Automatically Verified Mechanized Proof of One-Encryption Key Exchange

Bruno Blanchet
blanchet@di.ens.fr

INRIA, École Normale Supérieure, CNRS, Paris

January 2012
Motivation

- OEKE (One-Encryption Key Exchange) [Bresson, Chevassut, Pointcheval, CCS’03]:
  - Variant of EKE (Encrypted Key Exchange)
  - A password-based key exchange protocol.
  - A non-trivial protocol.
  - It took some time before getting a computational proof of this protocol.
- Our goal:
  - Mechanize, and automate as far as possible, its proof using the automatic computational protocol verifier CryptoVerif.
  - This is an opportunity for several interesting extensions of CryptoVerif.
# Proofs by sequences of games

Proofs in the computational model are typically proofs by sequences of games [Shoup, Bellare&Rogaway]:

- The first game is the real protocol.
- One goes from one game to the next by syntactic transformations or by applying the definition of security of a cryptographic primitive. The difference of probability between consecutive games is negligible.
- The last game is "ideal": the security property is obvious from the form of the game.
  (The advantage of the adversary is 0 for this game.)

<table>
<thead>
<tr>
<th>Game 0</th>
<th>➞</th>
<th>Game 1</th>
<th>➞</th>
<th>…</th>
<th>➞</th>
<th>Game n</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protocol to prove</td>
<td>$p_1$ negligible</td>
<td>$p_2$ negligible</td>
<td>…</td>
<td>$p_n$ negligible</td>
<td>Property obvious</td>
<td></td>
</tr>
</tbody>
</table>
CryptoVerif background: Indistinguishability

- The game $G$ interacting with an adversary (evaluation context) $C$ is denoted $C[G]$.
- $C[G]$ may execute events, collected in a sequence $\mathcal{E}$.
- A distinguisher $D$ takes as input $\mathcal{E}$ and returns true or false.
  - Example: $D_e(\mathcal{E}) = \text{true}$ if and only if $e \in \mathcal{E}$. $D_e$ is abbreviated $e$.
- $\Pr[C[G] : D]$ is the probability that $C[G]$ executes $\mathcal{E}$ such that $D(\mathcal{E}) = \text{true}$.

Definition (Indistinguishability)

We write $G \approx^V_p G'$ when, for all evaluation contexts $C$ acceptable for $G$ and $G'$ with public variables $V$ and all distinguishers $D$,

Properties of indistinguishability

Lemma

1. Reflexivity: $G \approx^V_0 G$.
2. Symmetry: $\approx^V_p$ is symmetric.
3. Transitivity: if $G \approx^V_p G'$ and $G' \approx^V_{p'} G''$, then $G \approx^V_{p+p'} G''$.
4. Application of context: if $G \approx^V_p G'$ and $C$ is an evaluation context acceptable for $G$ and $G'$ with public variables $V$, then $C[G] \approx^V_{p'} C[G']$, where $p'(C', D) = p(C'[C[]], D)$ and $V' \subseteq V \cup \text{var}(C)$.
Introduction

Assumptions

On Shoup’s lemma

The proof

Conclusion

OEKE

Client $U$ | Server $S$

| shared $pw$ |

$x \xleftarrow{R} [1, q - 1]$  
$X \leftarrow g^x$  

$y \xleftarrow{R} [1, q - 1]$  
$Y \leftarrow g^y$

$Y \leftarrow D_{pw}(Y^*)$  

$S, Y^* \xrightarrow{S} Y^* \leftarrow E_{pw}(Y)$

$K_U \leftarrow Y^x$

$Auth \leftarrow H_1(U || S || X || Y || K_U)$

$sk_U \leftarrow H_0(U || S || X || Y || K_U)$

$Auth \xrightarrow{Auth} K_S \leftarrow X^y$

if $Auth = H_1(U || S || X || Y || K_S)$ then

$sk_S \leftarrow H_0(U || S || X || Y || K_S)$
The proof relies on the **Computational Diffie-Hellman** assumption and on the **Ideal Cipher Model**.

