MPRI

SECURE: Proofs of Security Protocols

1. A Formal Framework for Cryptographic Protocol Verification

Adrien Koutsos, Inria Paris

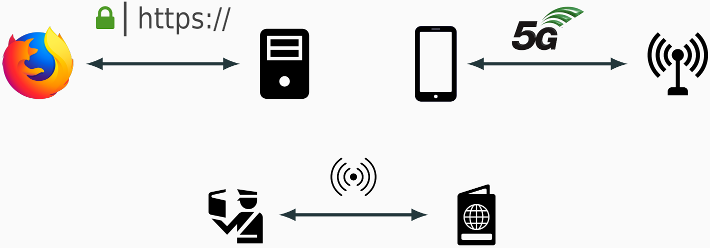
2025/2026

Introduction

Context

Security Protocols

- **Distributed programs** which aim at providing some **security properties**.
- Uses **cryptographic primitives**: e.g. encryptions, signatures, hashes.

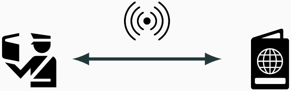


Context: Security Properties

There is a large variety of **security properties** that such protocols must provide.

Confidentiality
Authentication

Privacy



Privacy

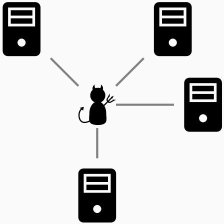
Context: Attacker Model

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.

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 lecture.



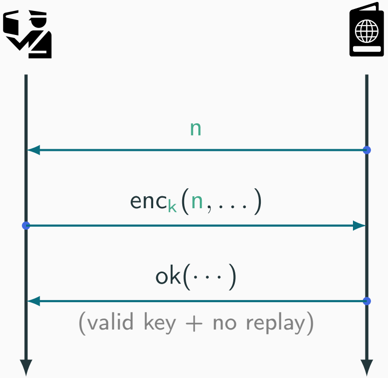
BAC Protocol (simplified)

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.

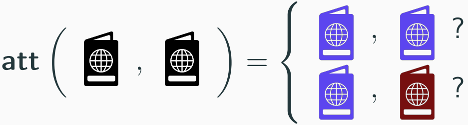


BAC Protocol (simplified)

Privacy: Unlinkability

No adversary can know whether it interacted with a particular user, **in any context**.

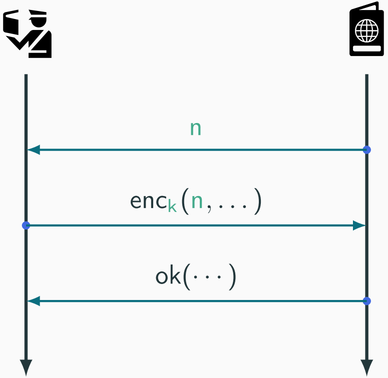
Example. For two user sessions:



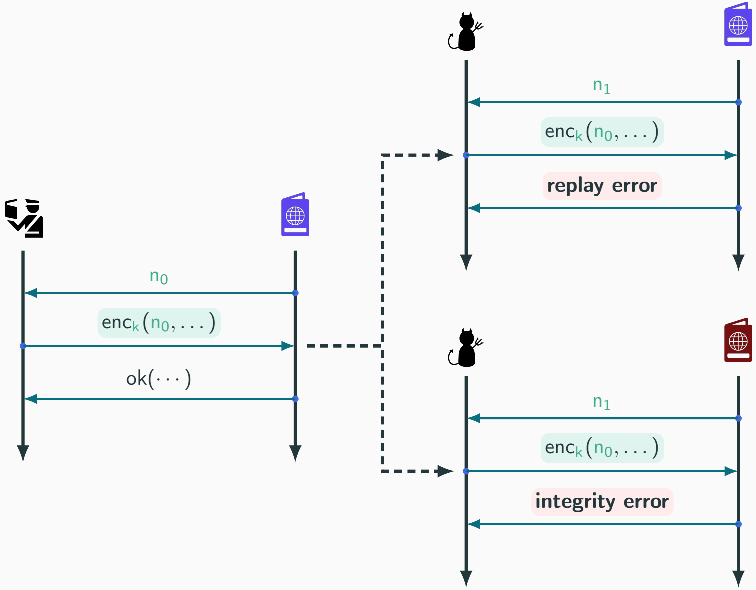
French version of BAC:

- $\neq$ **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**.

Protocol-level attacks are found even on **critical** cryptographic protocols under a lot of **public scrutiny**. Some examples:

- **TLS**, e.g. LogJam$_{14}$ [1], TripleHandshake$_{15}$ [10], Drown$_{16}$ [2], ...
- **Wifi encryption** with WPA2, nonce reuse attacks$_{17,18}$ [12, 13].
- **Credit card** payment system: PIN code bypass$_{21}$ [8, 9].

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

High-Confidence Security Guarantees

Verification

Formal mathematical proof of security protocols:

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

- 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. Rocq and Lean).
  - In security protocol analysis, often in **dedicated** tools.
    E.g. CryptoVerif [11], EasyCrypt [7, 6], Squirrel [4, 3].

Goal of these lectures

Design a **formal framework** allowing for the **mechanized verification** of **cryptographic protocols**.

- At the intersection of **cryptography** and **verification**.
- Particular verification challenges:
  - small or medium-sized programs
  - complex properties
  - concurrent and probabilistic programs + arbitrary adversary

Scope of the Lectures

What is **not** in these lectures:

- **Side-channels** or **post-quantum cryptography**.
- Verification of **implementations**.

  (See Aymeric Fromherz's lectures.)
- Using **tools** (we focus on foundations).

  (See Bruno Blanchet's lectures.)

The CCSA Approach to Cryptographic Protocol Verification

The Computationally Complete Symbolic Attacker (**CCSA**) [5] is a framework in the **computational model** for the **verification** of cryptographic protocols.

