Verifying Cryptographic Protocols

Demi-heure de Science

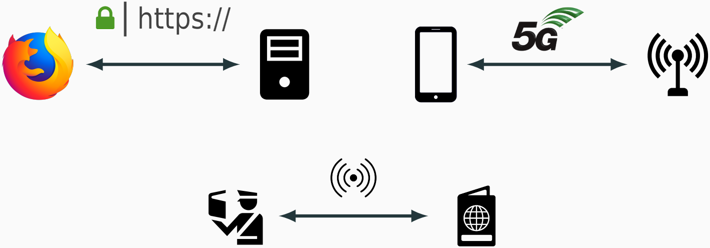
Adrien Koutsos Prosecco

9 November 2023

Context

Security Protocols

- **Distributed programs** which aim at providing some **security properties**.
- Uses **cryptographic primitives**: e.g. encryption.



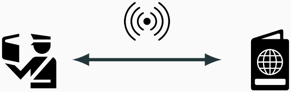
Context: Security Properties

There is a large variety of **security properties**.

Confidentiality

Authentication

Privacy



Privacy

Context: Attacker Model

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.

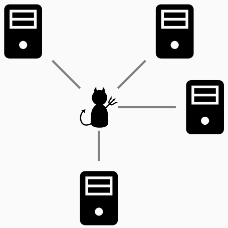
Context: Attacker Model

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) $\Rightarrow$ not in this talk.



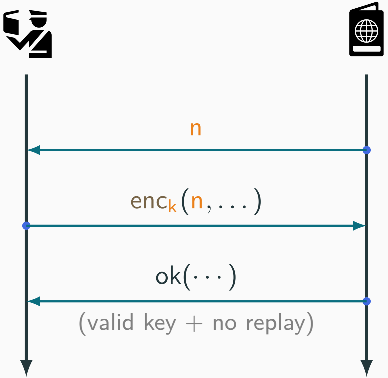
BAC Protocol (simplified)

The Basic Access Control protocol in **e-passports**:

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

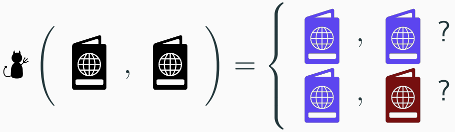
Some security mechanisms:

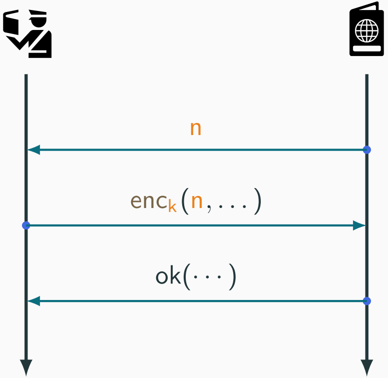
- **integrity**: obtaining **key** k requires **physical access**.
- **no replay**: random nonce n, old messages cannot be re-used.



Privacy: Unlinkability
No adversary knows whether it interacted with a particular user, **in any context**.

Example. For two user sessions:

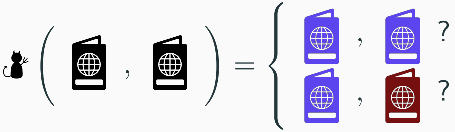




BAC Protocol (simplified)

Privacy: Unlinkability
No adversary knows whether it interacted with a particular user, **in any context**.

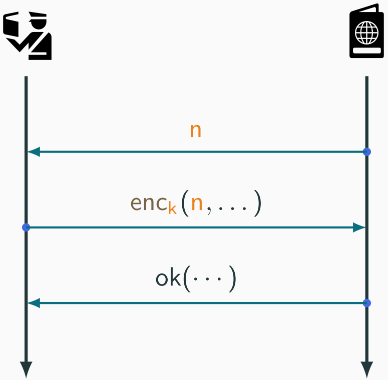
Example. For two user sessions:



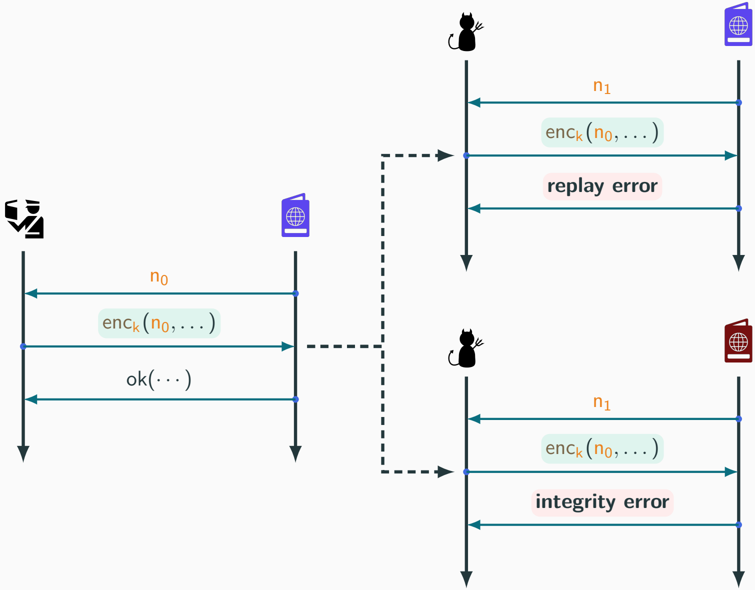
French version of BAC:

- $\neq$ **error messages** for replay and integrity checks.

$\Rightarrow$ **unlinkability attack.**



BAC Protocol: Privacy Attack



BAC Protocol: Lessons

Take-away lessons:

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

BAC Protocol: Lessons

Take-away lessons:

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

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

High-Confidence Security Guarantees

Verification
Formal mathematical proof of security protocols:

$$\underset{\text{system}}{\mathcal{S}} \quad \underset{\text{satisfies}}{\models} \quad \underset{\text{property}}{\Phi}$$

- 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.

Computer-aided Verification of Cryptographic Protocols

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

Outline

1. Cryptographic Protocol Verification

2. The Squirrel Prover
   Mechanized Verification of Security Protocols

Cryptographic Protocol Verification

Cryptographic Protocol Verification

Verification

$\forall$  $\in \mathcal{C}.\ ($  $\,||\, \mathcal{P}) \models \Phi$

Requires to formalize:

- the **protocol** under study $\mathcal{P}$.
- the **adversarial model**, i.e. a class $\mathcal{C}$ of adversaries.
- the **security property** Φ.
- the **cryptographic arguments**.

Modeling the System

$$\forall \; \in \mathcal{C}. \;\; (\; \mid\mid \mathcal{P}) \models \Phi$$

- **Protocol**: a **concrete** concurrent program.

  E.g. imperative or functional progr. language, or applied π-calculus.

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

Modeling the System

$$\forall\ \in \mathcal{C}.\ (\ \,||\, \mathcal{P}) \models \Phi$$

- **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).

$\forall$  $\in \mathcal{C}.\ ($  $\ ||\ \mathcal{P}) \models \Phi$

- **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 PQTM. $\Rightarrow$ not in this talk.
  - **Computationally-bounded**: adversary is a **probabilistic Polynomial-TIME program** ($\mathrm{PPT\textsc{ime}}$).

Modeling the System

$$\forall \text{😈} \in \mathcal{C}.\ \ (\text{😈} \mid\mid \mathcal{P}) \models \Phi$$

- **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 PQTM. $\Rightarrow$ not in this talk.
  - **Computationally-bounded**: adversary is a **probabilistic Polynomial-TIME program** ($\mathrm{PPT{\small IME}}$).

- **The full system:** interaction $(\text{😈} \mid\mid \mathcal{P})$.

Modeling the System

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

One **input/output** block:

$\mathsf{input}(\mathsf{in})$
$\mathsf{y} = \mathsf{dec}(\mathsf{in}, \mathsf{k})$
$\mathsf{out} = \mathsf{enc}(\mathsf{y} + 1, \mathsf{k})$
$\mathsf{output}(\mathsf{out})$

Modeling the System

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

One **input/output** block:

- Network input $\Rightarrow$ function call to .

