**Password-based Authenticated Key Exchange State of the Art**

David Pointcheval  
*CNRS-ENS – France*  

LBNL  
*Berkeley – California - USA*  

*August 2004*

---

**Summary**

- Authenticated Key Exchange
- Password-based Authentication
- Encrypted Key Exchange
- Open Key Exchange
- Implementation Concerns
- In Practice

---

**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 $RS$, to anybody else
### Additional 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 (leaked to the adversary), previously shared secrets are still semantically secure

### 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**: history built using
  - **execute**-queries $\rightarrow$ transcripts
  - **reveal**-queries $\rightarrow$ session keys

- **Active**: entire control of the network
  - **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

### Diffie-Hellman Key Exchange

- The most famous key exchange protocol: **Diffie-Hellman**

  ![Diffie-Hellman Diagram]

  It is **not authenticated**: anybody can say “I am Alice” or “I am Bob”

  $\Rightarrow$ semantic security against **passive adversaries**
Authentication

To prevent active attacks (*manufactured send*), some kind of authentication is required:

- **Asymmetric**: \((sk_A, pk_A)\) and possibly \((sk_B, pk_B)\)
  
  parties authenticate to each other using
  
  the knowledge of the private key associated
  
  to the certified public key

- **Symmetric**: common (high-entropy) secret
  
  they use the long term secret to derive
  
  a secure and authenticated ephemeral key \(SK\)

- **Password**: common (low-entropy) secret
  
  *e.g.* a 20-bit password

---

Password-based Authentication

Password (low-entropy secret) *e.g.* 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

---

Summary

- Authenticated Key Exchange
  
  - Password-based Authentication
  
  - Encrypted Key Exchange
  
  - Open Key Exchange
  
  - Implementation Concerns
  
  - In Practice

---

Dictionary Attack

- **On-line exhaustive search**
  
  - cannot be avoided
  
  - can be made less serious (delay, limitations, ...)

  We want it to be the *best attack*...

- **Off-line exhaustive search**
  
  - a few passive or active attacks
  
  - failure ⇒ erasure of *MANY* passwords from the list

  this is called *dictionary attack*
One wants to prevent dictionary attacks:
- any passive trial (execute + reveal)
  - no useful information about the password
- one active trial (send)
  - cancels at most one password from the list of possible passwords

---

**EKE = Encrypted Key Exchange**

EKE = Diffie-Hellman, with encrypted flows
- Must be done carefully...
- From $X'$, for any password $\pi$
  - decrypt $X'$
  - check whether it begins with “Alice”

avoid any redundancy

---

**Summary**

- Authenticated Key Exchange
- Password-based Authentication
- Encrypted Key Exchange
- Open Key Exchange
- Implementation Concerns
- In Practice

---

**EKE = Encrypted Key Exchange**

The correct scheme: without redundancy

Bad one:

<table>
<thead>
<tr>
<th>Alice</th>
<th>Password $\pi$</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x, X = g^x$</td>
<td>$X' = E_x(Alice, X)$</td>
<td></td>
</tr>
<tr>
<td>$y = D_y(Y')$</td>
<td>$Y' = E_y(Bob, Y)$</td>
<td></td>
</tr>
<tr>
<td>$K = y' = g^{x K}$</td>
<td>$SK = H(Alice, Bob, X, Y, K)$</td>
<td></td>
</tr>
</tbody>
</table>

Good one:

<table>
<thead>
<tr>
<th>Alice</th>
<th>Password $\pi$</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x, X = g^x$</td>
<td>$Alice, X' = E_x(X)$</td>
<td></td>
</tr>
<tr>
<td>$y = D_y(Y')$</td>
<td>$Bob, Y' = E_y(Y)$</td>
<td></td>
</tr>
<tr>
<td>$K = y' = g^{x K}$</td>
<td>$SK = H(Alice, Bob, X, Y, K)$</td>
<td></td>
</tr>
</tbody>
</table>

$E_{\pi}$ must be a bijection from the group $<g>$
**SPEKE = Simple Password Exponential Key Exchange**

**Variant of DH-EKE:** Jablon 1996

<table>
<thead>
<tr>
<th>Alice</th>
<th>Password ( p )</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x, X = g^x )</td>
<td>Alice, ( X )</td>
<td>( y, Y = g^y )</td>
</tr>
<tr>
<td>( K = Y^x )</td>
<td>( K = X^y )</td>
<td></td>
</tr>
</tbody>
</table>