- ⇒ Model these assumptions in CryptoVerif.

The proof uses **Shoup’s lemma**:

- Insert an event and later prove that the probability of this event is negligible.
- ⇒ Implement this reasoning technique in CryptoVerif.

The **probability of success of an attack must be precisely evaluated** as a function of the size of the password space.

- ⇒ Optimize the computation of probabilities in CryptoVerif.
Computational Diffie-Hellman assumption

Consider a multiplicative cyclic group $G$ of order $q$, with generator $g$. A probabilistic polynomial-time adversary has a negligible probability of computing $g^{ab}$ from $g$, $g^a$, $g^b$, for random $a, b \in \mathbb{Z}_q$. 
Computational Diffie-Hellman assumption in CryptoVerif

Consider a multiplicative cyclic group $G$ of order $q$, with generator $g$. A probabilistic polynomial-time adversary has a negligible probability of computing $g^{ab}$ from $g$, $g^a$, $g^b$, for random $a, b \in \mathbb{Z}_q$.

In CryptoVerif, this can be written

\[
\begin{align*}
&\forall i \leq N \text{ new } a : Z; \text{ new } b : Z; (OA() := \exp(g, a), OB() := \exp(g, b), \\
&\quad \forall i' \leq N' \text{ OCDH}(z : G) := z = \exp(g, \text{mult}(a, b))) \\
&\approx \\
&\forall i \leq N \text{ new } a : Z; \text{ new } b : Z; (OA() := \exp(g, a), OB() := \exp(g, b), \\
&\quad \forall i' \leq N' \text{ OCDH}(z : G) := false)
\end{align*}
\]
Computational Diffie-Hellman assumption in CryptoVerif

Consider a multiplicative cyclic group $G$ of order $q$, with generator $g$. A probabilistic polynomial-time adversary has a negligible probability of computing $g^{ab}$ from $g$, $g^a$, $g^b$, for random $a, b \in \mathbb{Z}_q$.

In CryptoVerif, this can be written

$$\begin{align*}
!^{i \leq N} \text{new } a : Z; \text{new } b : Z; (OA() := \exp(g, a), OB() := \exp(g, b), \\
!^{i' \leq N'} \text{OCDH}(z : G) := z = \exp(g, \text{mult}(a, b)))
\end{align*}$$

$$\approx$$

$$\begin{align*}
!^{i \leq N} \text{new } a : Z; \text{new } b : Z; (OA() := \exp(g, a), OB() := \exp(g, b), \\
!^{i' \leq N'} \text{OCDH}(z : G) := \text{false})
\end{align*}$$

Application: semantic security of hashed El Gamal in the random oracle model (A. Chaudhuri).
Computational Diffie-Hellman assumption in CryptoVerif

This model is not sufficient for OEKE and other practical protocols.

- It assumes that $a$ and $b$ are chosen under the same replication.
- In practice, one participant chooses $a$, another chooses $b$, so these choices are made under different replications.
Extending the formalization of CDH in CryptoVerif

\[\begin{align*}
\text{\texttt{ia}} \leq \text{na} & \quad \textbf{new} \quad \texttt{a} : Z; (\text{OA}() := \exp(g, a), \text{Oa}() := a, \\
\text{\texttt{iaCDH}} \leq \text{naCDH} & \quad \texttt{OCDHa}\left(m : G, j \leq nb\right) := m = \exp(g, \text{mult}(b[j], a))) , \\
\text{\texttt{ib}} \leq \text{nb} & \quad \textbf{new} \quad \texttt{b} : Z; (\text{OB}() := \exp(g, b), \text{Ob}() := b, \\
\text{\texttt{ibCDH}} \leq \text{nbCDH} & \quad \texttt{OCDHb}\left(m : G, j \leq na\right) := m = \exp(g, \text{mult}(a[j], b)))
\end{align*}\]
Extending the formalization of CDH in CryptoVerif

