Verifying Cryptographic Protocols

Demi-heure de Science

Adrien Koutsos  Prosecco

9 November 2023
Security Protocols

- **Distributed programs** which aim at providing some **security properties**.
- Uses **cryptographic primitives**: e.g. encryption.
There is a large variety of security properties.

- Confidentiality
- Authentication
- Privacy
Against whom should these properties hold?

- **concretely**, in the **real world**: malicious individuals, corporations, state agencies, ...
- **more abstractly**, one (or many) computers sitting on the network.
Against whom should these properties hold?

- **concretely**, in the **real world**: malicious individuals, corporations, state agencies, ...
- more **abstractly**, one (or many) computers sitting on the network.

**Abstract attacker model**

- **Network capabilities**: worst-case scenario: *eavesdrop, block and forge* messages.
- **Computational capabilities**: the adversary’s *computational power*.
- **Side-channels capabilities**: observing the agents (e.g. time, power-consumption) ⇒ not in this talk.
The Basic Access Control protocol in e-passports:

- uses an RFID tag.
- guard access to information stored.
- should guarantee data confidentiality and user privacy.

Some security mechanisms:

- **integrity**: obtaining key $k$ requires physical access.
- **no replay**: random nonce $n$, old messages cannot be re-used.
Privacy: Unlinkability
No adversary knows whether it interacted with a particular user, in any context.

Example. For two user sessions:

\[
\left( \begin{array}{c}
\text{\textbullet} \\
\end{array} \right)
\left( \begin{array}{c}
\text{\textbullet} \\
\end{array} \right) = \begin{cases} 
\left( \begin{array}{c}
\text{\textbullet} \\
\text{\textbullet} \\
\text{\textbullet} \\
\end{array} \right) & , \quad \text{?} \\
\end{cases}
\]
Privacy: Unlinkability
No adversary knows whether it interacted with a particular user, in any context.

Example. For two user sessions:

BAC Protocol (simplified)

French version of BAC:

- \( \neq \) error messages for replay and integrity checks.

\( \Rightarrow \) unlinkability attack.
BAC Protocol: Privacy Attack
Take-away lessons:

- **This is a protocol-level attack**: no issue with cryptography:
  \[ \Rightarrow \text{cryptographic primitives are but an ingredient.} \]

- **Innocuous-looking changes can break security**:
  \[ \Rightarrow \text{designing security protocols is hard.} \]
Take-away lessons:

- This is a **protocol-level attack**: no issue with cryptography:
  ⇒ cryptographic primitives are but an **ingredient**.
- **Innocuous-looking changes** can **break** security:
  ⇒ designing security protocols is **hard**.

How to get a **strong confidence** in a protocol’s **security guarantees**?
Verification

**Formal mathematical proof** of security protocols:

\[
\begin{align*}
S & \vdash \Phi \\
\text{system} & \quad \text{satisfies} \quad \text{property}
\end{align*}
\]

- Must be **sound**: proof $\Rightarrow$ property always holds
- Usually **undecidable**: approaches either **incomplete** or **interactive**.
- **Machine-checked proofs** yield a high degree of confidence.
  - **general-purpose** tools (e.g. **Coq** and **Lean**).
  - in security protocol analysis, mostly **dedicated** tools.
    E.g. **CryptoVerif**, **EasyCrypt**, **Squirrel**.
Research Goal
Design formal frameworks allowing for mechanized verification of cryptographic protocols.

- At the intersection of cryptography and verification.
- Particular verification challenges:
  - small or medium-sized programs
  - complex properties
  - probabilistic programs + arbitrary adversary
1 Cryptographic Protocol Verification

2 The SQUIRREL Prover
   Mechanized Verification of Security Protocols
Cryptographic Protocol Verification
Verification

\[ \forall \mathcal{A} \in \mathcal{C}. \ (\mathcal{A} \ |\ P) \models \Phi \]

Requires to formalize:

- the protocol under study \( P \).
- the adversarial model, i.e. a class \( \mathcal{C} \) of adversaries.
- the security property \( \Phi \).
- the cryptographic arguments.
Modeling the System

\[ \forall \text{\#} \in C. \ (\text{\#} \parallel P) \models \Phi \]

- **Protocol**: a **concrete** concurrent program.
  