$$
\begin{array}{l}
\mathsf{input}(\mathsf{in}) \\
\mathsf{y} = \mathsf{dec}(\mathsf{in}, \mathsf{k}) \\
\mathsf{out} = \mathsf{enc}(\mathsf{y} + 1, \mathsf{k}) \\
\mathsf{output}(\mathsf{out})
\end{array}
\longrightarrow
\begin{array}{l}
\mathsf{in} = \;() \\
\mathsf{y} = \mathsf{dec}(\mathsf{in}, \mathsf{k}) \\
\mathsf{out} = \mathsf{enc}(\mathsf{y} + 1, \mathsf{k}) \\
\mathsf{output}(\mathsf{out})
\end{array}
$$

Modeling the System

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

One **input/output** block:

- Network input $\Rightarrow$ function call to .
- Network output $\Rightarrow$ add to 's knowledge.

$$\begin{array}{l}\mathsf{input}(\mathsf{in})\\ \mathsf{y} = \mathsf{dec}(\mathsf{in}, \mathsf{k})\\ \mathsf{out} = \mathsf{enc}(\mathsf{y} + 1, \mathsf{k})\\ \mathsf{output}(\mathsf{out})\end{array} \longrightarrow \begin{array}{l}\mathsf{in} = \ ()\\ \mathsf{y} = \mathsf{dec}(\mathsf{in}, \mathsf{k})\\ \mathsf{out} = \mathsf{enc}(\mathsf{y} + 1, \mathsf{k})\\ \mathsf{output}(\mathsf{out})\end{array} \longrightarrow \begin{array}{l}\mathsf{in} = \ (\mathsf{known})\\ \mathsf{y} = \mathsf{dec}(\mathsf{in}, \mathsf{k})\\ \mathsf{out} = \mathsf{enc}(\mathsf{y} + 1, \mathsf{k})\\ \mathsf{known} = \mathsf{out} :: \mathsf{known}\end{array}$$

Modeling the System

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

One **input/output** block:

- Network input $\Rightarrow$ function call to 😈.
- Network output $\Rightarrow$ add to 😈's knowledge.

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

→

```
in = 😈 ()
y = dec(in, k)
out = enc(y + 1, k)
output(out)
```

→

B

```
in = 😈 (known)
y = dec(in, k)
out = enc(y + 1, k)
known = out :: known
```

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

$$\mathsf{in}@B \overset{\text{def}}{=} \text{😈}(\,\mathsf{known}@\mathsf{pred}(B)\,)$$

$$\mathsf{known}@B \overset{\text{def}}{=} \mathsf{out}@B :: \mathsf{known}@\mathsf{pred}(B)$$

Security Properties

Equivalence properties Φ
real/ideal world indistinguishability:

$$\mathsf{S}_{\mathsf{r}} \sim \mathsf{S}_{\mathsf{i}}$$

- S_{r}: **real-world** scenario for $\mathcal{P}$.
- S_{i}: **ideal-world** scenario for $\mathcal{P}$, where security is obvious.

$$\begin{array}{rl} & \mathtt{out@}\,,\ \mathtt{out@} \\ \sim & \mathtt{out@}\,,\ \mathtt{out@} \end{array}$$

For all $\in \mathrm{PPT}\textsc{ime}$:

$$\Big| \Pr(\mathsf{S}_{\mathsf{r}}() = \mathsf{r}) - \Pr(\mathsf{S}_{\mathsf{i}}() = \mathsf{r}) \Big| \text{ negligible}$$

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~~?
solve a problem?

Cryptographic reduction: (Φ security property of $\mathcal{P}$)

If an adversary can break Φ

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: Ind-CPA

message key

A **symmetric encryption** function $\mathsf{enc}(m, \mathsf{k})$.

Hardness assumption:

cannot learn anything from an **encrypted** message (except its length).