Key ingredients

- **Protocol executions** modeled as **pure symbolic terms**.
- A **probabilistic logic**.
  $\Rightarrow$ interpret terms as PTIME-computable bitstring distributions.
- **Reasoning rules** capturing **cryptographic arguments**.
- **Abstract approach**: no probabilities, no security parameter.

Outline

Processes

Processes

Process Syntax

Process: Syntax

Elementary processes:

$$\mathsf{E} ::= \mathbf{in}(\mathsf{c},\mathsf{x}) \mid \mathbf{out}(\mathsf{c},t) \mid \nu\,\mathsf{n} \mid \text{if } b \text{ then } \mathsf{E} \mid \mathsf{E}.\,\mathsf{E} \mid \mathbf{null} \qquad (\mathsf{c} \in \mathcal{C},\ \ \mathsf{x},\mathsf{n} \in \mathcal{X})$$

where $\mathcal{C}$ is a set of channel symbols and $\mathcal{X}$ a set of variables.

Processes:

$$\mathsf{P}_0 ::= \mathsf{E} \mid (\mathsf{P}_0 \mid \mathsf{P}_0) \qquad \mathsf{P} ::= \mathsf{P}_0 \mid \nu\,\mathsf{n}.\,\mathsf{P} \qquad (\mathsf{n} \in \mathcal{X})$$

We let chans(P) be the channels of a process P.

Restrictions: elementary process must use a single channel, and distinct elementary processes must use distinct channels. I.e. if $\mathsf{P} = \mathsf{E}_1 \mid \cdots \mid \mathsf{E}_n$

$$\text{then} \quad \forall i.\ |\mathrm{chans}(\mathsf{E}_i)| \leq 1 \qquad \forall i \neq j, \mathrm{chans}(\mathsf{E}_i) \cap \mathrm{chans}(\mathsf{E}_j) = \emptyset.$$

Example of a Protocol

As an example, we consider a simple authentication protocol:

The Private Authentication (PA) Protocol, v1

$$\begin{aligned} \mathsf{I} &= \nu\, \mathsf{n_I}. & &\mathbf{out}(\mathtt{I}, \{\langle \mathsf{pk_I}\,, \mathsf{n_I} \rangle\}_{\mathsf{pk_S}}) \\ \mathsf{S} &= \nu\, \mathsf{n_S}.\, \mathbf{in}(\mathtt{S}, \mathsf{x}). & &\mathbf{out}(\mathtt{S}, \{\langle \pi_2(\mathsf{dec}(\mathsf{x}, \mathsf{sk_I}))\,, \mathsf{n_S} \rangle\}_{\mathsf{pk_I}}) \end{aligned}$$

where $\mathsf{pk_I} \equiv \mathsf{pk}(\mathsf{k_I})$ and $\mathsf{pk_S} \equiv \mathsf{pk}(\mathsf{k_S})$.

The full protocol is $\nu\, \mathsf{k_I}.\, \nu\, \mathsf{k_S}.\, (\mathsf{I} \mid \mathsf{S})$.

Notation: $\equiv$ *denotes* ***syntactic*** *equality of terms.*

Processes

Terms Syntax and Semantics

Terms

We use **terms** to model *protocol messages*, built upon a set of **symbols** $\mathcal{S}$ which includes:

- **Variables** $\mathcal{X}$, used, e.g. x in $\textbf{in}(\mathtt{A}, \mathrm{x})$ or n in ν n.
- **Function symbols** $\mathcal{F}$, e.g.:

$$\langle \cdot\,,\,\cdot \rangle,\ \pi_1(\cdot),\ \pi_2(\cdot) \qquad \{\cdot\}^{\cdot}_{\cdot},\ \mathsf{pk}(\cdot),\ \mathsf{sk}(\cdot),$$
$$\mathsf{if} \cdot \mathsf{then} \cdot \mathsf{else} \cdot \qquad \cdot = \cdot \qquad \cdot \wedge \cdot,\ \cdot \vee \cdot,\ \cdot \rightarrow \cdot$$

We note $\mathcal{T}(\mathcal{S})$ the set of well-typed (see next slide) terms over symbols $\mathcal{S}$. Terms are usually written t, and boolean terms b.

Examples

$$\mathsf{pk}(\mathsf{k_A}) \qquad \{\langle \mathsf{pk_A}\,,\,\mathsf{n_A} \rangle\}_{\mathsf{pk_B}} \qquad \pi_1(\mathsf{n}_A)$$

Terms: Types

Types

Each symbol $s \in \mathcal{S}$ comes with a type $\mathtt{type}(s)$ of the form:

$$(\tau_{\mathrm{b}}^1 \star \cdots \star \tau_{\mathrm{b}}^n) \to \tau_{\mathrm{b}} \qquad \text{or} \qquad \tau_{\mathrm{b}}$$

where $\tau_{\mathrm{b}}^1, \ldots, \tau_{\mathrm{b}}^n, \tau_{\mathrm{b}}$ are all **base types** in $\mathbb{B}$.

- We ask that $\mathbb{B}$ contains at least the `message` and `bool` types.
- We restrict *variables* to base types, i.e.:

$$\forall x \in \mathcal{X}, \mathtt{type}(x) \in \mathbb{B}.$$

- We require that terms are well-typed and of a base type:

$$\vdash t : \tau_{\mathrm{b}} \qquad \text{where } \tau_{\mathrm{b}} \in \mathbb{B}.$$

Terms: Semantics

The **interpretation** $(\!|t|\!)_{\mathbb{L}}^{\eta,\sigma}$ of a term t as a bitstring is parameterized by:

- the **security parameter** η;
- a **library** $\mathbb{L}$ which provides the semantics $(\!|\cdot|\!)_{\mathbb{L}}$ of **symbols** in $\mathcal{F}$ (details on next slides);
- the **valuation** $\sigma : \mathcal{X} \hookrightarrow \{0,1\}^*$ mapping variables to their values.[1]

We may omit σ, $\mathbb{L}$ and η when they are clear from the context.

