Automatic, computational proof of EKE using CryptoVerif
(Work in progress)

Bruno Blanchet
blanchet@di.ens.fr
Joint work with David Pointcheval

CNRS, École Normale Supérieure, INRIA, Paris

May 2010
Motivation

- **EKE (Encrypted Key Exchange):**
  - A password-based key exchange protocol.
  - A non-trivial protocol.
  - It took some time before getting a proper 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.

This work is still in progress.
We consider the variant of EKE of [Bresson, Chevassut, Pointcheval, CCS’03].

\[
\begin{array}{c|c}
\text{Client } U & \text{Server } S \\
\hline
\text{shared } pw \\
 X \leftarrow g^x & Y \leftarrow g^y \\
 U, X & S, Y^* \\
\end{array}
\]

\[
\begin{aligned}
 Y & \leftarrow D_{pw}(Y^*) \\
 K_U & \leftarrow Y^x \\
 Auth & \leftarrow H_1(U||S||X||Y||K_U) \\
 sk_U & \leftarrow H_0(U||S||X||Y||K_U) \\
 Auth & \rightarrow K_s \leftarrow X^y \\
\text{if } Auth = H_1(U||S||X||Y||K_S) & \text{ then } sk_S \leftarrow H_0(U||S||X||Y||K_S)
\end{aligned}
\]
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

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

\[ \约为 \]

\[ \exists i \leq N \quad \text{new } a : Z ; \text{new } b : Z ; (OA() := \exp(g, a), OB() := \exp(g, b),
\]

\[ \exists i' \leq N' \quad \text{OCDH}(z : G) := z = \exp(g, \text{mult}(a, b))) \]

Application: semantic security of hashed El Gamal in the random oracle model (A. Chaudhuri).
This model is **not sufficient** for EKE 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.
Computational Diffie-Hellman assumption in CryptoVerif

\begin{align*}
!^{i_a \leq N_a} \textbf{new} \ a : Z ; \ (OA() := \exp(g, a), Oa() := a, \\
!^{i_a \text{CDH} \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 \text{CDH} \leq n_b \text{CDH}} \text{OCDHb}(m : G, j \leq N_a) := m = \exp(g, \text{mult}(a[j], b))) \\
\approx \\
!^{i_a \leq N_a} \textbf{new} \ a : Z ; \ (OA() := \exp(g, a), Oa() := \textbf{let} \ ka = \text{mark} \ \textbf{in} \ a, \\
!^{i_a \text{CDH} \leq n_a \text{CDH}} \text{OCDHa}(m : G, j \leq N_b) := \\
\textbf{find} \ u \leq n_b \ \textbf{suchthat} \ \text{defined}(kb[u], b[u]) \land b[j] = b[u] \ \textbf{then} \\
m = \exp(g, \text{mult}(b[j], a)) \\
\textbf{else if} \ \text{defined}(ka) \ \textbf{then} \ m = \exp(g, \text{mult}(b[j], a)) \ \textbf{else} \ \text{false}, \\
!^{i_b \leq N_b} \textbf{new} \ b : Z ; \ (OB() := \exp(g, b), Ob() := \textbf{let} \ kb = \text{mark} \ \textbf{in} \ b, \\
!^{i_b \text{CDH} \leq n_b \text{CDH}} \text{OCDHb}(m : G, j \leq N_a) := (\text{symmetric of OCDHa}))
\end{align*}
Computational Diffie-Hellman assumption in CryptoVerif

\[\begin{align*}
!a & \leq Na \\
\text{new} \ & a : Z; (OA()) := \exp(g, a), Oa()[3] := a, \\
!aCDH & \leq naCDH \\
OCDHa(m : G, j \leq Nb)[\text{required}] := m = \exp(g, mult(b[j]), a) \\
!b & \leq Nb \\
\text{new} \ & b : Z; (OB()) := \exp(g, b), Ob()[3] := b, \\
!bCDH & \leq nbCDH \\
OCDHb(m : G, j \leq Na) := m = \exp(g, mult(a[j], b))
\end{align*}\]
Other declarations for Diffie-Hellman (1)

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

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

**Injectivity**

new \( x_1 : Z; \) new \( x_2 : Z; \) new \( x_3 : Z; \) new \( x_4 : Z; \)
\( \text{mult}(x_1, x_2) = \text{mult}(x_3, x_4) \)
\( \approx_{1/|Z|} \)
\( (x_1 = x_3 \land x_2 = x_4) \lor (x_1 = x_4 \land x_2 = x_3) \)

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

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

This equivalence is very general, apply it only manually.