\[ \begin{align*}
! \text{ia} \leq Na \quad & \text{new } a : Z; (OA()) := \exp(g, a), Oa() := a, \\
! \text{iaCDH} \leq naCDH \quad & \text{OCDHa}(m : G, j \leq Nb) := m = \exp(g, \text{mult}(b[j], a))), \\
! \text{ib} \leq Nb \quad & \text{new } b : Z; (OB()) := \exp(g, b), Ob() := b, \\
! \text{ibCDH} \leq nbCDH \quad & \text{OCDHb}(m : G, j \leq Na) := m = \exp(g, \text{mult}(a[j], b))). \\
\end{align*} \]

\[ \begin{align*}
\approx \\
! \text{ia} \leq Na \quad & \text{new } a : Z; (OA()) := \exp(g, a), Oa() := \text{let } ka = \text{mark in } a, \\
! \text{iaCDH} \leq naCDH \quad & \text{OCDHa}(m : G, j \leq Nb) := \\
& \text{find } u \leq nb \text{ suchthat defined}(kb[u], b[u]) \land b[j] = b[u] \text{ then} \\
& \quad m = \exp(g, \text{mult}(b[j], a)) \\
& \text{else if defined}(ka) \text{ then } m = \exp(g, \text{mult}(b[j], a)) \text{ else false),} \\
! \text{ib} \leq Nb \quad & \text{new } b : Z; (OB()) := \exp(g, b), Ob() := \text{let } kb = \text{mark in } b, \\
! \text{ibCDH} \leq nbCDH \quad & \text{OCDHb}(m : G, j \leq Na) := (\text{symmetric of OCDHa}))
\]
Extending the formalization of CDH in CryptoVerif

\[ !^{ia \leq Na} \textbf{new} \ a : Z; (OA()) := \text{exp}(g, a), Oa()[3] := a, \]
\[ !^{iaCDH \leq naCDH} OCDHa(m : G, j \leq Nb)[\text{useful\_change}] := m = \text{exp}(g, \text{mult}(a[j], b)) \]
\[ !^{ib \leq Nb} \textbf{new} \ b : Z; (OB()) := \text{exp}(g, b), Ob()[3] := b, \]
\[ !^{ibCDH \leq nbCDH} OCDHb(m : G, j \leq Na) := m = \text{exp}(g, \text{mult}(a[j], b)) \]
\[ \approx (\#OCDHa + \#OCDHb) \times \max(1, e^2 \#Oa) \times \max(1, e^2 \#Ob) \times \text{pCDH}(\text{time} + (na + nb + \#OCDHa + \#OCDHb) \times \text{time}(\text{exp})) \]
\[ !^{ia \leq Na} \textbf{new} \ a : Z; (OA()) := \text{exp}'(g, a), Oa() := \text{let} \ ka = \text{mark in} \ a, \]
\[ !^{iaCDH \leq naCDH} OCDHa(m : G, j \leq Nb) := \]
\[ \text{find} \ u \leq nb \text{ such that defined}(kb[u], b[u]) \land b[j] = b[u] \text{ then} \]
\[ m = \text{exp}(g, \text{mult}(b[j], a)) \]
\[ \text{else if defined}(ka) \text{ then} \ m = \text{exp}'(g, \text{mult}(b[j], a)) \text{ else} \ false \), \]
\[ !^{ib \leq Nb} \textbf{new} \ b : Z; (OB()) := \text{exp}'(g, b), Ob() := \text{let} \ kb = \text{mark in} \ b, \]
\[ !^{ibCDH \leq nbCDH} OCDHb(m : G, j \leq Na) := (\text{symmetric of} \ OCDHa) \]
Other declarations for Diffie-Hellman (1)

\[ g : G \]
\[ \exp(G, Z) : G \]
\[ \text{mult}(Z, Z) : Z \text{ commutative} \]
\[ \exp(\exp(z, a), b) = \exp(z, \text{mult}(a, b)) \]
\[ (g^a)^b = g^{ab} \text{ and } (g^b)^a = g^{ba}, \text{ equal by commutativity of } \text{mult} \]

\[ \exp(g, x) = \exp(g, y) \implies (x = y) \]
\[ \exp'(g, x) = \exp'(g, y) \implies (x = y) \]

