Authenticated Key Exchange

passwords, groups, low-power devices

Caen – March 2004
Joint work with Emmanuel Bresson
and Olivier Chevassut

David Pointcheval
CNRS-ENS, Paris, France

Summary

- Provable Security
- Authenticated Key Exchange
  - Security Model
  - Examples
  - Authentication
  - Password-based
- Group Key Exchange
  - Security Model
  - Example
  - Dynamic groups
Summary

- **Provable Security**
  - Authenticated Key Exchange
    - Security Model
    - Examples
    - Authentication
    - Password-based
  - Group Key Exchange
    - Security Model
    - Example
    - Dynamic groups

Algorithmic Assumptions

**necessary**

- $n = pq$: **public** modulus
- $e$: **public** exponent
- $d = e^{-1} \mod \varphi(n)$: **private**

**RSA Encryption**

- $E(m) = m^e \mod n$
- $D(c) = c^d \mod n$

If the RSA problem is easy, secrecy is not satisfied: anybody may recover $m$ from $c$
Algorithmic Assumptions sufficient?

Security proofs give the guarantee that the assumption is **enough** for secrecy:
- if an adversary can break the secrecy
- one can break the assumption
  ⇒ “reductionist” proof

Proof by Reduction

Reduction of a problem $\mathbf{P}$ to an attack $Atk$:
- Let $A$ be an adversary that breaks the scheme
- Then $A$ can be used to solve $\mathbf{P}$
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 $P$

$P$ intractable $\Rightarrow$ scheme unbreakable

Provably Secure Scheme

To prove the security of a cryptographic scheme, one has to make precise
- the algorithmic assumptions
  - the RSA problem, the Diffie-Hellman problems, ...
- the security notions to be guaranteed
  - depends on the scheme
- a reduction
  - an adversary can help to break the assumption
  - simulation of the « view » of the adversary
Summary

- Provable Security
- Authenticated Key Exchange
  - Security Model
    - Examples
    - Authentication
    - Password-based
- Group Key Exchange
  - Security Model
  - Example
  - Dynamic groups

Authenticated Key Exchange

Two parties (Alice and Bob) agree on a common secret key $sk$, in order to establish a secret channel

- Intuitive goal: implicit authentication
  - only the intended partners can compute the session key
- Formally: semantic security
  - the session key $sk$ is indistinguishable from a random string $r$, to anybody else
Further Properties

- **Mutual authentication**
  - They are both sure to actually share the secret with the people they think they do

- **Forward-secrecy**
  - Even if a long-term secret data is corrupted, previously shared secrets are still semantically secure

Semantic Security

- For breaking the semantic security, the adversary asks one test-query which is answered, according to a random bit $b$, by
  - the actual secret data $sk$ (if $b=0$)
  - a random string $r$ (if $b=1$)

$\Rightarrow$ the adversary has to guess this bit $b$
The Leakage of Information

- The protocol is run over a public network, then the transcripts are public:
  - an **execute**-query provides such a transcript to the adversary
- The secret data $sk$ may be misused (with a weak encryption scheme, ...):
  - the **reveal**-query is answered by this secret data $sk$

---

Passive/Active Adversaries

- **Passive adversary**: history built using
  - the **execute**-queries $\rightarrow$ transcripts
  - the **reveal**-queries $\rightarrow$ session keys
- **Active adversary**: entire control of the network
  - the **send**-queries
    - *active, adaptive adversary on concurrent executions*
    - to send message to Alice or Bob
    - (in place of Bob or Alice respectively)
    - to intercept, forward and/or modify messages
Security Model

As many **execute**, **send** and **reveal** queries as the adversary wants

But one **test**-query, with $b$ to be guessed...

---

Formal Model

Bellare-Rogaway model revisited by Shoup

A can ask:
- **send**-queries
- **reveal**-queries
- **execute**-queries
- **test**-query
- **corrupt**-queries

0/1
Forward Secrecy

Forward secrecy means that the adversary cannot distinguish a session key established \textit{before} any corruption of the long-term private keys:

- the \texttt{corrupt}-query is answered by the long-term private key of the corrupted party
- then the \texttt{test}-query must be asked on a session key established \textit{before} any \texttt{corrupt}-query

---

Freshness

\textit{sk} is \textbf{fresh} if it is \textit{known} by the players but not clearly known by the adversary.

