leftarrow} \mathbb{Z}_p, Y \leftarrow g^y \\ K \leftarrow Y^x = g^{xy} & \xleftarrow{\quad Y \quad} & K \leftarrow X^y = g^{xy} \end{array}$$

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

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

$\Longrightarrow$ low-entropy shared secret:
a password *pw*

BAC: Basic Access Control

The symmetric encryption and MAC keys are derived from *pw*

$$
\begin{array}{lcl}
\textit{Passport} & & \textit{Reader} \\
r_P, k_P \stackrel{\$}{\leftarrow} \{0,1\}^{64} & \xrightarrow{\quad r_P \quad} & r_R, k_R \stackrel{\$}{\leftarrow} \{0,1\}^{64} \\
 & & C_R \leftarrow Enc_{pw}(r_R, r_P, k_R) \\
C_P \leftarrow Enc_{pw}(r_P, r_R, k_P) & \xleftarrow{\quad C_R, M_R \quad} & M_R \leftarrow Mac_{pw}(C_R) \\
M_P \leftarrow Mac_{pw}(C_P) & \xrightarrow{\quad C_P, M_P \quad} & \\
 & K \leftarrow k_P \oplus k_R &
\end{array}
$$

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?

Off-line Dictionary Attacks

As in the previous scenario, after having

- eavesdropped some (possibly many) transcripts
- interacted (quite a few times) with players

the adversary accumulates enough information
to take the real password apart from the dictionary
efficient password-recovery after off-line exhaustive search

For the BAC: quite a few passive eavesdroppings are enough
to recover the password!

How many active interactions could one enforce?

On-line Dictionary Attacks

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

In a finite group $\mathbb{G}$, of prime order p, with key derivation function $\mathcal{K}$

EKE: Encrypted Key Exchange [Bellovin–Merritt, 1992]

$$\begin{array}{lcl} x \xleftarrow{\$} \mathbb{Z}_p, X \leftarrow g^x & & y \xleftarrow{\$} \mathbb{Z}_p, Y \leftarrow g^y \\ X' \leftarrow \mathcal{E}_{\mathsf{pw}}(X) & \xrightarrow{\quad X\ X' \quad} & X \leftarrow \mathcal{D}_{\mathsf{pw}}(X') \\ Y \leftarrow \mathcal{D}_{\mathsf{pw}}(Y') & \xleftarrow{\quad Y\ Y' \quad} & Y' \leftarrow \mathcal{E}_{\mathsf{pw}}(Y) \\ k \leftarrow Y^x = g^{xy} & & k \leftarrow X^y = g^{xy} \\ K \leftarrow \mathcal{K}(A,B,X,Y,k) & & K \leftarrow \mathcal{K}(A,B,X',Y',k) \end{array}$$

DH Key Exchange with flows encrypted under pw

SPEKE: Simple Password Exponential Key Exchange [Jablon, 1996]

$$\begin{array}{lcl} & g \leftarrow \mathcal{G}(A,B,\mathsf{pw}) & \\ x \xleftarrow{\$} \mathbb{Z}_p, X \leftarrow g^x & \xrightarrow{\quad X \quad} & y \xleftarrow{\$} \mathbb{Z}_p, Y \leftarrow g^y \\ k \leftarrow Y^x = g^{xy} & \xleftarrow{\quad Y \quad} & k \leftarrow X^y = g^{xy} \\ K \leftarrow \mathcal{K}(A,B,X,Y,k) & & K \leftarrow \mathcal{K}(A,B,g,X,Y,k) \end{array}$$

DH Key Exchange with a basis derived from pw

PACE: Password Authenticated Connection Establishment

The recent alternative to BAC is PACE:
Password Authenticated Connection Establishment
In the spirit of SPEKE: a generator derived from the password

With security analyses:

- PACE v1 [Bender–Fischlin–Kuegler, 2009]
- PACE v2 [Coron–Gouget–Icart–Paillier, 2011]

What does security really mean?