Injectivity

\[ \text{new } x_1 : Z; \text{new } x_2 : Z; \text{new } x_3 : Z; \text{new } x_4 : Z; \]
\[ \text{mult}(x_1, x_2) = \text{mult}(x_3, x_4) \approx_{1/|Z|} \text{false} \]
\[ (\text{mult}(x, y) = \text{mult}(x, y')) \implies (y = y') \]

Collision between products
Other declarations for Diffie-Hellman (2)

\[ \begin{align*}
! & \leq N \textbf{new} \ X : G; \ OX() := X \\
\approx_0 \ [\text{manual}] \ & ! \leq N \textbf{new} \ x : Z; \ OX() := \exp(g, x)
\end{align*} \]

This equivalence is very general, apply it only manually.

\[ \begin{align*}
! & \leq N \textbf{new} \ X : G; \ (OX()) := X, \ ! \leq N' \textbf{new} \ OXm(m : Z) [\text{useful\_change}] := \exp(X, m) \\
\approx_0 \ & \ 
\end{align*} \]

\[ \begin{align*}
! & \leq N \textbf{new} \ x : Z; \ (OX()) := \exp(g, x), \ ! \leq N' \textbf{new} \ OXm(m : Z) := \exp(g, \text{mult}(x, m))
\end{align*} \]

This equivalence is a particular case applied only when \( X \) is inside \( \exp \), and good for automatic proofs.

\[ \begin{align*}
! & \leq N \textbf{new} \ x : Z; \ OX() := \exp(g, x) \\
\approx_0 \ & ! \leq N \textbf{new} \ X : G; \ OX() := X
\end{align*} \]

And the same for \( \exp' \).
Extensions for CDH

The implementation of the support for CDH required two extensions of CryptoVerif:

- An array index \( j \) occurs as argument of a function.
  - extend the language of equivalences used for specifying assumptions on primitives.
- The equality test \( m = \exp(g, \text{mult}(b, a)) \) typically occurs inside the condition of a \texttt{find}.
  - This \texttt{find} comes from the transformation of a hash function in the Random Oracle Model.

After transformation, we obtain a \texttt{find} inside the condition of a \texttt{find}. 
The Ideal Cipher Model

- For all keys, encryption and decryption are two inverse random permutations, independent of the key.
  - Some similarity with SPRP ciphers but, for the ideal cipher model, the key need not be random and secret.
- In CryptoVerif, we replace encryption and decryption with lookups in the previous computations of encryption/decryption:
  - If we find a matching previous encryption/decryption, we return the previous result.
  - Otherwise, we return a fresh random number.
  - We eliminate collisions between these random numbers to obtain permutations.
- **No extension** of CryptoVerif is needed to represent the Ideal Cipher Model.
Shoup’s lemma

Goal: bound $\Pr[C[G_0] : e_0]$. 

- $G_0$ 
  - probability $p$

- $G_n$ 
  - $\Pr[C[G_{n+1}] : e]$

- $G_{n+1}$ 
  - event $e$
  - probability $p'$

- $G_{n'}$ 
  - events $e_0$ and $e$ never executed

\[
\Pr[C[G_0] : e_0] \leq p + \Pr[C[G_{n+1}] : e] + p' \\
\leq p + p' + p' \\
\leq p + 2p'
\]
Improved version of Shoup’s lemma

Goal: bound $\Pr[C[G_0]: e_0]$.

- $G_0$: probability $p$
- $G_n$: differ only when $e$ is executed
- $G_{n+1}$: event $e$
- $G_{n'}$: events $e_0$ and $e$ never executed

\[
\Pr[C[G_0]: e_0] \leq p + \Pr[C[G_n]: e_0] \\
\leq p + \Pr[C[G_{n+1}]: e_0 \lor e] \\
\leq p + p' + \Pr[C[G_{n'}]: e_0 \lor e] \\
\leq p + p'
\]
Improved Shoup’s lemma

Lemma

Let $C$ be a context acceptable for $G$ and $G'$ with public variables $V$.