[1] $f : \mathbb{A} \hookrightarrow \mathbb{B}$ denotes a *partial* f function from $\mathbb{A}$ to $\mathbb{B}$.

Terms: Semantics

Function symbols.

For a **function symbols** $f \in \mathcal{F}$, we simply apply $(\!|f|\!)_{\mathbb{L}}$:

$$\langle\!|f(t_1, \ldots, t_n)|\!\rangle_{\mathbb{L}}^{\eta,\sigma} \stackrel{\text{def}}{=} (\!|f|\!)_{\mathbb{L}}(1^\eta, \langle\!|t_1|\!\rangle_{\mathbb{L}}^{\eta,\sigma}, \ldots, \langle\!|t_n|\!\rangle_{\mathbb{L}}^{\eta,\sigma})$$

Restriction: $(\!|f|\!)_{\mathbb{L}}$ must be **poly-time** computable and **deterministic**.

Variables.

The interpretation $\langle\!|x|\!\rangle_{\mathbb{L}}^{\eta,\sigma}$ of a **variable** $\mathsf{x} \in \mathcal{X}$ is given by the valuation σ:

$$\langle\!|\mathsf{x}|\!\rangle_{\mathbb{L}}^{\eta,\sigma} \stackrel{\text{def}}{=} \sigma(\mathsf{x})$$

Terms: Builtins

We **force** the interpretation of some **function symbols**.

- if · then · else· is interpreted as **branching**:

$$(\!|\text{if} \cdot \text{then} \cdot \text{else}\cdot|\!)_{\mathbb{M}}(1^\eta, b, v_1, v_2) \stackrel{\text{def}}{=} \begin{cases} v_1 & \text{if } b = 1 \\ v_2 & \text{otherwise} \end{cases}$$

- $\cdot = \cdot$ is interpreted as an **equality** test:

$$(\!|\cdot = \cdot|\!)_{\mathbb{M}}(1^\eta, v_1, v_2) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } v_1 = v_2 \\ 0 & \text{otherwise} \end{cases}$$

Similarly, we force the interpretations of $\wedge, \vee, \rightarrow, \text{true}, \text{false}, \ldots$

Processes

Process Semantics

Process: Concrete Configuration

A **concrete configuration** is a tuple $(\phi, \sigma, \mathsf{P})$ representing a **partially executed process** where:

- the **concrete frame** ϕ is the sequence bitstrings $w_1, \ldots, w_n \in \{0,1\}^*$ outputted since the protocol execution started.
- the **valuation** σ stores the inputs and random samplings.
- the **process** P remains to be executed.

A concrete configuration records the **current state** of an execution.

Initial configuration: $(\epsilon, [\mathsf{st} \mapsto 0], \mathsf{P})$

- st is a special variable used to store the **adversarial state**.

Finite Distributions

Processes are **probabilistic**:

$\Rightarrow$ The semantics of a process is a distribution of **configurations**.

Discrete distributions.

A **discrete distribution** over a set $\mathbb{S}$ can be written as a formal sum $\sum_{i \in I} a_i \cdot s_i$ where:

$$\text{I is countable} \qquad \sum_{i \in I} a_i = 1 \qquad a_i \geq 0 \ \text{ for any } i \in I$$

Examples

- $\frac{1}{3} \cdot \texttt{"foo"} + \frac{2}{3} \cdot \texttt{"bar"}$
- $\frac{1}{2} \cdot 0 + \frac{1}{2} \cdot 1$ (unbiased coin flip).

Process: Concrete Semantics

A **trace** $\mathtt{tr}$ is a sequence of **observable actions** $\alpha_1, \ldots, \alpha_n$. The trace $\mathtt{tr}$ represents a protocol/adversary **interaction scenario**.

$$\alpha ::= \mathbf{in}(\mathsf{c}) \mid \mathbf{out}(\mathsf{c}) \qquad \mathtt{tr} ::= \epsilon \mid \alpha \mid \mathtt{tr}, \mathtt{tr} \qquad \text{(where } \mathsf{c} \in \mathcal{C}\text{)}$$

The **concrete semantics** is given by the relation (see next slides):

$$(\phi, \sigma, \mathsf{P}) \xRightarrow[\mathbb{L},\mathcal{A},\eta]{\mathtt{tr}} \sum_{i \in I} a_i \cdot (\phi_i, \sigma_i, \mathsf{P}_i)$$

Adversary. $\mathcal{A}$ is a **stateful**, **probabilistic** and **poly-time** program.

Notations. We omit $\mathcal{A}, \mathbb{L}, \eta$ when they are fixed or clear from context. Further, we write $\Rightarrow$ instead of $\xRightarrow{\epsilon}$.

Process: Concrete Semantics

Structural rules:

$$\frac{}{(\phi, \sigma, \mathbf{null} \mid \mathsf{P}) \Rightarrow (\phi, \sigma, \mathsf{P})} \qquad \frac{}{(\phi, \sigma, \mathsf{P}) \Rightarrow (\phi, \sigma, \mathsf{P}') \text{ when } \mathsf{P} \approx_{\mathsf{AC}} \mathsf{P}'}$$

where $\approx_{\mathsf{AC}}$ is the small congruence relation over processes which contains:

- **commutativity** $\mathsf{P}_0 \mid \mathsf{P}_1 \approx_{\mathsf{AC}} \mathsf{P}_1 \mid \mathsf{P}_0$;
- **associativity** $(\mathsf{P}_0 \mid \mathsf{P}_1) \mid \mathsf{P}_2 \approx_{\mathsf{AC}} \mathsf{P}_0 \mid (\mathsf{P}_1 \mid \mathsf{P}_2)$;
- α-**renaming** $\nu\, \mathsf{n}_0.\, \mathsf{P} \approx_{\mathsf{AC}} \nu\, \mathsf{n}_1.\, \mathsf{P}[\mathsf{n}_0 \mapsto \mathsf{n}_1]$
  (same for inputs $\mathbf{in}(\mathsf{c}, \mathsf{x}).\, \mathsf{P}$).