Security Models

- Game-based Security [Bellare–P.–Rogaway, 2000]
  - Find-then-Guess
  - Real-or-Random [Abdalla–Fouque–P., 2005]
- Simulation-based Security [Boyko–MacKenzie–Patel, 2000]
- Universal Composability [Canetti–Halevi–Katz–Lindell–MacKenzie, 2005]

Where

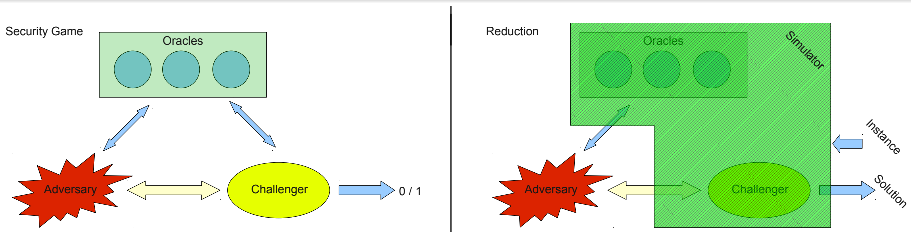
- The adversary controls all the communications:
  It can create, modify, transfer, alter, delete messages
- Users can participate in concurrent executions of the protocol
  Instances of the players are denoted A^i and B^j

On-line dictionary attack should be the best attack
$\Longrightarrow$ No adversary should win with probability greater than q_S/N
where $q_S = \#$Active Sessions and $N = \#$Dictionary

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)



Proof by contradiction

Game-based Security: PAKE

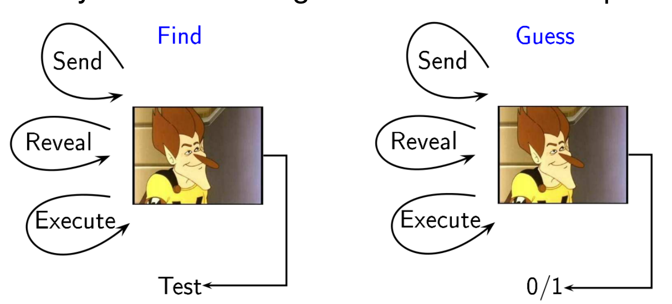
[Bellare–P.–Rogaway, 2000]

The adversary $\mathcal{A}$ interacts with oracles:

- $Execute(A^i, B^j)$
  $\mathcal{A}$ gets the transcript of an execution between A and B
  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 (active sessions)
- $Reveal(U^i)$
  $\mathcal{A}$ 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 (*i.e.* $Reveal(U^i)$)
  - If $b = 1$, $\mathcal{A}$ 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



$$Adv^{FtG}(\mathcal{A}) = 2 \times \Pr[b' = b] - 1 \leq \frac{\mathcal{O}(q_S)}{N} + negl()$$

Security Games: Advanced Security Notions

- Semantic Security
  The Find-then-Guess game models the **secrecy** of the key
  $\Longrightarrow$ 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
  $\Longrightarrow$ 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

Secure Protocols: EKE-like

With both Random Oracles and an Ideal Cipher

- EKE (ROM+ICM) [Bellare–P.–Rogaway, 2000]
  $\Longrightarrow$ with Forward-Secrecy
- OEKE (ROM+ICM) [Bresson–Chevassut–P., 2003]
  $\Longrightarrow$ with Forward-Secrecy and Client-Authentication
  Formally verified with CryptoVerif [Blanchet, 2012]

With Random Oracles (and One-time Pad)

- OMDHKE (ROM) [Bresson–Chevassut–P., 2004]
  $\Longrightarrow$ with Forward-Secrecy and Server-Authentication
- SPAKE (ROM)
  [Abdalla–P., 2005]

$$\begin{array}{ccc}
x \stackrel{\$}{\leftarrow} \mathbb{Z}_p, X \leftarrow g^x & & y \stackrel{\$}{\leftarrow} \mathbb{Z}_p, Y \leftarrow g^y \\
X' \leftarrow X \cdot h^{\mathsf{pw}} & \xrightarrow{X'} & X \leftarrow X' / h^{\mathsf{pw}}, k \leftarrow X^y \\
Y \leftarrow Y' / h^{\mathsf{pw}}, k \leftarrow Y^x & \xleftarrow{Y'} & Y' \leftarrow Y \cdot h^{\mathsf{pw}} \\
\multicolumn{3}{c}{K \leftarrow \mathcal{K}(A, B, X', Y', \mathsf{pw}, k)}
\end{array}$$