1. **Improved Shoup’s lemma:**
   
   If $G'$ differs from $G$ only when $G'$ executes event $e$, then
   \[ \Pr[C[G] : D] \leq \Pr[C[G'] : D \lor e]. \]

2. If $G \approx^V_p G'$, then
   \[ \Pr[C[G] : D] \leq p(C, D) + \Pr[C[G'] : D]. \]

3. \[ \Pr[C[G] : D \lor D'] \leq \Pr[C[G] : D] + \Pr[C[G] : D']. \]
Definition (Secrecy)

Let $x$ be a one-dimensional array.

Let $R_x$ be a process that
- chooses a bit $b$;
- provides test queries that, on input $u$, return $x[u]$ when $b = 1$ and a random value $y[u]$ when $b = 0$;
- expects a value $b'$ from the adversary and executes event $S$ when $b' = b$.

Let $C$ be a context acceptable for $G | R_x$ without public variables that does not contain $S$.

\[
\text{Adv}^{\text{secrecy}(x)}_G(C) = 2 \Pr[C[G | R_x] : S] - 1
\]
Definition (Secrecy)

Let $x$ be a one-dimensional array.
Let $R_x$ be a process that

- chooses a bit $b$;
- provides test queries that, on input $u$, return $x[u]$ when $b = 1$ and a random value $y[u]$ when $b = 0$;
- expects a value $b'$ from the adversary and executes event $S$ when $b' = b$.

Let $C$ be a context acceptable for $G | R_x$ without public variables that does not contain $S$.

$$\text{Adv}^\text{secrecy}(x)(C) = 2 \Pr[C[G | R_x] : S] - 1$$
Proof of secrecy

Goal: secrecy of $x$ in $G_0$

$G_0 \mid R_x$

probability $p$

$G_n \mid R_x$

secrecy proved: $\Pr[C[G_n \mid R_x] : S] = \frac{1}{2}$

$$\text{Adv}_{G_0}^{\text{secrecy}(x)}(C) = 2 \Pr[C[G_0 \mid R_x] : S] - 1$$

$$\leq 2(p + \Pr[C[G_n \mid R_x] : S]) - 1$$

$$\leq 2p$$
Proof of secrecy with Shoup’s lemma

\[\begin{align*}
G_0 \mid R_x & \quad \text{goal: secrecy of } x \text{ in } G_0 \\
\uparrow & \quad \text{probability } p \\
G_n \mid R_x & \quad \text{differ only when } e \text{ is executed} \\
\uparrow & \quad \text{probability } p' \\
G_{n+1} \mid R_x & \quad \text{event } e \\
\uparrow & \quad \text{probability } p'' \\
G_{n'} \mid R_x & \quad \text{secrecy proved: } \Pr[C[G_n' \mid R_x] : S] = \frac{1}{2} \\
\uparrow & \quad \text{probability } p'' \\
G_{n''} \mid R_x & \quad \text{event } e \text{ never executed}
\end{align*}\]

\[\text{Adv}^{\text{secrecy}(x)}_{G_0}(C) \leq 2(p + \Pr[C[G_n \mid R_x] : S]) - 1 \leq 2(p + \Pr[C[G_{n+1} \mid R_x] : S \lor e]) - 1 \leq 2(p + p' + \Pr[C[G_{n'} \mid R_x] : S \lor e]) - 1 \leq 2(p + p' + \Pr[C[G_{n'} \mid R_x] : e]) \leq 2(p + p' + p'')\]

Bruno Blanchet (INRIA, ENS, CNRS)
**Improved proof of secrecy with Shoup’s lemma**