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

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

$$\left| \begin{array}{l} \Pr(\mathsf{S_L}(\cdot) = \mathsf{L}) - \\ \Pr(\mathsf{S_R}(\cdot) = \mathsf{L}) \end{array} \right|$$

negligible

$\mathcal{P}$ **secure** if:

for all breaking $\mathcal{P}$ there exists breaking $\mathsf{S_L} \sim \mathsf{S_R}$

Cryptographic Arguments

Game-hopping
Combines several proof-steps:

$$\mathsf{S}_0 \sim_{\epsilon_1} \cdots \sim_{\epsilon_n} \mathsf{S}_n \quad \Rightarrow$$
$$\mathsf{S}_0 \sim_{\epsilon_1 + \cdots + \epsilon_n} \mathsf{S}_n$$

Each step $\mathsf{S}_i \sim_{\epsilon_{i+1}} \mathsf{S}_{i+1}$ justified by:

- a cryptographic reduction;
- a probabilistic argument (e.g. small probability of guessing);
- *etc...*

Cryptographic Arguments as Reasoning Rules

- 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:

$$\frac{u \sim w \qquad w \sim v}{u \sim v}$$

  - **Cryptographic**, e.g. IND-CPA:

$$\frac{\mathsf{len}(\mathsf{m}_0) = \mathsf{len}(\mathsf{m}_1)}{\mathsf{enc}(\mathsf{m}_0, \mathsf{k}) \sim \mathsf{enc}(\mathsf{m}_1, \mathsf{k})} \qquad \text{when k correctly used in } \mathsf{m}_0, \mathsf{m}_1$$

Cryptographic Arguments as Reasoning Rules

- Previous slides: **cryptographers' point-of-view**.
- A more **abstract** and **logical** presentation as **reasoning rules**:
  - **Structural**, to organize proofs:

$$\frac{u \sim w \qquad w \sim v}{u \sim v}$$

  - **Cryptographic**, e.g. IND-CPA:

$$\frac{\mathsf{len}(\mathsf{m}_0) = \mathsf{len}(\mathsf{m}_1)}{\mathsf{enc}(\mathsf{m}_0, \mathsf{k}) \sim \mathsf{enc}(\mathsf{m}_1, \mathsf{k})} \quad \text{when } \mathsf{k} \text{ correctly used in } \mathsf{m}_0, \mathsf{m}_1$$

- **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 π-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 π-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.
- **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

```
global lemma unlinkability (t : timestamp[const]) :
  [happens(t)] -> equiv(frame@t).
Proof.
  intro Hap.
  enrich
    seq(A:index          => pk( kA    (A  ) )),
    seq(A:index, i:index => pk( kAbis (A,i) )),
    seq(A:index          => pk( kB    (A  ) )).

  induction t.

   (* init *)
  * auto.

  (* Case A *)
  * rewrite /* /=.
    by apply IH.

  (* Case A1 *)
  * rewrite /* /=.
    fa 3; fa 4; fa 4; fa 4; fa 4.
    fresh 6; 1:auto.
    by fresh 5.

  * (* Case B *)
    rewrite /frame /output /exec /cond /dmess /=.
    fa 3; fa 4; fa 4.
    enckp 4; 1: auto.
    cca1 4.
    + auto.
    + (* Pushing conditional underneath len(_) *)
      rewrite if_len !length_pair.
      rewrite (if_same_branch (len(nB(A,i)) ++ len(nB(A,i)))) //.
      fa 4; fa 4; fa 4; fa 4.
      fresh 5; 1:auto.
      by fresh 4.
Qed.
```

```
[goal> Focused goal (1/2):
System: default (same for equivalences)
Variables: A,i:index[const, glob]
Hap: [happens(B(A, i))]
IH: equiv(seq(A:index=>pk (kB A)), seq(A,i:index=>pk (kAbis (A, i))),
          seq(A:index=>pk (kA A)), frame@pred (B(A, i)))
----------------------------------------
forall (A0,i0:index), B(A0, i0) < B(A, i) => A <> A0 || i <> i0
```

```
U:%*-  *goals*        All (1,0)      (Squirrel goals +1 company Helm)
```

```
Checking for side conditions on the right
 Bad occurrences of kA(A) and rB((A, i)) in other actions:
   rB((A, i))
     (collision with rB((A, i)))
     in action B(A, i)
     in term
       enc (if (fst (dmess(A, i)@B(A, i)) = pk (kAbis (A, i)) &&
                len (snd (dmess(A, i)@B(A, i))) = len (nB (A, i))) then
              <snd (dmess(A, i)@B(A, i)),nB (A, i)>
           else <nB (A, i),nB (A, i)>, rB (A, i), pk (kAbis (A, i)))

 Total: 1 bad occurrence
        0 of them are subsumed by another
        1 bad occurrence remaining
```