Quite Simple Scheme

Smooth Projective Hash Functions

Definition [Cramer–Shoup, 2002] [Gennaro–Lindell, 2003]

Let $\{H\}$ be a family of functions from X to $\mathbb{G}$
and L a subset (language) of this domain X
such that, for any point $x \in L$, and a witness w,

- $H(x) = \mathsf{Hash}_L(\mathsf{hk}; x)$, with the secret hashing key hk
- $H(x) = \mathsf{ProjHash}_L(\mathsf{hp}; x, w)$, with the public projected key hp

- Hard-Partitioned Subset: L and X hard to distinguish
- Smoothness: if $x \notin L$, $H(x)$ and hp are independent
- Pseudo-Randomness: if $x \in L$, $H(x)$ is pseudo-random, with hp but without a witness w

Secure Protocols: KOY/GL-like

With $\mathcal{L}$ = language of the valid commitments of pw

GL (Standard + CRS)
[Gennaro–Lindell, 2003]
$\Longrightarrow$ Forward-secrecy

(main steps – more details are required)

$$\begin{array}{ccc}
C_1 \leftarrow \mathit{Commit}(\mathsf{pw}; r_1) & \xrightarrow{C_1} & C_2 \leftarrow \mathit{Commit}(\mathsf{pw}; r_2) \\
& \xleftarrow{C_2,\, hp_1} & hk_1, hp_1 \text{ on } C_1 \\
hk_2, hp_2 \text{ on } C_2 & \xrightarrow{hp_2} & \\
\multicolumn{3}{c}{\mathsf{ProjHash}(hp_1; C_1, r_1) = H_1 = \mathsf{Hash}(hk_1; C_1)} \\
\multicolumn{3}{c}{\mathsf{Hash}(hk_2; C_2) = H_2 = \mathsf{ProjHash}(hp_2; C_2, r_2)} \\
\multicolumn{3}{c}{K \leftarrow H_1 \cdot H_2}
\end{array}$$

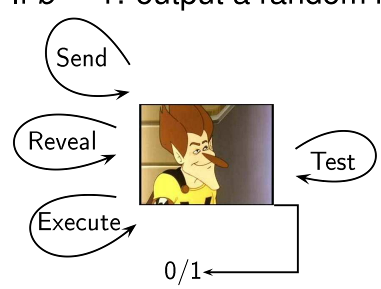
Generalization of the KOY protocol [Katz–Ostrovsky–Yung, 2001]

With hp_1 and hp_2 independent of C_1 and C_2 resp.
$\Longrightarrow$ can be made in One-Round only [Katz–Vaikuntanathan, 2011]

Security Game: Real-or-Random [Abdalla–Fouque–P., 2005]

Secrecy/independence of all the keys:
many *Test*-queries on any U^i with the same bit b

- If no key defined by the protocol yet: output $\perp$
- 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



$$Adv^{RoR}(\mathcal{A}) = 2 \times \Pr[b' = b] - 1$$

Security Game: Real-or-Random

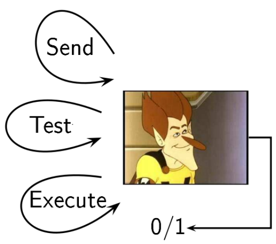
Semantic Security (Encryption) [Bellare–Desai–Jokipii–Rogaway, 1997]

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

$$Adv^{RoR}(t, q_T) \leq q_T \times Adv^{FtG}(t)$$

where q_T is the number of Test-queries

For Password-based Authenticated Key Exchange:



$Adv^{FtG}(t) \leq \frac{\mathcal{O}(q_S)}{N}$

$\not\Rightarrow Adv^{RoR}(t, q_T) \leq \frac{\mathcal{O}(q_S)}{N}$

$\Longrightarrow$ Much stronger notion

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

$\Longrightarrow$ Much simpler security notion

Game-based Security: Limitations

- Proven bound: $\mathcal{O}(q_S)/N$, but almost never q_S/N
  $\Longrightarrow$ 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

$\Longrightarrow$ Universal Composability more appropriate? [Canetti, 2001]