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

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

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

And the same for \( \exp' \).
The implementation of the support for CDH required two extensions of CryptoVerif:

- An array index $j$ occurs as argument of a function.
- The equality test $m = \exp(g, \mul(b, a))$ 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`.

We added support for these constructs in CryptoVerif.
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

Game 0

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

Game \( n \)

\[ \uparrow \Pr[\text{event } e \text{ in game } n + 1] \]

Game \( n + 1 \) \hspace{1cm} \text{event } e

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

Game \( n' \) \hspace{1cm} \text{event } e \text{ never executed}

\hspace{1cm} \text{no attack}

\[ \Pr[\text{attack in game 0}] \leq \Pr[\text{dist. } 0/n] + \Pr[\text{dist. } n/n + 1] + \Pr[\text{dist. } n + 1/n'] \]

\[ \leq \Pr[\text{dist. } 0/n] + \Pr[\text{dist. } n + 1/n'] + \Pr[\text{dist. } n + 1/n'] \]

\[ \leq p + 2p' \]
Improved version with sets of traces

Game 0

\[ \uparrow \quad p \quad \downarrow \]

Game \( n \)

\[ \uparrow \quad p \quad e \quad \downarrow \]

Game \( n + 1 \)

\( \text{event } e \)

\[ \uparrow \quad p \quad p' \quad \text{no event } e \quad \downarrow \quad \text{no attack} \]

Game \( n' \)

\( \text{event } e \text{ never executed} \)

\( \text{no attack} \)

\[
\text{Tr(attack in game 0)} \subseteq \text{Tr(dist. } 0/n) \cup \text{Tr(dist. } n/n + 1) + \text{Tr(dist. } n + 1/n')
\]

\[
\subseteq \text{Tr(dist. } 0/n) \cup \text{Tr(event } e \text{ in game } n + 1) \cup \text{Tr(dist. } n + 1/n')
\]

\[
\subseteq \text{Tr(dist. } 0/n) \cup \text{Tr(dist. } n + 1/n') \cup \text{Tr(dist. } n + 1/n')
\]

So \( \Pr[\text{attack in game 0}] \leq p + p' \).
Impact on EKE

- The proof of [Bresson et al, CCS’03] uses the standard Shoup lemma. Probability of an attack:

$$3 \times \frac{q_s}{N} + 8q_h \times \text{Succ}^{cdh}_G(t') + \text{collision terms}$$

- $q_s$ interactions with the parties
- $q_h$ hash queries
- dictionary size $N$

- With the previous remark and the same proof, we obtain instead:

$$\frac{q_s}{N} + q_h \times \text{Succ}^{cdh}_G(t') + \text{collision terms}$$

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

This remark is general: it is not specific to EKE 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 EKE:
  - 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

- 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 built to the server.
- The manual proof indications consist in **manually inserting these two events**.
  After that, one runs the automatic proof strategy of CryptoVerif.
- All manual commands are **checked** by CryptoVerif, so that an incorrect proof cannot be produced.
- CryptoVerif cannot guess where events should be inserted.
Missing step

One argument is still missing to complete the proof:

- The goal is to obtain a final game in which 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.
- The result of some of these encryptions/decryptions becomes useless after some transformations.

However, CryptoVerif is currently unable to remove the corresponding lookups that compare with $pw$. 
A possible solution

- **Move** the choice of the (random) result of encryption/decryption to the point at which it is used.
  - This point is typically another encryption/decryption query in which we compared with a previous query.

- After simplification, we end up with **finds** that have **several branches** that execute the same code up to variable names.

- **Merge these branches**, thus removing the test of the **find** which included the comparison with **pw**.
  - This merging is delicate because the code differs by the variable names, and there exist **finds** on these variables.
  - The branches of these **finds** must also be merged simultaneously.

This solution is still to verify and implement.
Final step

Assuming the previous step is implemented:

- 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 step

Assuming the previous step is implemented:

- 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:
  - \( (q_E + q_D)/N \)
  - \( (N_U + N_S)/N \)
    - 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.
    - \( N_U \) sessions of the client.
    - \( N_S \) sessions of the server.
    - Dictionary size \( N \).
Final step

Assuming the previous step is implemented:

- 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.
Conclusion

The case study of EKE 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.
- 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.