  E.g. imperative or functional progr. language, or applied $\pi$-calculus.

- **Adversary**: an **abstract** unknown program.
  
  What computational capabilities?
Modeling the System

∀φ ∈ C. (φ || P) |= Φ

- **Protocol**: a **concrete** concurrent program.
  - E.g. imperative or functional progr. language, or applied π-calculus.

- **Adversary**: an **abstract** unknown program.
  - What computational capabilities?

  - **Computationally-bounded**: adversary is a **probabilistic Polynomial-TIME program** (PPTIME).
∀ϕ ∈ C. (ϕ || P) ⊨ Φ

- **Protocol**: a **concrete** concurrent program.
  E.g. imperative or functional progr. language, or applied π-calculus.

- **Adversary**: an **abstract** unknown program.
  What computational capabilities?
  - **Quantum adversary**: adversary is a PQT M. ⇒ not in this talk.
  - **Computationally-bounded**: adversary is a **probabilistic**
    Polynomial-TIME program (PPTIME).
∀ϕ ∈ C. (ϕ || P) ⊧ Φ

- **Protocol**: a **concrete** concurrent program.
  E.g. imperative or functional progr. language, or applied π-calculus.

- **Adversary**: an **abstract** unknown program.
  What computational capabilities?
  - **Quantum adversary**: adversary is a PQT(M). ⇒ not in this talk.
  - **Computationally-bounded**: adversary is a probabilistic Polynomial-TIME program (PPTIME).

- **The full system**: interaction (ϕ || P).
Modeling the System

How do we model the interaction (\(\mathcal{H} \mid \mathcal{P}\))?

One **input/output** block:

```
input(in)
y = dec(in, k)
out = enc(y + 1, k)
output(out)
```
Modeling the System

How do we model the interaction ($\mathcal{A} || \mathcal{P}$)?

One **input/output** block:

- Network input $\Rightarrow$ function call to $\mathcal{A}$.

```
input(in)
\[ y = \text{dec}(in, k) \]
\[ \text{out} = \text{enc}(y + 1, k) \]
output(out)
```

```
in = \mathcal{A} ()
\[ y = \text{dec}(in, k) \]
\[ \text{out} = \text{enc}(y + 1, k) \]
output(out)
```
How do we model the interaction $(α || P)$?

One **input/output** block:

- **Network input** $⇒$ function call to $α$.
- **Network output** $⇒$ add to $α$’s knowledge.

\[
\begin{align*}
\text{input}(\text{in}) & \quad \text{in} = α() \\
y = \text{dec}(\text{in}, k) & \quad y = \text{dec}(\text{in}, k) \\
\text{out} = \text{enc}(y + 1, k) & \quad \text{out} = \text{enc}(y + 1, k) \\
\text{output}(\text{out}) & \quad \text{output}(\text{out}) \\
\end{align*}
\]

\[
\begin{align*}
\text{output}(\text{out}) & \quad \text{known} = \text{out} :: \text{known} \\
\end{align*}
\]

Many input/output blocks, add the time:

\[
\begin{align*}
\text{def}(\text{known}) & \quad \text{in} = α(\text{known}) \\
y = \text{dec}(\text{in}, k) & \quad y = \text{dec}(\text{in}, k) \\
\text{out} = \text{enc}(y + 1, k) & \quad \text{out} = \text{enc}(y + 1, k) \\
\end{align*}
\]
Modeling the System

How do we model the interaction \((\mathcal{A} \parallel P)\)?

One **input/output** block:

- Network input \(\Rightarrow\) function call to \(\mathcal{A}\).
- Network output \(\Rightarrow\) add to \(\mathcal{A}\)'s knowledge.

\[
\text{input}(\text{in})
\]
\[
y = \text{dec}(\text{in}, k)
\]
\[
\text{out} = \text{enc}(y + 1, k)
\]
\[
\text{output}(\text{out})
\]

\[
\text{in} = \mathcal{A}() \\
y = \text{dec}(\text{in}, k) \\
\text{out} = \text{enc}(y + 1, k) \\
\text{output}(\text{out})
\]

\[
\text{in} = \mathcal{A}(\text{known}) \\
y = \text{dec}(\text{in}, k) \\
\text{out} = \text{enc}(y + 1, k) \\
\text{known} = \text{out} :: \text{known}
\]

Many **input/output** blocks, add the **time**:

\[
in @ B \overset{\text{def}}{=} \mathcal{A}( \text{known} @ \text{pred}(B) )
\]
\[
\text{known} @ B \overset{\text{def}}{=} \text{out} @ B :: \text{known} @ \text{pred}(B)
\]
Equivalence properties $\Phi$

real/ideal world indistinguishability:

$$S_r \sim S_i$$

- $S_r$: real-world scenario for $\mathcal{P}$.
- $S_i$: ideal-world scenario for $\mathcal{P}$, where security is obvious.

