Automatically Verified Mechanized Proof of One-Encryption Key Exchange

Bruno Blanchet
blanchet@di.ens.fr

INRIA, École Normale Supérieure, CNRS, Paris

March 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 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

- \( C[G] \) = game \( G \) interacting with an adversary (evaluation context) \( C \).
- \( C[G] \) may execute events, collected in a sequence \( \mathcal{E} \).
- A distinguisher \( D \) takes as input \( \mathcal{E} \) and returns \text{true} or \text{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 \( D \),

\[
\]
## OEKE

### Assumptions

Client $U$ and Server $S$ share a password $pw$. The protocol proceeds as follows:

1. $x \leftarrow_R [1, q - 1]$
2. $X \leftarrow g^x$
3. $y \leftarrow_R [1, q - 1]$
4. $Y \leftarrow g^y$
5. $Y^* \leftarrow D_{pw}(Y)$
6. $S, Y^* \leftarrow E_{pw}(Y)$
7. $K_U \leftarrow Y^x$
8. $sk_U \leftarrow H_0(U \parallel S \parallel X \parallel Y \parallel K_U)$
9. $Auth \leftarrow H_1(U \parallel S \parallel X \parallel Y \parallel K_U)$
10. $K_S \leftarrow X^y$
11. $sk_S \leftarrow H_0(U \parallel S \parallel X \parallel Y \parallel K_S)$

If $Auth = H_1(U \parallel S \parallel X \parallel Y \parallel K_S)$, then $sk_S$. 

### The proof

The protocol is secure under the assumption that the password $pw$ is kept secret. The security proof involves showing that an adversary cannot forge a valid session key without knowing the password.

---

Bruno Blanchet (INRIA, ENS, CNRS)  
OEKE in CryptoVerif  
March 2012  
5 / 23
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:

⇒ 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.
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 $a, b \in \mathbb{Z}_q$. 
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

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

\[
\begin{align*}
\forall i \leq N & \quad \textbf{new} \ a : Z; \textbf{new} \ b : Z; (OA() := \exp(g, a), OB() := \exp(g, b), \\
\forall i' \leq N' & \quad OCDH(z : G) := z = \exp(g, \text{mult}(a, b))) \\
\simeq \\
\forall i \leq N & \quad \textbf{new} \ a : Z; \textbf{new} \ b : Z; (OA() := \exp(g, a), OB() := \exp(g, b), \\
\forall i' \leq N' & \quad OCDH(z : G) := \text{false})
\end{align*}
\]

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

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

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

\[ \begin{align*}
!^{i_a \leq N_a} & \textbf{new } a : Z; \ (OA()) := \exp(g, a), Oa() := a, \\
!^{i_a \leq n_a \text{CDH}} & \text{OCDHa}(m : G, j \leq N_b) := m = \exp(g, \text{mult}(b[j], a))), \\
!^{i_b \leq N_b} & \textbf{new } b : Z; \ (OB()) := \exp(g, b), Ob() := b, \\
!^{i_b \leq n_b \text{CDH}} & \text{OCDHb}(m : G, j \leq N_a) := m = \exp(g, \text{mult}(a[j], b))) \\
\approx_p & \\
!^{i_a \leq N_a} & \textbf{new } a : Z; \ (OA()) := \exp(g, a), Oa() := \text{let } ka = \text{mark in } a, \\
!^{i_a \leq n_a \text{CDH}} & \text{OCDHa}(m : G, j \leq N_b) := \\
& \text{find } u \leq n_b \text{ suchthat defined}(kb[u], b[u]) \land b[j] = b[u] \text{ then } \\
& m = \exp(g, \text{mult}(b[j], a)) \\
& \text{else if defined}(ka) \text{ then } m = \exp(g, \text{mult}(b[j], a)) \text{ else false),} \\
!^{i_b \leq N_b} & \textbf{new } b : Z; \ (OB()) := \exp(g, b), Ob() := \text{let } kb = \text{mark in } b, \\
!^{i_b \leq n_b \text{CDH}} & \text{OCDHb}(m : G, j \leq N_a) := (\text{symmetric of OCDHa}) \\
\end{align*} \]
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 $m = \text{exp}(g, \text{mult}(b, a))$ 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.
Shoup’s lemma

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

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

$G_0$  

\[ \uparrow \Pr[C[G_{n+1}] : e] \]

$G_n$  

\[ \uparrow \text{ event } e \]

$G_{n+1}$  

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

$G_{n'}$  

\[ \uparrow \text{ events } e_0 \text{ and } e \text{ never executed} \]

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

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

- $G_0$ line:
  - $\uparrow$ probability $p$

- $G_n$ line:
  - $\uparrow$ differ only when $e$ is executed

- $G_{n+1}$ line:
  - event $e$
  - $\uparrow$ probability $p'$

- $G_{n'}$ line:
  - 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 $G \approx_p G'$, then
   
   $\Pr[C[G] : D] \leq p(C, D) + \Pr[C[G'] : D]$.


We also gain a factor 2 for the probability of events in proofs of secrecy, using a similar technique.
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:
  \[ \text{Adv}_{G_0}^{\text{ake}}(C) \leq \frac{4N_S + 2N_U}{N} + 8q_h \times \text{Succ}_{\text{cdh}}^{\text{G}}(t') + \text{collision terms} \]

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

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

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

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

- 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 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)
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.
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.
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

- By previous transformations, the encryptions/decryptions under $pw$ become **lookups that compare $pw$** to keys used in other encryption/decryption queries.

- The (random) result $Y$ of some of these encryptions/decryptions is used only in comparisons with previous encryption/decryption queries.

  1. **Delay the choice of these $Y$.**
     These $Y$ are then chosen in the decryption oracle.
     The decryption oracle choses:
     - the delayed $Y$, $Y'$, or
     - the standard fresh decryption result $Y_d$.

  2. **Merge $Y'$ into $Y_d$.**

  3. **Merge the two branches of **find** that choose $Y_d$.**
Final elimination of collisions with the password

- 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:
  - $(q_E + q_D)/N$
    - 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

- 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 \( N \).
Final elimination of collisions with the password

- 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:
  - $(q_E + q_D)/N$
  - $(N_U + N_S)/N$

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.