Process: Concrete Semantics

Branching rules:

$$\frac{\langle\!| b |\!\rangle_{\mathbb{L}}^{\eta,\sigma} = 1}{(\phi, \sigma, \text{if } b \text{ then } \mathsf{P} \mid \mathsf{P}') \Rightarrow (\phi, \sigma, \mathsf{P} \mid \mathsf{P}')}$$

$$\frac{\langle\!| b |\!\rangle_{\mathbb{L}}^{\eta,\sigma} = 0}{(\phi, \sigma, \text{if } b \text{ then } \mathsf{P} \mid \mathsf{P}') \Rightarrow (\phi, \sigma, \mathsf{P}')}$$

Output rules:

$$\frac{\phi' = (\phi, \langle\!| t |\!\rangle_{\mathbb{L}}^{\eta,\sigma})}{\big(\phi, \sigma, (\mathbf{out}(\mathsf{c}, t); \mathsf{P}) \mid \mathsf{P}'\big) \xRightarrow{\mathbf{out}(\mathsf{c})} (\phi', \sigma, \mathsf{P} \mid \mathsf{P}')}$$

$$\frac{\begin{array}{c}\phi' = (\phi, \mathsf{error}) \\ \mathsf{c} \notin \mathsf{chans}(\mathsf{P}) \text{ or } \mathsf{P} = (\mathbf{in}(\mathsf{c}, \mathsf{x}).\, \mathsf{P}_0)\end{array}}{\big(\phi, \sigma, \mathsf{P}\big) \xRightarrow{\mathbf{out}(\mathsf{c})} (\phi', \sigma, \mathsf{P})}$$

Process: Concrete Semantics

New rule:

$$\frac{\mathsf{n} \notin \mathsf{dom}(\sigma)}{\left(\phi, \sigma, (\nu\, \mathsf{n}; \mathsf{P}) \mid \mathsf{P}'\right) \Rightarrow \sum_{|w|=\eta} \frac{1}{2^\eta} \cdot \left(\phi, \sigma[\mathsf{n} \mapsto w], \mathsf{P} \mid \mathsf{P}'\right)}$$

Input rule:

$$\frac{\mathsf{x} \notin \mathsf{dom}(\sigma) \qquad \mathcal{A}(1^\eta, \phi, \sigma[\mathsf{st}]) = \sum_i a_i \cdot (w_i, s_i)}{\left(\phi, \sigma, (\mathbf{in}(\mathsf{c}, \mathsf{x}); \mathsf{P}) \mid \mathsf{P}'\right) \xRightarrow{\mathbf{in}(\mathsf{c})} \sum_i a_i \cdot \left(\phi', \sigma[\mathsf{x} \mapsto w_i, \mathsf{st} \mapsto s_i], \mathsf{P} \mid \mathsf{P}'\right)}$$

$$\frac{\mathsf{c} \notin \mathsf{chans}(\mathsf{P}) \text{ or } \mathsf{P} = (\mathbf{out}(\mathsf{c}, t).\, \mathsf{P}_0) \qquad \mathcal{A}(1^\eta, \phi, \sigma[\mathsf{st}]) = \sum_i a_i \cdot (w_i, s_i)}{\left(\phi, \sigma, \mathsf{P}\right) \xRightarrow{\mathbf{in}(\mathsf{c})} \sum_i a_i \cdot \left(\phi', \sigma[\mathsf{st} \mapsto s_i], \mathsf{P}\right)}$$

Remark: $\mathsf{dom}(f) \stackrel{\text{def}}{=} \{x \mid f(x) \text{ defined}\}$ is the domain of the partial function f.

Process: Concrete Semantics

Transitivity rule:

$$\frac{(\phi, \sigma, \mathsf{P}) \xRightarrow{\texttt{tr}_0} \sum_i a_i \cdot (\phi_i, \sigma_i, \mathsf{P}_i) \qquad \forall i.\, (\phi_i, \sigma_i, \mathsf{P}_i) \xRightarrow{\texttt{tr}_1} \sum_j b_{i,j} \cdot (\phi_{i,j}, \sigma_{i,j}, \mathsf{P}_{i,j})}{(\phi, \sigma, \mathsf{P}) \xRightarrow{\texttt{tr}_0, \texttt{tr}_1} \sum_{i,j} a_i \cdot b_{i,j} \cdot (\phi_{i,j}, \sigma_{i,j}, \mathsf{P}_{i,j})}$$

Process: Concrete Semantics

The relation $\Rightarrow$ is **non-deterministic**.

Still, thanks to the **restrictions** on processes, different non-deterministic choices yield **identical distributions** on frames:

$$
\begin{array}{rl}
& (\phi, \sigma, \mathsf{P}) \xRightarrow{\mathtt{tr}} \sum_i a_i \cdot (\phi_0^i, \sigma_0^i, \mathsf{P}_0^i) \\
\wedge & (\phi, \sigma, \mathsf{P}) \xRightarrow{\mathtt{tr}} \sum_j b_j \cdot (\phi_1^j, \sigma_1^j, \mathsf{P}_1^j) \\
\Rightarrow & \sum_i a_i \cdot \phi_0^i = \sum_j b_j \cdot \phi_1^j
\end{array}
$$

Thus, given a process P and a trace $\mathtt{tr}$, if:

$$(\epsilon, [\mathsf{st} \mapsto 0], \mathsf{P}) \xRightarrow[\mathbb{L},\mathcal{A},\eta]{\mathtt{tr}} \sum_i a_i \cdot (\phi_i, \sigma_i, \mathsf{P}_i)$$

then the **distribution** $\sum_i a_i \cdot \phi_i$ is the **concrete execution** of P over $\mathtt{tr}$, denoted $\text{c-frame}^{\eta}_{\mathbb{L},\mathcal{A}}(P, \mathtt{tr})$.