For all $\mathcal{O} \in \text{PPTIME}$:

$$\left| \Pr(S_r(\mathcal{O}) = r) - \Pr(S_i(\mathcal{O}) = r) \right| \text{ negligible}$$

Examples: strong secrecy, privacy.
Cryptographic Arguments

How to prove that no program can break a protocol?
Cryptographic Arguments

How to prove that no program can break a protocol? solve a problem?
Cryptographic Arguments

How to prove that no program can break a protocol?

Cryptographic reduction:

\[
(\Phi \text{ security property of } \mathcal{P})
\]

If an adversary can break \( \Phi \)

then

there exists an adversary breaking \( \mathcal{H} \)

(with similar running time)

Hardness assumption: problem \( \mathcal{H} \) assumed not efficiently solvable.

- mathematical problem (e.g. Discrete-Log).
- lower-level cryptographic problem (e.g. encryption is Ind-CPA).
Hardness Assumption: \textbf{IND-CPA}

A \textbf{symmetric encryption} function $\text{enc}(m, k)$.

\textbf{Hardness assumption:}

\textbf{\textbullet} cannot learn anything from an \textbf{encrypted} message (except its length).

\textbf{Equivalence} $S_L \sim S_R$:

- \textbf{\textbullet} chooses $m_L, m_R$ of the same length
- $S_L$: encryption $\text{enc}(m_L, k)$
- $S_R$: encryption $\text{enc}(m_R, k)$

\[ \Pr(S_L(\bullet) = L) - \Pr(S_R(\bullet) = L) \] negligible

\textbf{P secure} if:

for all \textbf{\textbullet} breaking $P$ there exists \textbf{\textbullet} breaking $S_L \sim S_R$
Game-hopping
Combines several proof-steps:

\[ S_0 \sim_{\epsilon_1} \cdots \sim_{\epsilon_n} S_n \Rightarrow \]
\[ S_0 \sim_{\epsilon_1 + \cdots + \epsilon_n} S_n \]

Each step \( S_i \sim_{\epsilon_{i+1}} S_{i+1} \) justified by:

- a cryptographic reduction;
- a probabilistic argument (e.g. small probability of guessing);
- etc...
Previous slides: cryptographers’ point-of-view.
Cryptographic Arguments as Reasoning Rules

- Previous slides: cryptographers’ point-of-view.
- A more abstract and logical presentation as reasoning rules:
  - **Structural**, to organize proofs:
    \[
    \frac{u \sim w \quad w \sim v}{u \sim v}
    \]
  - **Cryptographic**, e.g. **IND-CPA**:
    \[
    \frac{\text{len}(m_0) = \text{len}(m_1)}{\text{enc}(m_0, k) \sim \text{enc}(m_1, k)} \quad \text{when } k \text{ correctly used in } m_0, m_1
    \]
Cryptographic Arguments as Reasoning Rules

- Previous slides: **cryptographers’ point-of-view**.
- A more **abstract** and **logical** presentation as **reasoning rules**:
  - **Structural**, to organize proofs:
    \[
    \frac{u \sim w \quad w \sim v}{u \sim v}
    \]
  - **Cryptographic**, e.g. **IND-CPA**:
    \[
    \frac{\text{len}(m_0) = \text{len}(m_1)}{\text{enc}(m_0, k) \sim \text{enc}(m_1, k)}
    \]
    when \(k\) correctly used in \(m_0, m_1\)
- **Probabilities** completely **abstracted away**.
- **Application conditions** are the difficult part.
The Squirrel Prover
Mechanized Verification of Security Protocols
The Squirrel Prover

- Tool for verification of security protocols:
  - **Input language**: applied $\pi$-calculus.
  - Automatically translated as input/output blocks.