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: Ind-CPA

Init: $k \xleftarrow{\$}$;

$\mathcal{O}_{\mathsf{enc}}(m) := \{\texttt{return}\ \mathsf{enc}(m, k);\}$

$\mathcal{O}_{\mathsf{challenge}}(m_L, m_R) := \{$

$\quad \texttt{if}\ \mathsf{len}(m_L) = \mathsf{len}(m_R)\ \texttt{then}$

$\qquad \texttt{return}\ \mathsf{enc}(m_X, k);$

$\}$

$\boxed{S_X}$

guess of the value of X

$$\left| \begin{array}{l} \Pr(S_L(\ ) = L) - \\ \Pr(S_R(\ ) = L) \end{array} \right|$$

negligible

Security Properties

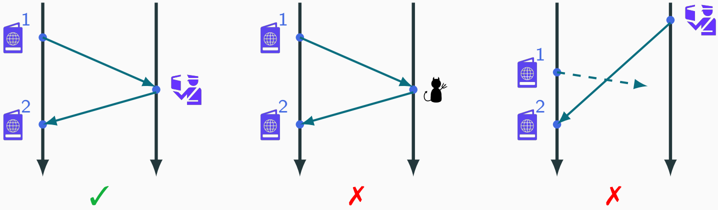
Reachability properties Φ

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

For all $\in \textsc{PPTime}$:

$$\Pr(\textbf{not}\ \Phi(\ ))\quad \text{negligible}$$

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



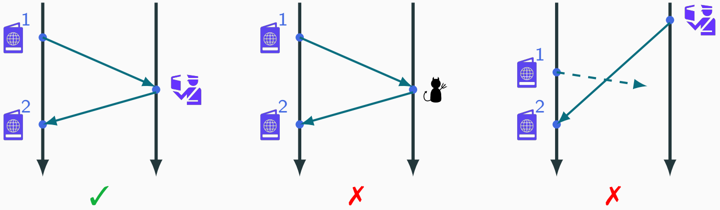
Security Properties

Reachability properties Φ
Directly expressed on $+\ \mathcal{P}$.
For all $\in \mathrm{PPT{\small IME}}$:

$$\Pr(\textbf{not}\ \Phi(\ ))\quad \text{negligible}$$

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

$$\mathsf{accept}@\,^{2} \Rightarrow \begin{pmatrix} ^{1} < \ < \ ^{2} \ \wedge \\ \mathsf{out}@\,^{1} = \mathsf{in}@ \ \wedge \\ \mathsf{out}@ \ = \mathsf{in}@\,^{2} \end{pmatrix}$$



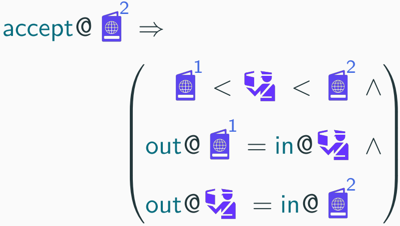
Security Properties

Reachability properties Φ

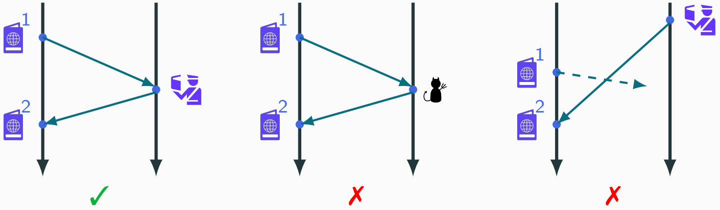
Directly expressed on $+\, \mathcal{P}$.