A Motivating Example

Toy Protocol

We consider a toy protocol as a motivating example:

$$\begin{array}{lll} \mathsf{A} : \nu\, \mathsf{n}. & \mathbf{out}(\mathtt{A}, \{\mathsf{n}\}_{\mathsf{k}}).\ \mathbf{out}(\mathtt{A}, \mathsf{n}) \\ \mathsf{B} : \mathbf{in}(\mathtt{B}, \mathsf{x}). & \text{if } \mathrm{dec}(\mathsf{x}, \mathsf{k}) \neq \perp \text{ then} \\ & \mathbf{in}(\mathtt{B}, \mathsf{y}).\ \mathbf{out}(\mathtt{B}, \mathrm{dec}(\mathsf{x}, \mathsf{k}) = \mathsf{y}) \end{array}$$

Security property.

B outputs false if A does not send its second message.

Assumptions. (informally)

We assume that the *symmetric encryption* $\{\cdot\}_{\cdot}$ is a secure AE:

- **Integrity:** if $\mathrm{dec}(m, \mathsf{k}) \neq \perp$ then m is an **honest encryption**.
- **Confidentiality:** $\{m_0\}_{\mathsf{k}}$ and $\{m_1\}_{\mathsf{k}}$ are **indistinguishable**.
  (when m_0 and m_1 are of the same length.)

Security of our Toy Protocol

$$\begin{array}{lll} \mathsf{A} : \nu\, \mathsf{n}. & \mathbf{out}(\mathtt{A}, \{\mathsf{n}\}_{\mathsf{k}}).\ \mathbf{out}(\mathtt{A}, \mathsf{n}) \\ \mathsf{B} : \mathbf{in}(\mathtt{B}, \mathsf{x}). & \text{if } \mathsf{dec}(\mathsf{x}, \mathsf{k}) \neq \bot \text{ then} \\ & \mathbf{in}(\mathtt{B}, \mathsf{y}).\ \mathbf{out}(\mathtt{B}, \mathsf{dec}(\mathsf{x}, \mathsf{k}) = \mathsf{y}) \end{array}$$

When executing the trace

$$\mathbf{out}(\mathtt{A}), \mathbf{in}(\mathtt{B}), \mathbf{in}(\mathtt{B}), \mathbf{out}(\mathtt{B}),$$

the adversary sees the messages:

$$\{\mathsf{n}\}_{\mathsf{k}}, \text{if } \left(\mathsf{dec}(\mathbf{att}_1(\Phi), \mathsf{k}) \neq \bot\right) \text{ then } \left(\mathsf{dec}(\mathbf{att}_1(\Phi), \mathsf{k}) = \mathbf{att}_2(\Phi)\right)$$

where $\Phi = \{\mathsf{n}\}_{\mathsf{k}}$, and $\mathbf{att}_1$, $\mathbf{att}_2$ represents, resp., the first and second inputs to B chosen by the adversary.

Security of our Toy Protocol

Our toy protocol is **secure** if the following **indistinguishability** holds:

$$
\begin{aligned}
& \{\mathsf{n}\}_{\mathsf{k}}, \text{if } \big(\mathrm{dec}(\mathbf{att}_1(\Phi), \mathsf{k}) \neq \perp\big) \text{ then } \big(\mathrm{dec}(\mathbf{att}_1(\Phi), \mathsf{k}) = \mathbf{att}_2(\Phi)\big) \\
\approx\ & \{\mathsf{n}\}_{\mathsf{k}}, \text{if } \big(\mathrm{dec}(\mathbf{att}_1(\Phi), \mathsf{k}) \neq \perp\big) \text{ then false}
\end{aligned}
$$

Informal Security Proof

Using the **integrity** of the encryption, we have:

$$\big(\mathrm{dec}(\mathbf{att}_1(\Phi), \mathsf{k}) \neq \bot\big) \Leftrightarrow \big(\mathbf{att}_1(\Phi) = \{\mathsf{n}\}_{\mathsf{k}}\big).$$

Thus:

$$\begin{aligned}
& \{\mathsf{n}\}_{\mathsf{k}}, \text{if } \big(\mathrm{dec}(\mathbf{att}_1(\Phi), \mathsf{k}) \neq \bot\big) && \text{then } \big(\mathrm{dec}(\mathbf{att}_1(\Phi), \mathsf{k}) = \mathbf{att}_2(\Phi)\big) \\
\approx\ & \{\mathsf{n}\}_{\mathsf{k}}, \text{if } \big(\mathbf{att}_1(\Phi) = \{\mathsf{n}\}_{\mathsf{k}}\big) && \text{then } \big(\mathrm{dec}(\mathbf{att}_1(\Phi), \mathsf{k}) = \mathbf{att}_2(\Phi)\big) \\
\approx\ & \{\mathsf{n}\}_{\mathsf{k}}, \text{if } \big(\mathbf{att}_1(\Phi) = \{\mathsf{n}\}_{\mathsf{k}}\big) && \text{then } \big(\mathsf{n} = \mathbf{att}_2(\Phi)\big)
\end{aligned}$$

Informal Security Proof

Using the **integrity** of the encryption, we have:

$$\big(\mathrm{dec}(\mathbf{att}_1(\Phi), \mathrm{k}) \neq \bot\big) \Leftrightarrow \big(\mathbf{att}_1(\Phi) = \{\mathrm{n}\}_{\mathrm{k}}\big).$$

Thus:

$$\begin{aligned}
& \{\mathrm{n}\}_{\mathrm{k}}, \text{if } \big(\mathrm{dec}(\mathbf{att}_1(\Phi), \mathrm{k}) \neq \bot\big) \text{ then } \big(\mathrm{dec}(\mathbf{att}_1(\Phi), \mathrm{k}) = \mathbf{att}_2(\Phi)\big) \\
\approx\ & \{\mathrm{n}\}_{\mathrm{k}}, \text{if } \big(\mathbf{att}_1(\Phi) = \{\mathrm{n}\}_{\mathrm{k}}\big) \text{ then } \big(\mathrm{dec}(\mathbf{att}_1(\Phi), \mathrm{k}) = \mathbf{att}_2(\Phi)\big) \\
\approx\ & \{\mathrm{n}\}_{\mathrm{k}}, \text{if } \big(\mathbf{att}_1(\Phi) = \{\mathrm{n}\}_{\mathrm{k}}\big) \text{ then } \big(\mathrm{n} = \mathbf{att}_2(\Phi)\big)
\end{aligned}$$

