Mechanized Proofs of Security Protocols with CryptoVerif

Nicola Vitacolonna

University of Udine

Department of Mathematics, Computer Science and Physics

March 21, 2020

The Symbolic (Dolev-Yao) Model: ProVerif

Cryptographic primitives are treated as blackboxes

```
fun enc(K, M): bitstring.
```

Messages are **terms** over such primitives

```
out(channel, enc(k, m));
```

The Symbolic (Dolev-Yao) Model: ProVerif (cont.)

The attacker can compute only using such primitives

```
equation forall k: K, m: M; dec(enc(k, m), k) = m.
```

⇒ **Perfect cryptography:** an encrypted message can be decrypted by the attacker **only** when the attacker knows the key (unrealistic)

Nonetheless, symbolic models are useful for finding **logical** flaws in security protocols (and largely successful at that!)

Computational Security

Computational security goals:

- *Security is only guaranteed against "efficient" adversaries that run for some feasible amount of time*
- *Adversaries can potentially succeed, but with very small probability*

In the following:

- "Adversary" means any probabilistic polynomial-time (PPT) algorithm

The Computational Model of Security Protocols

- Messages are bitstrings
- Cryptographic primitives are functions on bitstrings
- The adversary is a PPT Turing machine
- More realistic than the symbolic model
- Still a model, though: e.g., side channels, timing attacks, …, are out of scope

Overview of Computational Tools (Barbosa et al., 2019)

Tool		RF	Auto	Comp	CS	Link	TCB
AutoG&P$^{\diamond}$	[55]	◐	●	○	◐	○	self, SMT
CertiCrypt$^{\triangleright\diamond}$	[56]	◑	○	○	●	●	Coq
CryptHOL$^{\diamond}$	[57]	◑	○	●	◐	○	Isabelle
CryptoVerif$^{\star\diamond}$	[58]	◐	●	○	●	●	self
EasyCrypt$^{\triangleright\diamond}$	[59]	◑	○	●	◐	●	self, SMT
F7$^{\diamond}$	[17]	◐	○	●	○	●	self, SMT
F*$^{\diamond}$	[60]	◐	○	●	○	●	self, SMT
FCF$^{\diamond}$	[61]	◑	○	●	◐	●	Coq
ZooCrypt$^{\diamond}$	[62]	◐	●	○	●	○	self, SMT

Reasoning Focus (RF)	Concrete security (CS)	Specification language
◐ – automation focus	● – security + efficiency	⋆ – process calculus
◑ – expressiveness focus	◐ – security only	▷ – imperative
	○ – no support	◇ – functional

CryptoVerif

- Automated prover the computational model
- Proofs presented as sequences of games, obtained by suitable transformations (**game hopping**)
- Transformations preserve game equivalence up a to bounded probability
- Games are represented in a process calculus with **probabilistic semantics**
- All processes run in polynomial time
- Used to verify TLS 1.3, SSH, Kerberos, Signal, WireGuard, etc…

Main References

Most accessible introduction to CryptoVerif:

Bruno Blanchet and David Pointcheval,

Automated Security Proofs with Sequences of Games, 2006

`https://eprint.iacr.org/2006/069`, last revised on Dec 3, 2020

Formal semantics of CryptoVerif:

Bruno Blanchet

A computationally sound mechanized prover for security protocols, 2005

`https://eprint.iacr.org/2005/401`, last revised on Jun 16, 2012

CryptoVerif: A Computationally-Sound Security Protocol Verifier, 2017

`https://prosecco.gforge.inria.fr/personal/bblanche/cryptoverif/cryptoverif.pdf`

Other Resources

CryptoVerif's manual (comes with the software)

The 8th BIU Winter School: Mechanized Computational Protocol Proofs

CryptoVerif Programs

A CryptoVerif program is a list of **declarations** followed by either

- a **process** describing the security protocol, or
- an **equivalence query**, i.e., the request to prove that two processes are indistinguishable
- the request to prove an **indistinguishability property**

The declarations specify in particular the hypotheses on the cryptographic primitives and the security properties to prove

The Simplest CryptoVerif Program

```
process
  0
```

```
RESULT: Proved indistinguishability from the final game
```