- According to the function \( f \), this scheme can be either secure or totally insecure
  - If \( f \) is a random function (random oracle) onto the whole group \( <g> \): provably secure [MacKenzie 2001]

**PPK – AuthA - MDHKE**

Bresson-Chevassut-P. 2003/2004

A simple variant: **one-time pad**

\[
E_p(X) = X H(\pi)
\]

<table>
<thead>
<tr>
<th>Alice</th>
<th>Password ( p )</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x, X = g^x )</td>
<td>Alice, ( X' = X H(\pi) )</td>
<td>( X = X'/H(\pi) )</td>
</tr>
<tr>
<td>( y, Y = g^y )</td>
<td>( K = Y^x )</td>
<td>( K = X^y )</td>
</tr>
</tbody>
</table>

- If \( H \) is a random function (random oracle) onto the whole group \( <g> \): provably secure

**Simple Encrypted Key Exchange**

Abdalla-P. 2004

The simplest variant: \( E_p(X) = X U^\pi \)

<table>
<thead>
<tr>
<th>Alice</th>
<th>Password ( p )</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x, X = g^x )</td>
<td>Alice, ( X' = X U^\pi )</td>
<td>( x = X'/U^\pi )</td>
</tr>
<tr>
<td>( y, Y = g^y )</td>
<td>Bob, ( Y' = Y V^\pi )</td>
<td>( y = g^y )</td>
</tr>
</tbody>
</table>

- No random function onto groups
- Just two fixed elements \( U, V \) in \( <g> \)
- Non-concurrent executions: provably secure

**Generalized Encrypted Key Exchange**

More generally

- Alice generates a public key \( pk \), sends \( pk \) encrypted with the password \( \rightarrow pk' \)
- Bob recovers \( pk \), generates a random \( r \), encrypts it with \( pk \rightarrow c \)
  sends \( c \) encrypted with the password \( \rightarrow c' \)
- Alice recovers the common random \( r \)
- The session key \( SK \) is derived from this \( r \)
**Generalized Encrypted Key Exchange**

**Problems:**
- $pk$ must be truly random (no redundancy)
- $pk$ and $r$ are not on the same space, in general
- Nice exception: ElGamal (DH-EKE)
  - requires $E_\pi$ to be over $\langle g \rangle$
  - impossible to be used with RSA...

---

**Open Key Exchange**

Lucks 1997

The public key $pk$ is sent in **clear**:

<table>
<thead>
<tr>
<th>Alice</th>
<th>Password $\pi$</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>$sk, pk$</td>
<td>$Alice, pk'$ $\Rightarrow$ $Alice, pk'$ $\Rightarrow$ $E_{pk}(pk')$</td>
<td></td>
</tr>
<tr>
<td>$c = D_k(c') \Rightarrow r = D_k(c)$</td>
<td>$Bob, c' = E_k(c)$</td>
<td>$pk = D_k(pk')$</td>
</tr>
<tr>
<td>$r = D_k(c)$</td>
<td>$r, c = E_{sk}(r)$</td>
<td></td>
</tr>
<tr>
<td>$SK = H(Alice, Bob, pk', c', r)$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Requirements to avoid partition attacks:
- $E_\pi$ must be a bijection from the ciphertext space
- $E_{pk}$ must be a surjection onto this space

---

**Summary**

- Authenticated Key Exchange
- Password-based Authentication
- Encrypted Key Exchange
- **Open Key Exchange**
- Implementation Concerns
- In Practice

---

**Surjection**

Since $pk$ can be chosen by the adversary, one must check "$E_{pk}$ is a surjection"

- If not, given $c'$, one eliminates the $\pi$'s, that lead to $c'$'s which are not in the image set of $E_{pk}$: **partition attack**
- If so, given $c'$, any $\pi$ is possible: sending the correct $\pi$ means guessing the good $\pi$
  - zero-knowledge proof
    - concurrent
    - non-malleable
**Protected OKE**

- Specific to RSA
- Additional proof of valid modulus
- Flaw in the original scheme
- Repaired in SNAPI
  - But not very efficient:
    - very large exponents \( e > n \)
- Efficient variant with RSA-IPAKE
  - [Catalano-P. -Pornin - 2004]