For all $\in \mathrm{PPT{\small IME}}$:

$$\underbrace{\Pr(\textbf{not}\ \Phi(\ ))}_{\mathsf{Adv}_\Phi(\ )} \text{ negligible}$$



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



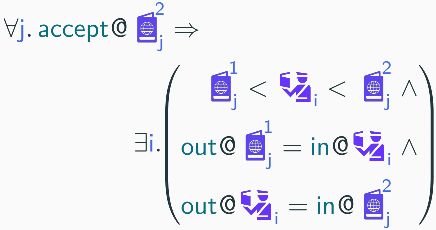
Security Properties

Reachability properties Φ

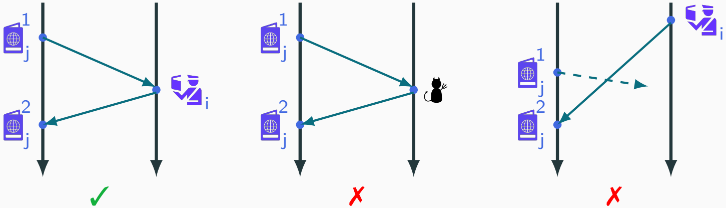
Directly expressed on + $\mathcal{P}$.

For all $\in$ PPTIME:

$$\underbrace{\Pr(\textbf{not}\ \Phi(\ ))}_{\mathsf{Adv}_\Phi(\ )}\ \text{negligible}$$



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



From Hardness Assumptions to Logical Rules

Cryptographic Reduction

Cryptographic reduction: (Φ security property of $\mathcal{P}$)

If an adversary can break Φ

then

there exists an adversary breaking $\mathcal{H}$.

Hardness Assumption: Example

A **cryptographic hash** function $\underbrace{\mathsf{H}(\ m}_{\text{message}},\ \underbrace{\mathsf{k}\ )}_{\text{key}}$.

Unforgeability: cannot produce valid hashes without knowing k.

Hardness Assumption: Example

message ← → key

A **cryptographic hash** function $\mathsf{H}(m, \mathsf{k})$.

Unforgeability: cannot produce valid hashes without knowing k.

$\underline{\text{Init:}}$ $\mathsf{k} \xleftarrow{\$}$;

$\underline{\mathcal{O}_{\text{hash}}(m_0)} := \{$

$\quad \mathcal{L} \leftarrow m_0 :: \mathcal{L}$;

$\quad \texttt{return } \mathsf{H}(m_0, \mathsf{k})$;

$\}$

$$\left(\begin{array}{r} (m, s) \text{ such that } m \notin \mathcal{L} \text{ and} \\ s = \mathsf{H}(m, \mathsf{k}) \end{array} \right)$$ **negligible**.

Hardness Assumption: Example

Example

$$\Phi \stackrel{\text{def}}{=} \quad \Big((\mathsf{H}(0,\mathsf{k}), \mathsf{H}(1,\mathsf{k})) = \mathsf{H}(m,\mathsf{k}) \Big) \Rightarrow m = 0 \ \vee \ m = 1$$

Proof by reduction

Build an adversary against Unforgeability (UF):

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

$$\mathsf{Adv}_{\mathsf{UF}}(\ ) = \mathsf{Adv}_{\Phi}(\ ) \qquad \in \mathrm{PPT}\textsc{ime} \text{ implies } \in \mathrm{PPT}\textsc{ime}$$

Hardness Assumption: Example

Example

$$\Phi \stackrel{\text{def}}{=} \Big((\mathsf{H}(0,\mathsf{k}),\mathsf{H}(1,\mathsf{k})) = \mathsf{H}(m,\mathsf{k}) \Big) \Rightarrow m = 0 \ \vee \ m = 1$$

Proof by reduction

Build an adversary against UNFORGEABILITY (UF):

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

$$\mathsf{Adv}_{\mathsf{UF}}() = \mathsf{Adv}_{\Phi}() \qquad \in \mathrm{PPT}\textsc{ime} \text{ implies } \in \mathrm{PPT}\textsc{ime}$$

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

$$\exists_1 \text{ s.t. } _1() = m$$

From Hardness Assumptions to Logical Rules

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)

From Hardness Assumptions to Logical Rules

(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** $\mathcal{H}_0 \sim \mathcal{H}_1$.
- a (logical) **security property**, e.g. $\mathsf{S}_0 \sim \mathsf{S}_1$.

Goal: for any , synthesize such that

$$\begin{cases} (\mathcal{H}_0) = \mathsf{S}_0() \\ \text{and } (\mathcal{H}_1) = \mathsf{S}_1() \end{cases}$$

From Hardness Assumptions to Logical Rules

- **General framework** to add new hardness assumptions.
- **Proof system** to establish the existence of .
  - 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  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).