\texttt{reveal} \hspace{1cm} \texttt{sk}

after a \texttt{reveal}-query, \textit{sk} is known

\texttt{corrupt}

after a \texttt{corrupt}-query, any future key is known
**Summary**

- **Provable Security**
- **Authenticated Key Exchange**
  - Security Model
  - **Examples**
    - Authentication
    - Password-based
- **Group Key Exchange**
  - Security Model
  - Example
  - Dynamic groups

---

**Diffie-Hellman Key Exchange**

The most classical key exchange scheme has been proposed by Diffie and Hellman:

- $\mathbf{G} = \langle g \rangle$, cyclic group of prime order $q$

- Alice chooses a random $x \in \mathbb{Z}_q$, computes and sends $X = g^x$

- Bob chooses a random $y \in \mathbb{Z}_q$, computes and sends $Y = g^y$

- They can both compute the value $K = Y^x = X^y$
**Properties**

- Without any authentication, no security is possible: man-in-the-middle attack
  - ⇒ some authentication is required
- If flows are **Strongly Authenticated** (ie. MAC or Signature), it provides the semantic security of the session key under the **DDH Problem**
- If one derives the session key as $sk = H(K)$, in the random oracle model, semantic security is relative to the **CDH Problem**

---

**Replay Attack**

No explicit authentication ⇒ replay attacks

- The adversary intercepts "Alice, $X, \textbf{Auth}(\text{Alice},X)"
- It can initiate a new session with it

Bob believes it comes from Alice
- Bob accepts the key, but does not share it with Alice
  - ⇒ **no mutual authentication**
- The adversary does not know the key either
  - ⇒ **still semantic security**
Summary

- Provable Security
- Authenticated Key Exchange
  - Security Model
  - Examples
- Authentication
  - Password-based
- Group Key Exchange
  - Security Model
  - Example
  - Dynamic groups

Mutual Authentication

Adding key confirmation rounds: mutual authentication

[Bellare-P.-Rogaway Eurocrypt ‘00]

Alice

\[ k_1 = H_1(\text{Alice}, \text{Bob}, SK) \]

Bob

\[ k_2 = H_2(\text{Alice}, \text{Bob}, SK) \]

\[ sk = H(\text{Alice}, \text{Bob}, X, Y, SK) \]

\[ k_1 \text{ correct?} \]

\[ k_2 \text{ correct?} \]
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

Asymmetric

- the most classical authentication mode is the signature (cf. SIGMA)
- the ability to decrypt (with an asymmetric encryption scheme) is also a way to provide authentication

*Mutual Authentication for Low-Power Devices*
[Jakobsson-P. - FC 01]
- Client: Schnorr signature with off-line pre-processing
  - very efficient signing process (for the client)
- Server: RSA decryption
  - efficient encryption process (for the client)
- Fixed random for the Server: precomputation
Summary

- Provable Security
- Authenticated Key Exchange
  - Security Model
  - Examples
  - Authentication
  - Password-based
- Group Key Exchange
  - Security Model
  - Example
  - Dynamic groups

Password-based Authentication

Password (short – low-entropy secret – say 20 bits)
- exhaustive search is possible
- basic attack: on-line exhaustive search
  - the adversary guesses a password
  - tries to play the protocol with this guess
  - failure ⇒ it erases the password from the list
  - and restarts…
- after 1,000,000 attempts, the adversary wins

cannot be avoided
Dictionary Attack

- The on-line exhaustive search
  - cannot be prevented
  - can be made less serious (delay, limitations, ...)
- We want it to be the best attack...
- The off-line exhaustive search
  - a few passive or active attacks
  - failure ⇒ erasure of MANY passwords from the list
  - this is called dictionary attack

Security

One wants to prevent dictionary attacks:
- a passive trial (execute + reveal)
  - does not reveal any information about the password
- an active trial (send)
  - allows to erase at most one password from the list of possible passwords
  - (or maybe 2 or 3 for technical reasons in the proof)
Example: EKE

The most famous scheme EKE: Encrypted Key Exchange

- Flows are encrypted with the password.
- Must be done carefully: no redundancy
- From $X'$, for any password $\pi$
  - decrypt $X'$
  - check whether it begins with “Alice”

\[
\begin{align*}
&x \in \mathbb{Z}_q^*, X = g^x \\
&Y = D_x(Y') \quad K = Y^x
\end{align*}
\]

\[
\begin{align*}
&x \in \mathbb{Z}_q^*, X = g^x \\
&X' = E_x(Alice, X) \\
&y \in \mathbb{Z}_q^*, Y = g^y \\
&y \leftarrow D_x(Y') \\
&k_i = H_i(Alice, Bob, K)
\end{align*}
\]

EKE - AuthA

AuthA

Bellare-Rogaway 2000
One-flow Encrypted Key Exchange

- EKE: security claimed, but never fully proved
- AuthA: security = open problem
Security Results