Negligible Functions

- A function f is **negligible** iff for every positive polynomial p there is k such that $f(n) < \frac{1}{p(n)}$ for all $n > k$
- Examples: $2^{-n}, 2^{-\sqrt{n}}, n^{-\log n}$
- If $f(n)$ and $g(n)$ are negligible functions then
  - $f(n) + g(n)$ is negligible
  - $p(n) \cdot f(n)$, where $p(n)$ is a polynomial, is negligible

A Trivial Example: Random Guessing Is Hard

Demo

Random Guessing: CryptoVerif Code

```
set minAutoCollElim = pest80.
type D [fixed,large].

event Bad.
query event(Bad).
channel start, adv.
process
  in(start, ());  new x: D;
  out(adv, ());   in(adv, x': D);
  if x = x' then  event Bad
```

One-Way Functions

A function $f\colon \{0,1\}^* \to \{0,1\}^*$ is **one-way** iff

- **Easy to compute:** there is a polynomial-time algorithm $\mathcal{A}_f$ such that $\mathcal{A}_f(x) = f(x)$ for all x
- **Hard to invert:** every PPT algorithm can invert f only with negligible probability

The Inverting Experiment

Let $f\colon \{0,1\}^* \to \{0,1\}^*$ be a function and let $\mathcal{A}$ be any adversary.
Let $\mathsf{Invert}_{f,\mathcal{A}}(n)$ denote the following experiment, parametrized by n:

1. Choose uniform $x \in \{0,1\}^n$
2. Compute $y := f(x)$ and give y to $\mathcal{A}$
3. Let $\mathcal{A}$ output x'
4. Output 1 if $f(x') = y$ ($\mathcal{A}$ **succeeds**) and 0 otherwise

f is **hard to invert** iff $\mathcal{A}$ succeeds with negligible probability, that is, if $\Pr[\mathsf{Invert}_{f,\mathcal{A}}(n) = 1]$ defines a negligible function for every $\mathcal{A}$

Example 2: Self-Composition of One-Way Permutations

Lemma: If f is a one-way permutation, then $g := f \circ f$ is a one-way function

Note: the statement does not hold if f is an arbitrary function

Example 2: Game Transformations (1)

```
  Game 1 is
       in(start, ());
       new x: D;
  (1)  out(adv, g(x));
       in(adv, x': D);
(1,2)  if (g(x) = g(x')) then
         event Bad
```

```
Game 2 is
     in(start, ());
     new x: D;
     out(adv, f(f(x)));
     in(adv, x': D);
     if x = x' then
       event Bad
```

1. Purely syntactic transformation (apply the definition of g)
2. Apply injectivity of f

Example 2: Game Transformations (2)

```
Game 2 is
      in(start, ());
      new x: D;
      out(adv, f(f(x)));
      in(adv, x': D);
(1)   if x = x' then
        event Bad
```

```
Game 3 is
      in(start, ());
      new x: D;
      out(adv, f(f(x)));
      in(adv, x': D);
      if (false) then
        event Bad
```

1. Apply one-wayness

Example 2: Game Transformations (3)

```
Game 3 is
      in(start, ());
      new x: D;
      out(adv, f(f(x)));
      in(adv, x': D);
      if (false) then
        event Bad
```

```
Game 4 is
      in(start, ());
      new x: D;
      out(adv, f(f(x)));
      in(adv, x': D)
```

```
RESULT Proved event(Bad) ==> false up to probability p
```

`p` is a negligible probability

Example 2: CryptoVerif Code

```
event Bad.
query event(Bad).

type D [fixed].

fun f(D): D.
equation forall x: D, x': D; (f(x) = f(x')) = (x = x').

fun g(D): D.
equation forall x: D; g(x) = f(f(x)).
```

Example 2: CryptoVerif Code (2)

```
param n, n1.
proba p.

equiv
  !n x <-R D;
    (Oy() := return (f(x)) | !n1 Oeq(x': D) := return (x' = x))
  <=(p)=>
  !n x <-R D;
    (Oy() := return (f(x)) | !n1 Oeq(x': D) := return (false)).
```

Example 2: CryptoVerif Code (3)

```
channel start, adv.

process
  in(start, ());
  new x: D;
  out(adv, g(x));
  in(adv, x': D);
  if g(x) = g(x') then event(Bad)
```

```
RESULT Proved event(Bad) ==> false up to probability p
```

