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.)
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 $D$,

$$|\Pr[C[G] : D] − \Pr[C[G'] : D]| \leq p(C, D).$$
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 $V' \subseteq V \cup \text{var}(C)$.
OEKE

Client $U$          Server $S$

<table>
<thead>
<tr>
<th>shared $pw$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x \leftarrow R [1, q - 1]$</td>
</tr>
<tr>
<td>$X \leftarrow g^x$</td>
</tr>
<tr>
<td>$Y \leftarrow D_{pw}(Y^*)$</td>
</tr>
<tr>
<td>$K_U \leftarrow Y^x$</td>
</tr>
<tr>
<td>$Auth \leftarrow H_1(U</td>
</tr>
<tr>
<td>$sk_U \leftarrow H_0(U</td>
</tr>
</tbody>
</table>

$y \leftarrow R [1, q - 1]$  
$Y \leftarrow g^y$  
$S, Y^* \leftarrow Y^*$  
$Y^* \leftarrow E_{pw}(Y)$  
$K_S \leftarrow X^y$
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

$$\forall i \leq N \ldotp \text{new } a : \mathbb{Z}; \text{new } b : \mathbb{Z}; (OA() := \exp(g, a), OB() := \exp(g, b), \\forall i' \leq N' \ldotp OCDH(z : G) := z = \exp(g, \text{mult}(a, b))) \approx \forall i \leq N \ldotp \text{new } a : \mathbb{Z}; \text{new } b : \mathbb{Z}; (OA() := \exp(g, a), OB() := \exp(g, b), \\forall i' \leq N' \ldotp OCDH(z : G) := false)$$
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

$$\forall i \leq N \new a : Z; \new b : Z; (OA() := \exp(g, a), OB() := \exp(g, b),$$
$$\forall i' \leq N' OCDH(z : G) := z = \exp(g, \text{mult}(a, b)))$$

$$\approx$$

$$\forall i \leq N \new a : Z; \new b : Z; (OA() := \exp(g, a), OB() := \exp(g, b),$$
$$\forall i' \leq N' OCDH(z : G) := \text{false})$$

Application: semantic security of hashed El Gamal in the random oracle model (A. Chaudhuri).
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

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

\[\begin{align*}
& \approx \\
!^{ia \leq na} & \textbf{new} \ a : Z; (OA()) := \exp(g, a), Oa() := a, \\
!^{iaCDH \leq naCDH} & \text{OCDHa}(m : G, j \leq nb) := \\
& \text{if} \ Ob[j] \text{ or } Oa \text{ has been called then} \\
& \quad m = \exp(g, \text{mult}(b[j], a)) \\
& \text{else} \ false), \\
!^{ib \leq nb} & \textbf{new} \ b : Z; (OB()) := \exp(g, b), Ob() := b, \\
!^{ibCDH \leq nbCDH} & \text{OCDHb}(m : G, j \leq na) := (\text{symmetric of OCDHa}))
\]
Extending the formalization of CDH in CryptoVerif

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

\[ \approx \]

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

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

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

\[ \text{exp}(g, x) = \text{exp}(g, y) \Rightarrow (x = y) \]
\[ \text{exp}'(g, x) = \text{exp}'(g, y) \Rightarrow (x = y) \]

**Injectivity**

\textbf{new} \ x1 : Z; \textbf{new} \ x2 : Z; \textbf{new} \ x3 : Z; \textbf{new} \ x4 : Z;

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

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

\[ i \leq N \text{new } X : G; \; OX() := X \]
\[ \approx_0 \text{[manual]} \; i \leq N \text{new } x : Z; \; OX() := \exp(g, x) \]

This equivalence is very general, apply it only manually.

\[ i \leq N \text{new } X : G; (OX()) := X, \; i' \leq N' \; OXm(m : Z)[\text{useful_change}] := \exp(X, m) \]
\[ \approx_0 \]
\[ i \leq N \text{new } x : Z; (OX()) := \exp(g, x), \; i' \leq N' \; OXm(m : Z) := \exp(g, \text{mult}(x, m)) \]

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

\[ i \leq N \text{new } x : Z; \; OX() := \exp(g, x) \]
\[ \approx_0 \; i \leq N \text{new } X : G; \; OX() := X \]

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 $m = \exp(g, \mult(b, a))$ typically occurs inside the condition of a `find`.
  - This `find` comes from the transformation of a hash function in the Random Oracle Model.

After transformation, we obtain a `find` inside the condition of a `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.
Goal: bound $\Pr[C[G_0] : e_0]$.

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

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

$G_0$ $\uparrow$ probability $p$

$G_n$ $\downarrow$ differ only when $e$ is executed

$G_{n+1}$ event $e$

$\downarrow$ probability $p'$

$G_{n'}$ 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 of secrecy

**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 $S$.

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

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 $S$.