Using the **confidentiality** of the encryption, we can replace $\{\mathrm{n}\}_{\mathrm{k}}$ by $\{0^\eta\}_{\mathrm{k}}$:

$$\approx\ \{0^\eta\}_{\mathrm{k}}, \text{ if } \big(\mathbf{att}_1(\{0^\eta\}_{\mathrm{k}}) = \{0^\eta\}_{\mathrm{k}}\big) \text{ then } \big(\mathrm{n} = \mathbf{att}_2(\{0^\eta\}_{\mathrm{k}})\big)$$

Informal Security Proof

Using the **integrity** of the encryption, we have:

$$\big(\mathrm{dec}(\mathbf{att}_1(\Phi), \mathsf{k}) \neq \bot\big) \Leftrightarrow \big(\mathbf{att}_1(\Phi) = \{\mathsf{n}\}_{\mathsf{k}}\big).$$

Thus:

$$\begin{aligned}
& \{\mathsf{n}\}_{\mathsf{k}}, \text{if } \big(\mathrm{dec}(\mathbf{att}_1(\Phi), \mathsf{k}) \neq \bot\big) \text{ then } \big(\mathrm{dec}(\mathbf{att}_1(\Phi), \mathsf{k}) = \mathbf{att}_2(\Phi)\big) \\
\approx\ & \{\mathsf{n}\}_{\mathsf{k}}, \text{if } \big(\mathbf{att}_1(\Phi) = \{\mathsf{n}\}_{\mathsf{k}}\big) \quad \text{ then } \big(\mathrm{dec}(\mathbf{att}_1(\Phi), \mathsf{k}) = \mathbf{att}_2(\Phi)\big) \\
\approx\ & \{\mathsf{n}\}_{\mathsf{k}}, \text{if } \big(\mathbf{att}_1(\Phi) = \{\mathsf{n}\}_{\mathsf{k}}\big) \quad \text{ then } \big(\mathsf{n} = \mathbf{att}_2(\Phi)\big)
\end{aligned}$$

Using the **confidentiality** of the encryption, we can replace $\{\mathsf{n}\}_{\mathsf{k}}$ by $\{0^\eta\}_{\mathsf{k}}$:

$$\approx\ \{0^\eta\}_{\mathsf{k}}, \text{ if } \big(\mathbf{att}_1(\{0^\eta\}_{\mathsf{k}}) = \{0^\eta\}_{\mathsf{k}}\big) \text{ then } \big(\mathsf{n} = \mathbf{att}_2(\{0^\eta\}_{\mathsf{k}})\big)$$

By **probabilistic independence**: $\mathsf{n} \neq \mathbf{att}_2(\{0^\eta\}_{\mathsf{k}})$ (overwhelmingly):

$$\approx\ \{0^\eta\}_{\mathsf{k}}, \text{if } \big(\mathbf{att}_1(\{0^\eta\}_{\mathsf{k}}) = \{0^\eta\}_{\mathsf{k}}\big) \text{ then false}$$

We conclude by a similar proof in the other direction.

Verification of Cryptographic Protocols

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

To build a verification framework from what we just did, we need to:

- Model the **adversary/protocol interaction** as symbolic terms.
- Express the **security property** Φ using a logical formula.
- Capture the **cryptographic arguments** $\models$ as reasoning rules.

Symbolic Protocol Execution

Symbolic Protocol Execution

Goal: obtain **symbolic representations** of protocol executions that:

- **faithfully** model the protocol semantics;
- are amenable to **formal reasoning**.

How? Use the same **techniques** as in our motivating example.

- **1. Explicit probabilistic dependencies:**
  randomness is sampled **eagerly** and not **lazily**.
- **2. Pure encoding of the adversary:**
  no adversarial state.

Symbolic Execution: Explicit Probabilistic Dependencies

1. Explicit probabilistic dependencies.

Key idea.

Sample randomness beforehand (eagerly), and retrieve it as needed.

Concretely, we rewrite a process P by **moving randomness early**:

$$\begin{aligned} \mathsf{P} = \mathsf{E}_1 \mid \cdots \mid \mathsf{E}_l \quad &\Leftrightarrow \quad (\nu\, \vec{\mathsf{n}_1}.\, \mathsf{E}'_1) \mid \cdots \mid (\nu\, \vec{\mathsf{n}_l}.\, \mathsf{E}'_l) \\ &\Leftrightarrow \quad \nu\, \vec{\mathsf{n}_1}, \ldots, \vec{\mathsf{n}_l}.\, (\mathsf{E}'_1 \mid \cdots \mid \mathsf{E}'_l) \end{aligned}$$

where $(\nu\, \vec{\mathsf{n}_i}.\, \mathsf{E}'_i)$ is E_i with all random samplings moved at the beginning. and names are distincts:

$$\forall i \neq j.\, \vec{\mathsf{n}_i} \cap \vec{\mathsf{n}_j} = \emptyset.$$

Symbolic Execution: Explicit Probabilistic Dependencies

More precisely, $\mathsf{E}_i \rightsquigarrow^! (\nu\, \vec{\mathsf{n}_i}.\, \mathsf{E}'_i)$ where $\rightsquigarrow$ is defined by the rules:

$$(\mathsf{E}_0.\, \nu\, \mathsf{n}.\, \mathsf{E}_1) \rightsquigarrow \nu\, \mathsf{n}.\, (\mathsf{E}_0.\, \mathsf{E}_1) \qquad (\mathsf{E}_0 \mid \nu\, \mathsf{n}.\, \mathsf{E}_1) \rightsquigarrow \nu\, \mathsf{n}.\, (\mathsf{E}_0 \mid \mathsf{E}_1)$$

$$\text{if } b \text{ then } \nu\, \mathsf{n}.\, \mathsf{E} \rightsquigarrow \nu\, \mathsf{n}.\, \text{if } b \text{ then } \mathsf{E}$$

where $\mathsf{n} \not\in \mathsf{fv}(\mathsf{E}_0)$ and $\mathsf{n} \not\in \mathsf{fv}(b)$.