- Implements a probabilistic logic:
  - Supports **reachability** and **equivalence** properties.
  - **Reasoning rules** valid w.r.t. comp. attacker.
  - In the **asymptotic security** setting.
The Squirrel Prover

- Tool for verification of **security protocols**:
  - **Input language**: applied $\pi$-calculus.
  - Automatically translated as input/output blocks.

- Implements a **probabilistic logic**:
  - Supports **reachability** and **equivalence** properties.
  - **Reasoning rules** valid w.r.t. comp. attacker $\hat{\rho}$.
  - In the **asymptotic security** setting.

- **Proof assistant**:
  - Users prove goals using sequences of tactics.
  - **Crypto.** tactics, e.g. cpa.
  - **Probabilistic** tactics, e.g. fresh.
  - **Structural** tactics, e.g. trans.
  - **Generic** tactics, e.g. apply, rewrite.
The Squirrel Prover
The Squirrel Prover

open-source tool

- Development team: Inria Paris (Prosecco), IRISA (Spicy team).
- Project web-page: https://squirrel-prover.github.io/
- Documentation web-page: https://squirrel-prover.github.io/documentation/
Conclusion
Conclusion

- **Computer-aided verification** of crypto. protocols allows for high security guarantees.

- Quick introduction to protocol verification:
  - **modeling** security properties.
  - formalizing **cryptographic arguments**.

- The **SQUIRREL** prover, an **interactive tool** for crypto. protocol verification.
Thank you for your attention
Hardness Assumption: $\text{Ind-CPA}$

Init: $k \leftarrow$; $\mathcal{O}_{\text{enc}}(m) := \{ \text{return } \text{enc}(m, k) ; \}$

$\mathcal{O}_{\text{challenge}}(m_L, m_R) := \{$
if $\text{len}(m_L) = \text{len}(m_R)$ then
    \text{return } \text{enc}(m_X, k);$$
\}$

$\Pr(S_L(\text{\ding{22}}) = L) - \Pr(S_R(\text{\ding{22}}) = L)$

negligible

guess of the value of $X$
Security Properties

Reachability properties $\Phi$
Directly expressed on $\mathcal{A} + \mathcal{P}$.
For all $\mathcal{A} \in \text{PPTIME}$:

$$\Pr(\text{not } \Phi(\mathcal{A})) \text{ negligible}$$

Examples: authentication, injective authentication, (weak) secrecy.
Security Properties

**Reachability properties $\Phi$**
Directly expressed on $\textsc{\!+} + \mathcal{P}$.
For all $\textsc{\!+} \in \text{PPTIME}$:

$$\Pr(\text{not } \Phi(\textsc{\!+})) \text{ negligible}$$

**Examples:** authentication, injective authentication, (weak) secrecy.
Security Properties

Reachability properties $\Phi$

Directly expressed on $\mathbb{A} + \mathcal{P}$.

For all $\mathbb{A} \in \text{PPTIME}$:

$$\Pr(\text{not } \Phi(\mathbb{A})) \text{ negligible}$$

$$\text{Adv}_{\Phi}(\mathbb{A})$$

Examples: authentication, injective authentication, (weak) secrecy.

$$\forall j. \text{accept@ } 2 \Rightarrow$$

$$(1 < 2 < 2 \land$$

$$\text{out@ } 1 = \text{in@ } 1 \land$$

$$\text{out@ } 2 = \text{in@ } 2$$)
Security Properties

**Reachability properties $\Phi$**
Directly expressed on $\mathcal{A} + \mathcal{P}$.
For all $\mathcal{A} \in \text{PPTIME}$:

$$\Pr(\text{not } \Phi(\mathcal{A})) \text{ negligible}$$

$$\text{Adv}_{\Phi}(\mathcal{A})$$

**Examples:** authentication, injective authentication, (weak) secrecy.

\[
\forall j. \text{ accept@} \ 2_j \Rightarrow
\begin{cases}
\exists i. \text{ out@} \ 1_j = \text{ in@} \ 2_i \\
\text{out@} \ 1_i = \text{ in@} \ 2_j
\end{cases}
\]
From Hardness Assumptions to Logical Rules
Cryptographic reduction: \( \Phi \) security property of \( \mathcal{P} \)

If an adversary \( \mathcal{A} \) can break \( \Phi \)

then

there exists an adversary \( \mathcal{A}' \) breaking \( \mathcal{H} \).
Hardness Assumption: Example

A cryptographic hash function $H(m, k)$.