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

\[
\begin{align*}
G_0 \mid R_x \\
\uparrow \text{probability } p \\
G_n \mid R_x \quad \text{secrecy proved: } \Pr[C[G_n \mid R_x] : S] = \frac{1}{2}
\end{align*}
\]

\[
\text{Adv}_{G_0}^{\text{secrecy}(x)}(C) = 2 \Pr[C[G_0 \mid R_x] : S] - 1 \\
\leq 2(p + \Pr[C[G_n \mid R_x] : S]) - 1 \\
\leq 2p
\]
Proof of secrecy with Shoup’s lemma

\[ G_0 \ | \ R_x \quad \text{goal: secrecy of } x \text{ in } G_0 \]

\[ \mathrel{\uparrow\downarrow} \text{probability } p \]

\[ G_n \ | \ R_x \quad \text{differ only when } e \text{ is executed} \]

\[ \mathrel{\uparrow\downarrow} \]

\[ G_{n+1} \ | \ R_x \quad \text{event } e \]

\[ \mathrel{\uparrow\downarrow} \text{probability } p' \]

\[ G_{n'} \ | \ R_x \quad \text{secrecy proved: } \Pr[C[G_{n'} \mid R_x] : S] = \frac{1}{2} \]

\[ \mathrel{\uparrow\downarrow} \text{probability } p'' \]

\[ G_{n''} \ | \ R_x \quad \text{event } e \text{ never executed} \]

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

| $G_0 \mid R_x$ | goal: secrecy of $x$ in $G_0$  
| --- | ---  
| $\uparrow$ probability $p$  
| $G_n \mid R_x$ | differ only when $e$ is executed  
| $\uparrow$ probability $p'$  
| $G_{n+1} \mid R_x$ | event $e$  
| $\uparrow$ probability $p''$  
| $G_n' \mid R_x$ | secrecy proved: $\Pr[C[G_n' \mid R_x] : S] = \frac{1}{2}$  
| event $e$ is independent of $S$  
| $\uparrow$ probability $p''$  
| $G_n'' \mid R_x$ | event $e$ never executed  

\[
\text{Adv}^\text{secrecy}(C)_{G_0} \leq 2(p + p' + \Pr[C[G_n' \mid R_x] : S \lor e]) - 1 \\
\leq 2(p + p' + \frac{1}{2} \Pr[C[G_n' \mid R_x] : e]) \leq 2(p + p') + p''
\]
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:
  \[ \text{Adv}^{\text{ake}}_{G_0}(C) \leq \frac{4N_S + 2N_U}{N} + 8q_h \times \text{Succ}_{\cdh}^{\text{cdh}}(t') + \text{collision terms} \]

- Improved computation of probabilities:
  \[ \text{Adv}^{\text{ake}}_{G_0}(C) \leq \frac{N_S + N_U}{N} + q_h \times \text{Succ}_{\cdh}^{\text{cdh}}(t') + \text{collision terms} \]

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

- Standard computation of probabilities:

\[ \text{Adv}_{G_0}^{c-\text{auth}}(C) \leq \frac{2N_S + N_U}{N} + 3q_h \times \text{Succ}_{G_0}^{\text{cdh}}(t') + \text{collision terms} \]

- Improved computation of probabilities:

\[ \text{Adv}_{G_0}^{c-\text{auth}}(C) \leq \frac{N_S + N_U}{N} + q_h \times \text{Succ}_{G_0}^{\text{cdh}}(t') + \text{collision terms} \]

- 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 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 \( U \)

\[
\begin{align*}
Y_U & \leftarrow D_{pw}(Y_U^*) \\
K_U & \leftarrow Y_U^x \\
Auth & \leftarrow H_1(U || S || X || Y_U || K_U) \\
sk_U & \leftarrow H_0(U || S || X || Y_U || K_U)
\end{align*}
\]

Decryption oracle

\[
(m, kd) \mapsto \text{return } D_{kd}(m)
\]
Delaying random choices: $Y_U$ (2)

Client $U$

\[\ldots\]

**find** $D_{pw}(Y_U^*)$ or $E_{pw}(\cdot) = Y_U^*$ in previous queries **then** \[\ldots\]
**else** $Y_U \leftarrow G$; $Auth \leftarrow H_1$; $sk_U \leftarrow H_0$

Decryption oracle

\[(m, kd) \mapsto \textbf{find } D_{kd}(m) \text{ or } E_{kd}(\cdot) = m \text{ in previous queries } \textbf{then } \ldots\]
**else** $Y_d \leftarrow G$; **return** $Y_d$

$\Rightarrow Y_U$ 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 $pw$. 
Delaying random choices (4)

- 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 $Auth \leftarrow R \ H_1; \ sk_U \leftarrow R \ H_0$

Decryption oracle

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

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

then $Y'_U \leftarrow R \ G; \ return \ Y'_U$

else $Y_d \leftarrow R \ G; \ return \ Y_d$

- 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 $\text{dec}(Y^*, \text{pw})$ and an encryption query $c = \text{enc}(p, k)$ of the adversary: $c = Y^* \land k = \text{pw}$, in the client.
  - Comparison between $Y = \text{dec}(Y^*, \text{pw})$ (obtained from $Y^* = \text{enc}(Y, \text{pw})$) and a decryption query $p = \text{dec}(c, k)$ of the adversary: $p = Y \land k = \text{pw}$, 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)  OEKE in CryptoVerif  January 2012  34 / 35
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:
  - \( \frac{q_E + q_D}{N} \)
  - \( \frac{N_U + N_S}{N} \)

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.