Private-Key Encryption Schemes

Key generation:

$$k \leftarrow \mathsf{Gen}(1^n)$$

(Probabilistic) encryption algorithm:

$$c \leftarrow \mathsf{Enc}_k(m)$$

Decryption algorithm:

$$m' := \mathsf{Dec}_k(c)$$

Which satisfy:

$$\mathsf{Dec}_k(\mathsf{Enc}_k(m)) = m$$

Indistinguishability Under Chosen Plaintext Attacks

- The challenger generates k and a uniform $b \in \{0, 1\}$
- The adversary $\mathcal{A}$ can adaptively query an **encryption oracle** with a chosen plaintext m_i to obtain $\text{Enc}_k(k, m_i)$
- $\mathcal{A}$ then presents a challenge m_0, m_1 and gets back $\text{Enc}_k(k, m_b)$
- $\mathcal{A}$ can continue to query the oracle, then it outputs $b' \in \{0, 1\}$
- $\mathcal{A}$ **succeeds** iff $b' = b$

Informally, a private-key encryption scheme is **CPA secure** (**IND-CPA**) iff $\mathcal{A}$ succeeds ($b' = b$) with negligible probability

Indistinguishability Under Chosen Ciphertext Attacks

- The challenger generates k and a uniform $b \in \{0,1\}$
- The adversary $\mathcal{A}$ can now adaptively query both an **encryption oracle** $\mathsf{Enc}_k(\cdot)$ and a **decryption oracle** $\mathsf{Dec}_k(\cdot)$
- $\mathcal{A}$ then presents a challenge m_0, m_1 and gets back $c \leftarrow \mathsf{Enc}_k(k, m_b)$
- $\mathcal{A}$ can continue to query both the encryption and decryption oracles, but it is not allowed to query the latter on c
- $\mathcal{A}$ eventually outputs $b' \in \{0,1\}$ and **succeeds** iff $b' = b$

Informally, a private-key encryption scheme is **CCA secure** (**IND-CCA2**) iff $\mathcal{A}$ succeeds with negligible probability

Integrity of Ciphertexts

- The challenger generates k
- The adversary $\mathcal{A}$ can adaptively query an **encryption oracle** $\mathrm{Enc}_k(\cdot)$
- $\mathcal{A}$ eventually outputs a ciphertext c
- $\mathcal{A}$ **succeeds** iff $\mathrm{Dec}_k(c)$ is a valid plaintext that was not queried before

Informally, a private-key encryption scheme is **unforgeable** (or **INT-CTXT**) iff $\mathcal{A}$ succeeds with negligible probability

Authenticated Encryption

A private-key encryption scheme that is CCA-secure and unforgeable is called an **authenticated encryption scheme**

Message Authentication Codes (MAC)

Key generation:

$$k \leftarrow \mathsf{Gen}(1^n)$$

(Probabilistic) tag generation algorithm:

$$t \leftarrow \mathsf{Mac}_k(m)$$

Verification algorithm:

$$\mathsf{Vrfy}_k(m, t) \in \{0, 1\}$$

Which satisfy:

$$\mathsf{Vrfy}_k(m, \mathsf{Mac}_k(m)) = 1$$

Strong Unforgeability under Chosen Message Attacks

- The challenger generates k
- The adversary $\mathcal{A}$ can adaptively query a **MAC oracle** $\mathsf{Mac}_k(\cdot)$ and a **verification oracle** $\mathsf{Vrfy}_k(\cdot,\cdot)$
- $\mathcal{A}$ eventually outputs a pair (m,t)
- $\mathcal{A}$ **succeeds** iff $\mathsf{Vrfy}_k(m,t) = 1$ and $\mathcal{A}$ did not query the oracle with message m getting t as a response

Informally, a message authentication code is **strongly secure** (or **SUF-CMA**) iff $\mathcal{A}$ succeeds with negligible probability

Example 3: Encrypt-Then-MAC

A sends to B a fresh secret s using encrypt-then-MAC with independent pre-shared keys k_e (for encryption) and k_m (for authentication)

$$A \to B: \quad e \leftarrow \mathsf{Enc}_{k_e}(s), \mathsf{Mac}_{k_m}(e)$$

Theorem: encrypt-then-MAC is an authenticated encryption scheme when the underlying encryption scheme is CPA-secure and the MAC is strongly secure