- Assumptions
  - the ideal-cipher model – for (E,D)
  - the random-oracle model – for H and H₁
- Semantic security of AuthA:
  - Advantage \( \geq 3 \frac{q_{\text{send}}}{\sqrt{N} + \varepsilon} \),
  - \( \Rightarrow \) CDH problem: probability \( \geq \varepsilon / 8q_{\text{hash}} \)
    (within almost the same time)
- Similar (but less efficient) results for EKE

New Security Results

- Assumptions
  - the random-oracle model
- Symmetric encryption = one-time pad:
  - \( E_{\pi}(X) = X \times G(\pi) \)
- Semantic security of AuthA:
  - Advantage \( \geq 12 \frac{q_{\text{send}}}{\sqrt{N} + \varepsilon} \),
  - \( \Rightarrow \) CDH problem: probability \( \geq \varepsilon / 12q_{\text{hash}}^2 \)
- Similar (but less efficient) results for EKE
Summary

- Provable Security
- Authenticated Key Exchange
  - Security Model
  - Examples
  - Authentication
  - Password-based
- Group Key Exchange
  - Security Model
  - Example
  - Dynamic groups

Model of Communication

- A set of $n$ players, modelled by oracles
- A multicast group consisting of a set of players
Modelling the Adversary

- **send**: send messages to instances
- **execute**: obtain honest executions of the protocol
- **reveal**: obtain an instance’s session key
- **corrupt**: obtain the value of the authentication secret

Summary

- Provable Security
- **Authenticated Key Exchange**
  - Security Model
  - Examples
  - Authentication
  - Password-based
- **Group Key Exchange**
  - Security Model
  - Example
  - Dynamic groups
A Group Key Exchange

- Generalization of the 2-party DH, the session key is $sk = H(g^{x_1 x_2 \cdots x_n})$
- Ring-based algorithm
  - up-flow: the contributions of each instance are gathered
  - down-flow: the last instance broadcasts the result
  - end: instances compute the session key

\[
\begin{align*}
  x_1 & \rightarrow g \\
  x_2 & \rightarrow g^{x_1} \\
  x_3 & \rightarrow g^{x_2 x_3} \\
  x_4 & \rightarrow g^{x_2 x_3 x_4}
\end{align*}
\]

The Algorithm

- Up-flow: $U_i$ raises received values to the power $x_i$
- Down-flow: $U_n$ broadcasts (except $g^{x_1 x_2 \cdots x_n}$)

Everything is authenticated (Signature/MAC)

\[
\begin{align*}
  x_1 & \rightarrow [g, g^{x_1}] \\
  x_2 & \rightarrow [g^{x_2}, g^{x_1}, g^{x_1 x_2}] \\
  x_3 & \rightarrow [g^{x_2 x_3}, g^{x_1 x_3}] \\
  \text{sk} & = H(g^{x_1 x_2 x_3})
\end{align*}
\]
Group CDH

- The CDH generalized to the multi-party case
  - given the values $g^{\prod x_i}$ for some choice of proper subset of \{1, ..., n\}
  - one has to compute the value $g^{x_1 x_2 ... x_n}$

- Example ($n=3$ and $I=\{1,2,3\}$)
  - given the set of the blue values
    $g, g^{x_1}, g^{x_2}, g^{x_1 x_2}$
  - compute the red value
    $g^{x_1 x_3}, g^{x_2 x_3}, g^{x_1 x_2 x_3}$

GCDH $\geq$ DDH or CDH

[BCP - SAC ‘02]

Security Result

- Theorem (in the random-oracle model)
  [BCPQ – ACM CCS ‘01]

\[
\text{Adv}^{ake} \leq 2q_{send}^n q_{hash} \cdot \text{Succ}^{gcdh}(n,T) \\
+ 2n \cdot \text{Succ}^{sign}(q_s,T)
\]

- Idea:
  - we introduce a Group Diffie-Hellman instance in the tested session
  - we have to guess in which send-queries: factor $q_{send}^n$
  - When the adversary has broken the scheme, the Group Diffie-Hellman solution is in the list of the queries to H
  - we have to guess it: factor $q_{hash}$
Improvements

- Security result: exponential in $n$
- Improvements
  - No guess of the tested pool
  - Use of the random self-reducibility of the DH problems
    $\Rightarrow$ reduction linear in $n$
  - Standard model (MAC and Left-Over-Hash Lemma)
- Dynamic groups
  - If one party leaves or joins the group, the protocol does not need to be restarted from scratch

Summary

- Provable Security
- Authenticated Key Exchange
  - Security Model
  - Examples
  - Authentication
  - Password-based
- Group Key Exchange
  - Security Model
  - Example
  - Dynamic groups
Dynamic Groups