Unforgeability: cannot produce valid hashes without knowing $k$. 
Hardness Assumption: Example

A cryptographic hash function $H(m, k)$.

Unforgeability: cannot produce valid hashes without knowing $k$.

\[
\begin{align*}
\text{Init: } & k \overset{\$}{\leftarrow} \\
\mathcal{O}_{\text{hash}}(m_0) := & \{ \\
& \mathcal{L} \leftarrow m_0 :: \mathcal{L}; \\
& \text{return } H(m_0, k); \\
\} \\
\end{align*}
\]

\[
\left( (m, s) \text{ such that } m \notin \mathcal{L} \text{ and } s = H(m, k) \right) \text{ negligible.}
\]
Example

\[ \Phi \stackrel{\text{def}}{=} \left( \mathcal{H}(H(0, k), H(1, k)) = H(m, k) \right) \Rightarrow m = 0 \lor m = 1 \]

Proof by reduction

Build an adversary against Unforgeability (UF):

- compute \( w_0 \leftarrow \mathcal{O}_{\text{hash}}(0) \) and \( w_1 \leftarrow \mathcal{O}_{\text{hash}}(1) \);
- black-box call: \( s \leftarrow \mathcal{H}(w_0, w_1) \);
- compute \( m \);
- return \( (m, s) \).

\[ \text{Adv}_{\text{UF}}(\mathcal{A}) = \text{Adv}_{\Phi}(\mathcal{A}) \quad \mathcal{A} \in \text{PPTIME} \implies \mathcal{A} \in \text{PPTIME} \]
Hardness Assumption: Example

Example

\[ \Phi \overset{\text{def}}{=} (\Omega(H(0, k), H(1, k)) = H(m, k)) \implies m = 0 \lor m = 1 \]

Proof by reduction

Build an adversary \( \xi \) against Unforgeability (UF):

- compute \( w_0 \leftarrow O_{\text{hash}}(0) \) and \( w_1 \leftarrow O_{\text{hash}}(1) \);
- black-box call: \( s \leftarrow \Omega(w_0, w_1) \);
- compute \( m \);
- return \((m, s)\).

\[ \text{Adv}_{UF}(\xi) = \text{Adv}_\Phi(\xi) \quad \Omega \in \text{PPTIME implies } \xi \in \text{PPTIME} \]

Remark: rule valid only if \( m \) computable by the adversary, e.g.

\[ \exists \xi_1 \text{ s.t. } \xi_1() = m \]
Until recently:

- **SQUIRREL** supported a limited set of hardness assumptions
  (symmetric/asymmetric encryption, signature, hash, DH, ...)

- Built-in tactics for each such assumptions:
  
  hardness assumption (imperative, stateful programs)
  
  \[\downarrow\]
  
  reasoning rules (pure, logic)
(recent join work with Justine Sauvage and David Baelde)

**Systematic cryptographic reductions**: allows to translate hardness assumptions into cryptographic rules.

**Inputs:**
- an (imperative, stateful) **hardness assumption** $H_0 \sim H_1$.
- a (logical) **security property**, e.g. $S_0 \sim S_1$.

**Goal**: for any $\bullet$, synthesize $\ast$ such that
\[
\begin{cases} 
\ast(H_0) = S_0(\bullet) \\
\text{and } \ast(H_1) = S_1(\bullet)
\end{cases}
\]
From Hardness Assumptions to Logical Rules

- **General framework** to add new hardness assumptions.

- **Proof system** to establish the existence of $\mathcal{H}$.
  - Tracking the state of $\mathcal{H}$: **Hoare pre- and post-conditions**.
    - E.g. track the set of hashed messages $\mathcal{L}$.
  - Correct randomness usage using (logical) **constraints**.
    - E.g. ensures that $\mathcal{H}$ does not directly use $k$.
  - Soundness: existence of a suitable **probabilistic coupling**.

- **Fully automated** (heuristic based $\Rightarrow$ incomplete) procedure.
  Approximate $\mathcal{H}$ state + randomness constraints (discharged to SQUIRREL).