CCA-security of the combined scheme reduces to CPA-security of $(\mathsf{Enc}_{k_e}, \mathsf{Dec}_{k_e})$

Strongly Secure MACs in CryptoVerif

An adversary $\mathcal{A}$ that has oracle access to Mac and Vrfy has a negligible probability of forging a MAC:

$$\Pr[k \leftarrow \mathsf{Gen}(1^n);\ (m,t) \leftarrow \mathcal{A}^{\mathsf{Mac}_k(\cdot),\mathsf{Vrfy}_k(\cdot,\cdot)};\ \mathsf{Vrfy}_k(m,t) = 1] \leq \mathsf{negl(n)}$$

If k is used only in Mac_k and Vrfy_k then $\mathsf{Vrfy}_k(m,t)$ can be 1 only if (m,t) is in the list of pairs (message,tag) corresponding to messages that $\mathcal{A}$ has submitted to the MAC oracle (obtaining the corresponding tag)

Strongly Secure MACs in CryptoVerif

CryptoVerif rewrites something like (simplified pseudo-code):

```
!N1 O1(x) := mac(k,x) | !N2 O2(m,t) := verify(k,m,t)
```

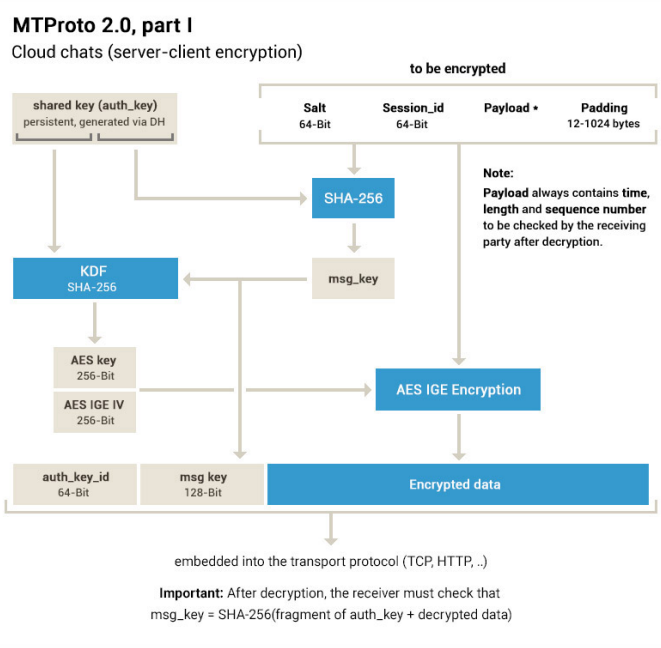
into an **array lookup** that searches through previous oracle queries:

```
!N1 O1(x)   := let t' = mac(k,x) in return (t') |
!N2 O2(m,t) := find j <= N1 suchthat defined(x[j],t'[j])
                 && (m = x[j]) && (t = t') then true else false
```