- **Join**: the last broadcast is sent to the new player and becomes the last up-flow \( \Rightarrow \) the new player introduces a new random

- **Remove**: the last remaining player introduces a new random \( x'_i \) in place of his \( x_i \) and broadcasts the useful values only

\[
\text{Remove 2 and 4 } \quad g^{x_i 2^i x_i 4} \quad g^{x_i 1^i x_i 3 x_i 4} \quad g^{x_i 1^i x_i 2 x_i 4} \quad g^{x_i 1^i x_i 2 x_i 3} \quad g^{x_i 1^i x_i 2 x_i 3 x_i 4} \quad g^{x_i 1^i x_i 2 x_i 3 x_i 4}
\]

Dynamic Groups: Security Result

- Group of \( n \) people
- Tested group of size \( s \)
- Number of operations (setup, join, remove): \( O \)
- Time: \( T \)

\[
\text{Adv}^{\text{ake}} \leq 2 \cdot Q \cdot C_n^s \cdot q_{\text{hash}} \cdot \text{Succ}^{\text{gcdh}}(s, T) + 2n \cdot \text{Succ}^{\text{sign}}(q_{\text{send}}, T)
\]

- **Idea:**
  - Guess the players in the tested group
  - Guess the last operation before the tested key
  - Guess the solution among the \( H \) queries
Improved Security Result

- Number of people involved in the group before the test-query (maybe removed) = s
- Number of operations (setup, join, remove): Q
- Time: T

\[ \text{Adv}^{\text{ake}} \leq 2nQ \cdot \text{Adv}^{\text{gddh}}(s,T) + 2n \cdot \text{Succ}^{\text{sign}}(q_{\text{send}}, T) \]

Idea:
- Guess the last operation before the tested key
- Guess of the index of the player which makes the last down-flow

Details

- Given instance:
  \[ g^{x_2} \quad g^{x_1} \quad g^{x_2 x_3} \quad g^{x_1 x_3} \quad g^{x_1 x_2} \quad g^{x_2 x_3 x_4} \quad g^{x_1 x_3 x_4} \quad g^{x_1 x_2 x_4} \quad g^{x_1 x_2 x_3} \quad g^{x_1 x_2 x_3 x_4} \]

- Use a new line for a new player, up to the s-1st
  - For additional players: known random
    \[ \Rightarrow \] known keys (reveal-queries)
  - Use the last line for the tested group, introducing \( x_4 \) at the \( Q^{th} \) operation
    \[ \Rightarrow \] test-query answered by the red value
  - After: back to s-1st line, but not necessarily removing \( x_4 \)
Details (Con'd)

- Extended instance:
  \[
  \begin{array}{ccccccc}
  g^{x_2} & g^{x_1} \\
  g^{x_2 x_3} & g^{x_1 x_3} & g^{x_1 x_2} \quad g^{x_1 x_4} & g^{x_2 x_4} \quad g^{x_3 x_4} \\
  g^{x_2 x_3 x_4} & g^{x_1 x_3 x_4} & g^{x_1 x_2 x_4} & g^{x_1 x_2 x_3} & g^{x_1 x_2 x_3 x_4}
  \end{array}
  \]

- In the \(s-1\)st line: all the combinations of \(s-2\) exponents
  - We remain on this line
  - We know the session key (in the \(s^{th}\) line)

---

Password-Based

[BCP – Eurocrypt '02]

- Generalization of the 2-party PAKE DH
- Encrypt each flow with password (in ICM)
  - Redundancy: dictionary attack
    \[ \Rightarrow \text{Randomization: } sk = H(g^{a_1 a_2 \ldots a_n x_1 x_2 \ldots x_n}) \]

\[
\begin{array}{ccccccc}
  a_1, x_1 & g & g^{a_1} & g^{a_1 x_1} \\
  a_2, x_2 & g^{a_1 a_2 x_2} & g^{a_1 a_2 x_1} & g^{a_1 a_2 x_1 x_2} \\
  a_3, x_3 & g^{a_1 a_2 a_3 x_3 x_4} & g^{a_1 a_2 a_3 x_1 x_3 x_4} & g^{a_1 a_2 a_3 x_1 x_2 x_3 x_4} \\
  a_4, x_4 & g^{a_1 a_2 a_3 a_4 x_3 x_4} & g^{a_1 a_2 a_3 a_4 x_1 x_3 x_4} & g^{a_1 a_2 a_3 a_4 x_1 x_2 x_3 x_4} & g^{a_1 a_2 a_3 a_4 x_2 x_3 x_4}
  \end{array}
\]