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>$G_0</td>
<td>R_x$</td>
<td>goal: secrecy of $x$ in $G_0$</td>
</tr>
<tr>
<td>$G_n</td>
<td>R_x$</td>
<td>differ only when $e$ is executed</td>
</tr>
<tr>
<td>$G_{n+1}</td>
<td>R_x$</td>
<td>event $e$</td>
</tr>
<tr>
<td>$G_{n'}</td>
<td>R_x$</td>
<td>secrecy proved: $\Pr[C[G_n'</td>
</tr>
<tr>
<td>$G_{n''}</td>
<td>R_x$</td>
<td>event $e$ is independent of $S$</td>
</tr>
<tr>
<td>$G_{n'''}</td>
<td>R_x$</td>
<td>event $e$ never executed</td>
</tr>
</tbody>
</table>

\[
\text{Adv}_{G_0}^{\text{secrecy}(x)}(C) \leq 2(p + p' + \Pr[C[G_n' | R_x] : S \lor e]) - 1
\leq 2(p + p' + \frac{1}{2} \Pr[C[G_n' | R_x] : e]) \leq 2(p + p') + p''
\]
Improved proof of secrecy with Shoup’s lemma

**Lemma**

If CryptoVerif proves the secrecy of $x$ in game $G$, and $e_1, \ldots, e_n$ are events introduced by Shoup’s lemma in previous steps of the proof, then

$$\Pr[C[G | R_x] : S \lor e_1 \lor \cdots \lor e_n] \leq \frac{1}{2} + \frac{1}{2} \Pr[C[G | R_x] : e_1 \lor \cdots \lor e_n].$$

Events $e_1, \ldots, e_n$ are independent of $S$.

$$\Pr[C[G] : S \lor e_1 \lor \cdots \lor e_n] = \Pr[C[G] : S] + \Pr[C[G] : \neg S \land (e_1 \lor \cdots \lor e_n)]$$

$$= \frac{1}{2} + \Pr[C[G] : \neg S] \Pr[C[G] : e_1 \lor \cdots \lor e_n]$$

$$= \frac{1}{2} + \frac{1}{2} \Pr[C[G] : e_1 \lor \cdots \lor e_n]$$
Impact on OEKE: Notations

- dictionary size $N$
- $N_U$ client instances under active attack
- $N_S$ server instances under active attack
- $N_P$ sessions under passive attack
- $q_h$ hash queries
Impact on OEKE: semantic security

- Standard computation of probabilities:
  \[ \text{Adv}_G^{ake}(C) \leq \frac{4N_S + 2N_U}{N} + 8q_h \times \text{Succ}_{G}^{\text{cdh}}(t') + \text{collision terms} \]

- Improved computation of probabilities:
  \[ \text{Adv}_G^{ake}(C) \leq \frac{N_S + N_U}{N} + q_h \times \text{Succ}_{G}^{\text{cdh}}(t') + \text{collision terms} \]

- The adversary can test **one password per session** with the parties.
Impact on OEKE: one-way authentication

- Standard computation of probabilities:

$$\text{Adv}^{c_{-\text{auth}}}_{G_0}(C) \leq \frac{2N_S + NU}{N} + 3q_h \times \text{Succ}_{cdh}^{G}(t') + \text{collision terms}$$

- Improved computation of probabilities:

$$\text{Adv}^{c_{-\text{auth}}}_{G_0}(C) \leq \frac{N_S + NU}{N} + q_h \times \text{Succ}_{cdh}^{G}(t') + \text{collision terms}$$

- The adversary can test one password per session with the parties.

This remark is general: it is not specific to OEKE or to CryptoVerif, and can be used in any proof by sequences of games.
CryptoVerif input

CryptoVerif takes as input:

- The **assumptions** on security primitives: CDH, Ideal Cipher Model, Random Oracle Model.
  - These assumptions are formalized in a library of primitives. The user does not have to redefine them.
- The **initial game** that represents the protocol OEKE:
  - Code for the client
  - Code for the server
  - Code for sessions in which the adversary listens but does not modify messages (passive eavesdroppings)
  - Encryption, decryption, and hash oracles
- The **security properties** to prove:
  - Secrecy of the keys $sk_U$ and $sk_S$
  - Authentication of the client to the server
- **Manual proof indications** (see next slide)
Manual proof indications

1. The proof uses **two events** corresponding to the two cases in which the adversary can guess the password:
   - The adversary impersonates the server by encrypting a $Y$ of its choice under the right password $pw$, and sending it to the client.
   - The adversary impersonates the client by sending a correct authenticator $Auth$ that it has built to the server.