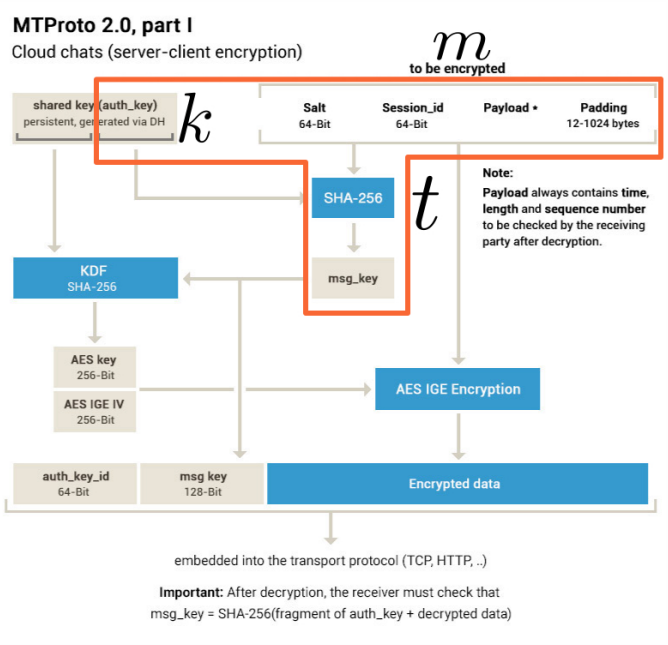
Encrypt-Then-MAC in CryptoVerif

Demo

Telegram's MTProto v2.0



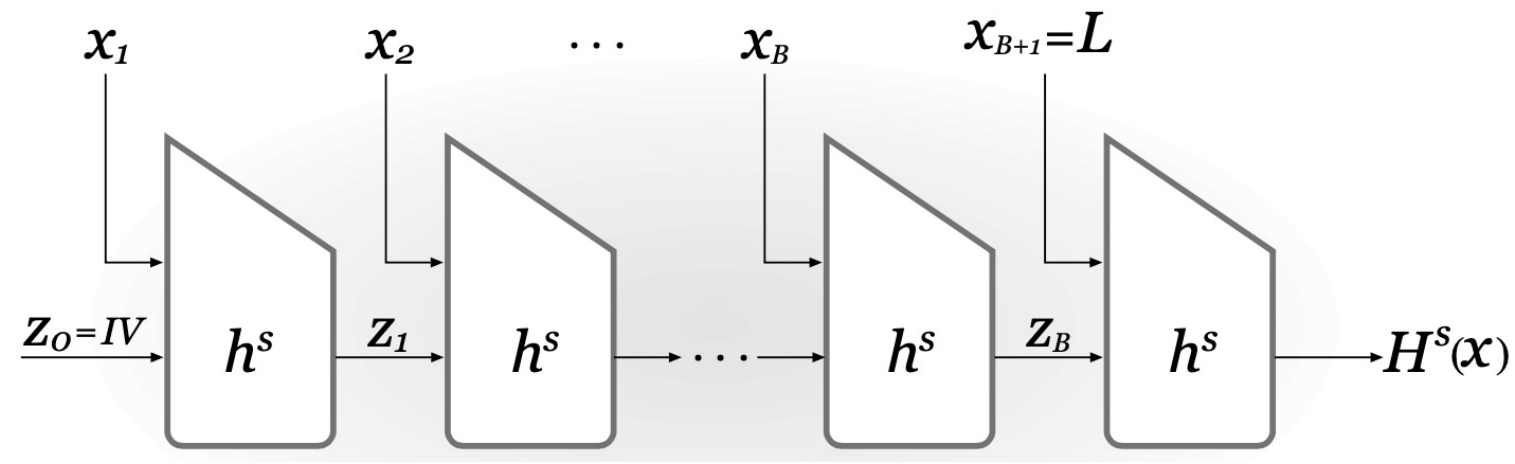
Telegram's MTProto v2.0



Message Authentication in MTProto: Remarks

- MTProto (alleged) authenticated encryption does not use encrypt-then-MAC (or MAC-then-encrypt, or encrypt-and-MAC), but an ad-hoc scheme
- The MAC is verified *after* decryption (see **The Cryptographic Doom Principle**)
- MTProto derives `msg_key` from $H(k_M \ \| \ m)$, which is known to be insecure when H is based on the Merkle-Damgård transform

The Merkle-Damgård Transform



Insecure MACs

- Let H be a keyed hash function built from a fixed-length hash function h with inputs of length n via Merkle-Damgård
- Define $\mathsf{Mac}_{s,k}(m) = H^s(k||m)$
- Request a tag t for an arbitrary message m of length n
- Compute $t' = h^s(t, \langle 3n \rangle)$, where $\langle i \rangle$ is the n-bit encoding of integer i
- Output the forged tag t' on the message $m \,||\, \langle 2n \rangle$
- By construction, $t = \mathsf{Mac}_{s,k}(m) = H^s(k||m) = h^s\Big(h^s\big(h^s(0^n, k), m\big), \langle 2n \rangle\Big)$
- Then $t' = h^s\Big(h^s\big(h^s\big(h^s(0^n, k), m\big), \langle 2n \rangle\big), \langle 3n \rangle\Big)$ is a valid tag