It extends the Simulation-based Security [Boyko–MacKenzie–Patel, 2000]

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:

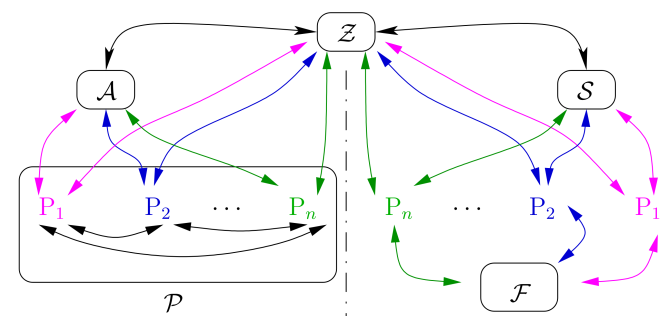
- it takes as input $x_1, \ldots, x_n$,
  the private information of each player,
- and outputs $y_1, \ldots, y_n$, given privately to each player

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

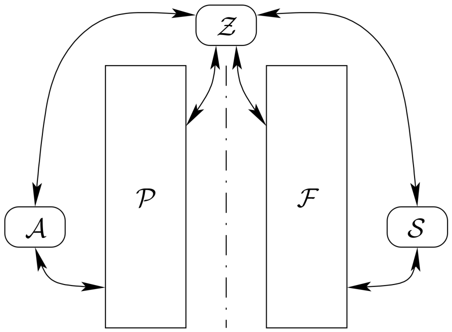
Simulator

$\mathcal{P}$ emulates $\mathcal{F}$ if, for any environment $\mathcal{Z}$, 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 $\mathcal{P}$
- $\mathcal{S}$ attacking the ideal functionality $\mathcal{F}$



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 $\mathcal{P}$

PAKE Ideal Functionality [Canetti–Halevi–Katz–Lindell–MacKenzie, 2005]

Queries

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

Corruption-Query

- $\mathcal{A}$ gets the long-term secrets (pw) and the internal state
- $\mathcal{A}$ 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

PAKE Ideal Functionality

Session Key [Canetti–Halevi–Katz–Lindell–MacKenzie, 2005]

- 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

PAKE Ideal Functionality

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
  $\Longrightarrow$ 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 ($pw \Longrightarrow$ 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 ($pw \Longrightarrow$ 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*) [Canetti–Halevi–Katz–Lindell–MacKenzie, 2005]
  $\Longrightarrow$ Static Corruptions
- With an equivocable/extractable commitment (*bit-by-bit*)
  $\Longrightarrow$ GL secure against Adaptive Corruptions [Abdalla–Chevalier–P., 2009]
- With hp independent of the commitment (*with NIZK*)
  $\Longrightarrow$ one-round only [Groce–Katz, 2010]
  [Katz–Vaikuntanathan, 2011]

With random oracles and an ideal cipher:

OEKE
[Abdalla–Catalano–Chevalier–P., 2008]

$$x \xleftarrow{\$} \mathbb{Z}_p, X \leftarrow g^x \xrightarrow{A, X} y \xleftarrow{\$} \mathbb{Z}_p, Y \leftarrow g^y$$
$$Y \leftarrow \mathcal{D}_{pw}(Y'), K = Y^x \xleftarrow{Y'} Y' \leftarrow \mathcal{E}_{pw}(Y), K = X^y$$
$$Auth = \mathcal{H}(A, B, X, Y', K) \xrightarrow{Auth} Auth \stackrel{?}{=} \mathcal{H}(A, B, X, Y', K)$$
$$sk = \mathcal{K}(A, B, X, Y', K)$$

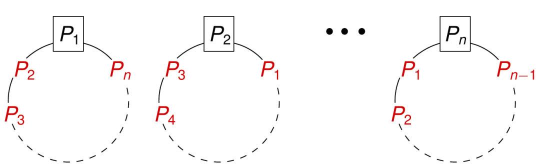
$\Longrightarrow$ First efficient scheme secure against Adaptive Corruptions

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



Split Functionality: initiates a sub-functionality for each sub-session

- Real player P_i : P_i non-corrupted at the beginning
- Adversary on behalf of P_j: P_j corrupted from the beginning

GPAKE: Each sub-session allows to test one password

Limitations of the `NewKey`-Query

Session Key: `NewKey`-Query

...
- a corrupted player $\Rightarrow$ key chosen by the adversary
- correct password guess $\Rightarrow$ 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
  $\Longrightarrow$ some information leaks!