Recall that process are taken modulo α-renaming.

Notations. $\mathsf{E}_0 \rightsquigarrow^! \mathsf{E}_1$ iff. $\mathsf{E}_0 \rightsquigarrow^* \mathsf{E}_1$ and $\mathsf{E}_1 \not\rightsquigarrow \mathsf{E}'$ for all E'.
$\mathsf{fv}(\mathsf{E})$ and $\mathsf{fv}(t)$ are the **free variables** of, resp., E and t.

Symbolic Executions: Pure Encoding of the Adversary

2. Pure encoding of the adversary.

We need a **deterministic** and **stateless** representation $\mathcal{A}_{\mathsf{p}}$ of the adversary $\mathcal{A}$.

- **deterministic**: sample the adversary's randomness ρ_{a} **eagerly** and pass it as an **explicit argument**.
- **stateless**: $\mathcal{A}_{\mathsf{p}}$ recomputes $\mathcal{A}$'s state each time it is called.
  This exploits determinism, and requires us to provide the full history each time we call $\mathcal{A}_p$.

Symbolic Executions: Pure Encoding of the Adversary

Informally, we want that:

$$
\begin{array}{rcccc}
 & \mathcal{A}(w_1) & , \ \ldots \ , & \mathcal{A}(w_n) \\
=_{\mathsf{distr.}} & \mathcal{A}_{\mathsf{p}}(w_1, \rho_{\mathsf{a}}) & , \ \ldots \ , & \mathcal{A}_{\mathsf{p}}(w_1, \ldots, w_n, \rho_{\mathsf{a}})
\end{array}
$$

where ρ_{a} is a *long enough* sequence of bits sampled **independently uniformly at random**.

We call $\mathcal{A}_{\mathsf{p}}$ a **pure representation** of $\mathcal{A}$.

We do not detail it, but $\mathcal{A}_{\mathsf{p}}$ can be systematically built from $\mathcal{A}$.

Symbolic Protocol Execution

Terms

Terms

To define our **symbolic execution rules**, we need to **extend** our set of **term symbols** $\mathcal{S} = \mathcal{N} \uplus \mathcal{X} \uplus \mathcal{F} \uplus \mathcal{G}$:

- **Variables** $\mathcal{X}$.
- **Function symbols** $\mathcal{F}$.
- **Names** $\mathcal{N}$.
- **Adversarial function symbols** $\mathcal{G}$, of any arity.

Changes.

- Names are no longer represented by variables in $\mathcal{X}$, but by special symbols in $\mathcal{N}$ with a tailored semantics (presented later).
  (For the sack of simplicity, we asks that all names are of type `message`.)
- Adversarial function symbols $\mathcal{G}$ are used to represent calls to $\mathcal{A}_{\mathsf{p}}$.

Adversarial function symbols

More precisely, if:

- there has already been n **outputs**, represented by the terms $t_1, \dots, t_n$;
- and we are doing the j-th **input** since the protocol started;

then the input bitstring is represented by:

$$\mathbf{att}_j(t_1, \dots, t_n)$$

where $\mathbf{att}_j \in \mathcal{G}$ is an **adversarial** function symbol of arity n.

💡 *j allows to have different values for consecutive inputs.*

Symbolic Protocol Execution

Symbolic Rules

Symbolic Executions

We describe a **systematic method** to compute, given a **process** P and a **trace** `tr` of **observable actions**, the **terms** representing the **outputted messages** during the execution of P over `tr`.

This is the **symbolic execution** of P over `tr`.

We deal with the protocol **randomness** and **adversarial inputs** using the two techniques we just saw.

Symbolic Configuration

Symbolic configuration

A **symbolic configuration** is a tuple $(\Phi; \lambda; j; \Pi_1, \dots, \Pi_l)$ where:

- Φ is a sequence of terms (in $\mathcal{T}(\mathcal{S})$).
- $\lambda : \mathcal{X} \hookrightarrow \mathcal{T}(\mathcal{S})$ is a symbolic valuation.
- $j \in \mathbb{N}$.
- for every i, $\Pi_i = (P_i, b_i)$ where P_i is a protocol and b_i is a boolean term.

Symbolic Configuration: Intuition

In a symbolic configuration $(\Phi; \lambda; j; \Pi_1, \ldots, \Pi_l)$:

- Φ is the **symbolic frame**, i.e. the sequence of terms outputted since the execution started.
- λ **records inputs**, it maps input variable to their corresponding term.
- j **counts the number of inputs** since the execution started.
- $\Pi = (P, b)$ **represents** P if b is true (and is **null** otherwise).
  Using this interpretation, $\Pi_1, \ldots, \Pi_l$ is the **current process**.