First, one uses manual proof indications to **manually insert these two events**.

   - CryptoVerif cannot guess where events should be inserted.

2. After that, one runs the **automatic proof strategy** of CryptoVerif.

3. Finally, one uses manual transformations to **eliminate uses of the password**.

All manual commands are checked by CryptoVerif, so that an incorrect proof cannot be produced.
Uses of the password after automatic transformations

- Goal: in the final game, the password is not used at all.
- The encryptions/decryptions under the password $pw$ are transformed into lookups that compare $pw$ to keys used in other encryption/decryption queries.
- After the automatic game transformations, the (random) result of some of these encryptions/decryptions is used only in comparisons with previous encryption/decryption queries. We remove the corresponding lookups that compare with $pw$, using manual transformations.
Delaying random choices: $Y_U$ (1)

Client $U$

\[ \begin{align*}
    Y_U & \leftarrow \mathcal{D}_{pw}(Y_U^*) \\
    K_U & \leftarrow Y_U^x \\
    Auth & \leftarrow \mathcal{H}_1(U||S||X||Y_U||K_U) \\
    sk_U & \leftarrow \mathcal{H}_0(U||S||X||Y_U||K_U)
\end{align*} \]

Decryption oracle

\[ (m, kd) \mapsto \textbf{return} \mathcal{D}_{kd}(m) \]
Delaying random choices: $Y_U$ (2)

Client $U$

\[ \begin{aligned}
\text{... find } D_{pw}(Y_U^*) \text{ or } E_{pw}(\cdot) = Y_U^* \text{ in previous queries then } &\ldots \\
\text{else } Y_U \xleftarrow{\mathcal{R}} G; \text{ Auth } \xleftarrow{\mathcal{R}} H_1; \text{ sk}_U \xleftarrow{\mathcal{R}} H_0
\end{aligned} \]

Decryption oracle

\[ (m, kd) \mapsto \begin{aligned}
\text{... find } D_{kd}(m) \text{ or } E_{kd}(\cdot) = m \text{ in previous queries then } &\ldots \\
\text{else } Y_d \xleftarrow{\mathcal{R}} G; \text{ return } Y_d
\end{aligned} \]

$Y_U$ used only in comparisons with previous queries.
move array $Y_U$: Move the choice of $Y_U$ to the point at which it is used.

In OEKE, this point is the decryption oracle. This oracle can return two randomly chosen values:
- the one that comes from the delayed choice of $Y_U, Y'_U$,
- the one that comes from fresh decryption queries, $Y_d$.

After simplification, we have a find with several branches that execute the same code up to variable names ($Y'_U$ vs. $Y_d$).

Merge these branches, thus removing the test of the find, which included the comparison with $pw$. 
move array $Y_U$: Move the choice of $Y_U$ to the point at which it is used.

After simplification, we have a **find** with several branches that execute the same code up to variable names ($Y'_U$ vs. $Y_d$).

Client $U$

**find** $D_{pw}(Y^*_U)$ or $E_{pw}(\cdot) = Y^*_U$ in previous queries **then** . . .

**else** $Auth \leftarrow H_1; sk_U \leftarrow H_0$

Decryption oracle

$(m, kd) \mapsto **find** D_{kd}(m)$ or $E_{kd}(\cdot) = m$ in previous queries **then** . . .

**else find** $j$ such that $m = Y^*_U[j] \land kd = pw$

**then** $Y'_U \leftarrow G; \text{return } Y'_U$

**else** $Y_d \leftarrow G; \text{return } Y_d$

**Merge these branches**, thus removing the test of the **find**, which included the comparison with $pw$. 
Delaying random choices (5)

- **move array** $Y_U$: Move the choice of $Y_U$ to the point at which it is used.