Contributiveness

[Adalla–Catalano–Chevalier–P., 2009]

Initial Definition of the Session Key

- no corrupted players, same passwords ⇒ same random key
- corrupted player or correct *TestPwd* ⇒ key chosen by $\mathcal{A}$
- otherwise ⇒ independent random keys

With Contributiveness

- at least one non-corrupted player, same passwords ⇒ same random key
- all players corrupted ⇒ key chosen by $\mathcal{A}$
- 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 $\Longrightarrow$ 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 $\mathcal{A}$ joins the protocol with a password on behalf of a player $\Longrightarrow \mathcal{A}$ impersonates P_i: it receives the messages for it
- `NewKey` = $\mathcal{A}$ asks for the session key to be generated
- `SendKey` = $\mathcal{A}$ asks for the session key to be delivered

`NewKey`-Query

- the two players are controlled by the adversary ⇒ No need to inform anybody: the adversary plays alone!
- Same passwords ⇒ same random key – $\mathcal{A}$ informed: `OK`
- otherwise ⇒ $\perp$ – $\mathcal{A}$ informed: `NOK`

More general $\Longrightarrow$ 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), $\Longrightarrow$ 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 $\mathcal{A}$ (for a player) joins the protocol with
  - its own language parameters: *Pub* and *Priv*
  - its partner's language parameters: *Pub′* and *Priv′*
  - its word *w*
- `NewKey` = $\mathcal{A}$ asks for the session key to be generated
- `SendKey` = $\mathcal{A}$ asks for the session key to be delivered

Consistent Inputs

The protocol succeeds with the same key if and only if

$$\begin{array}{cc} (Pub_a, Priv_a) = (Pub'_b, Priv'_b), & (Pub_b, Priv_b) = (Pub'_a, Priv'_a) \\ w_a \in L_a(Pub_a, Priv_a), & w_b \in L_b(Pub_b, Priv_b) \end{array}$$

LAKE: General Approach

Verification

- $Pub_a = Pub'_b$ & $Pub_b = Pub'_a$: public matching verification
- $Priv_a = Priv'_b$ & $Priv_b = Priv'_a$: implicit matching verification
  $\Longrightarrow$ as in PAKE
- $w_a \in L_a(Pub_a, Priv_a)$ & $w_b \in L_b(Pub_b, Priv_b)$: implicit verification
  $\Longrightarrow$ 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

(many more details are required)

$$\begin{array}{ccc} C_1 \leftarrow Commit(\cdot; r_1) & \xrightarrow{C_1} & C_2 \leftarrow Commit(\cdot; r_2) \\ & \xleftarrow{C_2, hp_1} & hk_1, hp_1 \text{ on } C_1 \\ hk_2, hp_2 \text{ on } C_2 & \xrightarrow{hp_2} & \end{array}$$

$$\mathrm{ProjHash}(hp_1; C_1, r_1) = H_1 = \mathrm{Hash}(hk_1; C_1)$$
$$\mathrm{Hash}(hk_2; C_2) = H_2 = \mathrm{ProjHash}(hp_2; C_2, r_2)$$
$$K \leftarrow H_1 \cdot H_2$$

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

- $Pub = \emptyset$, $Priv =$ pw and $L(Pub, Priv) = \{Priv\}$
  $\Longrightarrow$ PAKE (15 group elements exchanged)
  With $Priv = (g^{\mathrm{pw}}, h^{\mathrm{pw}})$: verifier-based PAKE (29 group elements)
- $Pub = M$, $Priv = vk$, $L(Pub, Priv) = \{\sigma, Verif(Priv, Pub, \sigma) = 1\}$
  $\Longrightarrow$ Secret Handshake [Balfanz–Durfee–Shankar–Smetters–Staddon–Wong, 2003]
  (43 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 $\Longrightarrow$ not used so far
- EKE Patent expired late 2011 $\Longrightarrow$ recent IETF RFC 6124

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