---

**IPAKE = Generalized OKE**

- Using any isomorphism and one-time pad

### Isomorphism for Password-based Authenticated Key Exchange

<table>
<thead>
<tr>
<th>Alice</th>
<th>Password π</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>sk, pk</td>
<td>Alice, pk</td>
<td>Bob, c' = c ( \oplus G(π) )</td>
</tr>
<tr>
<td>c = c' ( \oplus G(π) ), r = g_s(r)</td>
<td>k = H(Alice, Bob, r)</td>
<td></td>
</tr>
<tr>
<td>k = ( H(Alice, Bob, pk, c', π, r) )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- \( f_{pk} \) isomorphism from \( F_{pk} \) onto the group \( (G_{pk}, \oplus) \)
- Must be trapdoor “hard-to-invert”

---

**IPAKE = Applications**

- ElGamal encryption (Diffie-Hellman function)
  - PAK
  - [Boyko-MacKenzie-Patel 2000]
  - AuthA
  - [Bellare-Rogaway 2000]
  - OMDHKE
  - [Bresson-Chevassut-P. 2003]
- RSA function
  - Protected OKE
  - [Lucks 1997]
  - SNAPI
  - [MacKenzie-Patel-Swaminathan 2000]
- Modular square
  - The first Password-based AKE related to integer factoring

---

**Summary**

- Authenticated Key Exchange
- Password-based Authentication
- Encrypted Key Exchange
- Open Key Exchange

---

**Implementation Concerns**

- In Practice
Several Candidates

Many proposals:
- In the **standard model**: KOY protocol (DDH) [Katz-Ostrovy-Yung 2001]
  - But quite inefficient
- In the **random oracle model**:
  - EKE (CDH), OKE (CDH/RSA), SPEKE (CDH)
  - IPAKE (CDH/RSA/Fact), simple EKE (CDH)
  - Which seem very efficient...

Full-domain Functions

Most of these schemes require full-domain random functions/bijections:
- Hash function onto \(<g>\)
- Block cipher over \(<g>\)
- How to implement them?
  - Take a hash function / block cipher onto \(\{0,1\}^k\), where \(k\) is the length of any encoding of elements in \(<g>\)
  - Iterate it until falling in \(<g>\)

Implementation Details

Requirement:
- \(2^k/\text{Card}(<g>)\), must not be too large
  - Average number of iterations
  - \(g\) of (almost) maximal order
- \(g\) of (almost) maximal order
- Use of large exponents... or **elliptic curves**
- In \(\mathbb{Z}_p^*\), \(|p| = 1024 \Rightarrow \) exponents over \(>1000\) bits
  - Small subgroups are possible, but at the same high cost (large co-factor)
  - On EC, 160 bit field \(\Rightarrow\) exponents over 160 bits
  - Just one iteration on average

Timing Attacks

The number of iterations may depend on the password:

On basic EKE: responding time
- number of iterations for \((X, X')\) and \((Y, Y')\)
  - Each passive attack divides the set by 4
  - \(\Rightarrow\) partition attack!
One-Time Pad

The use of the one-time pad limits the damages:

- No pre-computation: responding time = number of iterations for $H(\pi)$
- But always the same information
- Pre-computation of $H(\pi)$: no information leaked

Simple EKE

- Particular case: the “simple EKE”
  - No full-domain function: easy to implement
  - Apply to any prime sub-group: quite efficient
  - But: restricted to non-concurrent executions

Summary

- Authenticated Key Exchange
- Password-based Authentication
- Encrypted Key Exchange
- Open Key Exchange
- Implementation Concerns

In Practice

Additional Properties

- Mutual authentication:
  - General construction (with key confirmation) [Bellare-P.-Rogaway 2000]

- Forward secrecy:
  - EKE/OKE provide it [Abdalla-Chevassut-P. 2004]
Hostile Environments

- The client machine may not be fully trusted:
  - When the user types his password: leaked...
  - ⇒ for some schemes, it is possible to use any kind of ephemeral secret shared between the user and the server (OTP, SecurID)

Work in progress...

Vulnerable Server

- The server machine may not be fully trusted:
  - The machine may be vulnerable, in case of corruption, all the passwords are leaked...
  - ⇒ verifier-based variants exist (the server just owns a verifier –an image of the password through a one-way function)
    - In case of corruption, a dictionary attack is necessary
    - By adding salts, it is made less effective
  - ⇒ the password can be distributed among several servers (threshold AKE)

Work in progress...