- After simplification, we have a **find** with several branches that execute the same code up to variable names ($Y'_U$ vs. $Y_d$).

- **Merge these branches**, thus removing the test of the **find**, which included the comparison with $pw$.
  Delicate because the code differs by the variable names ($Y'_U$ vs. $Y_d$) and there exist **finds** on these variables.

  1. **move binder** $r1$: reorder instructions so that they are in the same order in the branches to merge.
  2. **merge arrays** $Y_d$, $Y'_U$: merge the array $Y'_U$ into $Y_d$.
  3. **merge branches**: merge the branches of **find** themselves.
Delaying random choices

- move array, merge arrays, and merge branches are new game transformations.
- Similar technique for two other random values:
  - $Y$ in the eavesdropped sessions,
  - $Y$ in the server.
Final elimination of collisions with the password

After the previous steps:

- We obtain a game in which the only uses of $pw$ are:
  - Comparison between $\text{dec}(Y^*, pw)$ and an encryption query $c = \text{enc}(p, k)$ of the adversary: $c = Y^* \land k = pw$, in the client.
  - Comparison between $Y = \text{dec}(Y^*, pw)$ (obtained from $Y^* = \text{enc}(Y, pw)$) and a decryption query $p = \text{dec}(c, k)$ of the adversary: $p = Y \land k = pw$, in the server.

- We eliminate collisions between the password $pw$ and other keys.

- The difference of probability can be evaluated in two ways:
  - $(q_E + q_D)/N$
    - The password is compared with keys $k$ from $q_E$ encryption queries and $q_D$ decryption queries.
    - Dictionary size $N$.
  - $(N_U + N_S)/N$
Final elimination of collisions with the password

After the previous steps:
- We obtain a game in which the only uses of $pw$ are:
  - Comparison between $dec(Y^*, pw)$ and an encryption query $c = enc(p, k)$ of the adversary: $c = Y^* \land k = pw$, in the client.
  - Comparison between $Y = dec(Y^*, pw)$ (obtained from $Y^* = enc(Y, pw)$) and a decryption query $p = dec(c, k)$ of the adversary: $p = Y \land k = pw$, in the server.
- We eliminate collisions between the password $pw$ and other keys.
- The difference of probability can be evaluated in two ways:
  - $(q_E + q_D)/N$
  - $(N_U + N_S)/N$
    - In the client, for each $Y^*$, there is at most one encryption query with $c = Y^*$ so the password is compared with one key for each session of the client.
    - Similar situation for the server.
    - $N_U$ client instances under active attack
    - $N_S$ server instances under active attack
    - Dictionary size $N$. 

Bruno Blanchet (INRIA, ENS, CNRS)
Final elimination of collisions with the password

After the previous steps:

- We obtain a game in which the only uses of \( pw \) are:
  
  - Comparison between \( dec(Y^*, pw) \) and an encryption query \( c = enc(p, k) \) of the adversary: \( c = Y^* \land k = pw \), in the client.
  
  - Comparison between \( Y = dec(Y^*, pw) \) (obtained from \( Y^* = enc(Y, pw) \)) and a decryption query \( p = dec(c, k) \) of the adversary: \( p = Y \land k = pw \), in the server.

- We eliminate collisions between the password \( pw \) and other keys.

- The difference of probability can be evaluated in two ways:
  
  \[
  \frac{(q_E + q_D)}{N} \quad \frac{(N_U + N_S)}{N}
  \]

  The second bound is the best: the adversary can make many encryption/decryption queries without interacting with the protocol.

  - We extended CryptoVerif so that it can find the second bound.
  
  - We give it the information that the encryption/decryption queries are non-interactive, so that it prefers the second bound.
The case study of OEKE is interesting for itself, but it is even more interesting by the extensions it required in CryptoVerif:

- Treatment of the **Computational Diffie-Hellman** assumption.
- New **manual game transformations**, in particular for inserting events and merging branches of tests.
- Optimization of the **computation of probabilities for Shoup’s lemma**.
- Other optimizations of the computation of probabilities in CryptoVerif.

These extensions are of general interest.