Initial configuration: $(\epsilon; \emptyset; 0; (P, \top))$

Symbolic Execution: Branching and Input Rules

Rule for protocol branching:

$$\begin{array}{rr} & (\Phi; \lambda; j; (\text{if } b_0 \text{ then } \mathsf{P}, b_1), \Pi_1, \ldots, \Pi_l) \\ \hookrightarrow & (\Phi; \lambda; j; (\mathsf{P}, b_0 \wedge b_1), \Pi_1, \ldots, \Pi_l) \end{array}$$

Rules for inputs:

$$\begin{array}{rl} & (\Phi; \lambda; j; (\mathbf{in}(\mathsf{c}, \mathsf{x}).\mathsf{P}, b), \Pi_1, \ldots, \Pi_l) \\ \overset{\mathbf{in}(\mathsf{c})}{\hookrightarrow} & (\Phi; \lambda[\mathsf{x} \mapsto \mathbf{att}_j(\Phi)]; j+1; (\mathsf{P}, b), \Pi_1, \ldots, \Pi_l) \end{array} \qquad (\mathsf{x} \notin \mathrm{dom}(\lambda))$$

$$(\Phi; \lambda; j; \Pi_1, \ldots, \Pi_l) \overset{\mathbf{in}(\mathsf{c})}{\hookrightarrow} (\Phi; \lambda; j+1; \Pi_1, \ldots, \Pi_l) \qquad (\dagger)$$

where $(\dagger)$: $\mathsf{c} \notin \mathrm{chans}(\Pi_1, \ldots, \Pi_l)$ or $\exists i$ s.t. Π_i starts with $\mathbf{out}(\mathsf{c}, \cdot)$.

Symbolic Execution: Output Rule

Rules for outputs:

$$(\Phi; \lambda; j; (\mathbf{out}(\mathsf{c}, t).\mathsf{P}, b), \Pi_1, \ldots, \Pi_l)$$
$$\overset{\mathbf{out}(\mathsf{c})}{\hookrightarrow} \ (\Phi, (\text{if } b \text{ then } t)\lambda; \lambda; j; (\mathsf{P}, b), \Pi_1, \ldots, \Pi_l)$$

$$(\Phi; \lambda; j; \Pi_1, \ldots, \Pi_l) \overset{\mathbf{out}(\mathsf{c})}{\hookrightarrow} (\Phi, \mathtt{error}; \lambda; j; \Pi_1, \ldots, \Pi_l) \qquad (\ddagger)$$

where $(\ddagger)$: $\mathsf{c} \notin \text{chans}(\Pi_1, \ldots, \Pi_l)$ or $\exists i$ s.t. Π_i starts with $\mathbf{out}(\mathsf{c}, \cdot)$.

The input and output rules make sense because we restrict ourselves to elementary processes with distinct channels.

Given a process P (without ν) and a trace $\mathtt{tr}$, if:

$$(\epsilon; \emptyset; 0; (\mathsf{P}, \top)) \xhookrightarrow{\mathtt{tr}} (\Phi; \_; \_; \_)$$

then Φ is the **symbolic execution** of P over $\mathtt{tr}$, denoted $\mathsf{frame}(\mathsf{P}, \mathtt{tr})$.

Handling the ν construct.
If P contains ν, we compute P_0 s.t. $\mathsf{P} \rightsquigarrow^! \nu\, \vec{\mathsf{n}}.\, \mathsf{P}_0$ with $\vec{\mathsf{n}} \in \mathcal{N}$, and then symbolically execute P_0.

Symbolic Execution: Soundness

Claim (informal).

The symbolic execution is **sound** w.r.t. the concrete semantics.
More precisely, for every library $\mathbb{L}$, adversary $\mathcal{A}$, and security parameter η:

$$\mathsf{c\text{-}frame}^{\eta}_{\mathbb{L},\mathcal{A}}(\mathsf{P}, \mathtt{tr}) =_{\mathsf{distr.}} [\![\mathsf{frame}(\mathsf{P}, \mathtt{tr})]\!]^{\eta,\rho}_{\mathbb{L}[\mathbf{att} \mapsto \mathcal{A}_{\mathsf{p}}]}$$

where:

- ρ is sampled uniformly at random among bitstrings of sufficient length.
- $\mathcal{A}_{\mathsf{p}}$ is a pure representation of $\mathcal{A}$.

Remark: $[\![t]\!]^{\eta,\rho}_{\mathbb{M}}$ is the semantics of the terms coming from the symbolic execution, which we define in the next section.

Symbolic Execution: Exercises

Exercise

What are all the **possible symbolic executions** of the following protocols?

$\mathbf{in}(\mathtt{c}, \mathtt{x}).\,\mathbf{out}(\mathtt{c}, t)$

$\mathbf{out}(\mathtt{A}, t_1) \mid \mathbf{in}(\mathtt{B}, \mathtt{x}).\,\mathbf{out}(\mathtt{B}, t_2)$

$\text{if } b \text{ then } \mathbf{out}(\mathtt{c}, t_1) \text{ else } \mathbf{out}(\mathtt{c}, t_2)$

$\text{if } b \text{ then } \mathbf{out}(\mathtt{A}, t_1) \text{ else } \mathbf{out}(\mathtt{B}, t_2)$

Exercise

Extend the **symbolic** algorithm with a rule allowing to handle processes with let bindings.

Could the same thing be done for mutable, inter-process, state?

Semantics of Terms

Semantics of Terms

We showed how to represent **protocol execution**, on some fixed trace of observables `tr`, as a **sequence of terms**.

Intuitively, the terms corresponds to **PTIME-computable bitstring distributions**.

Example

If $\langle \_ \, , \, \_ \rangle$ is the concatenation, and samplings are done uniformly at random among bitstrings of length $\eta \in \mathbb{N}$, then:

$$\nu\, \mathsf{n}_0, \nu\, \mathsf{n}_1, \mathbf{out}(\mathsf{c}, \langle \mathsf{n}_0 \, , \, \langle 00 \, , \, \mathsf{n}_1 \rangle\rangle) \quad \text{yields} \quad \langle \mathsf{n}_0 \, , \, \langle 00 \, , \, \mathsf{n}_1 \rangle\rangle$$

which represent a distribution over bitstrings of length $2 \cdot \eta + 2$, where all bits are sampled uniformly and independently, except for the bits at positions η and $\eta + 1$, which are always 0.

Semantics of Terms

We interpret $t \in \mathcal{T}(\mathcal{S})$ as a **Probabilistic Polynomial-time Turing machine** (PPTM), with:

- a **working tape** (also used as input tape);
- two **read-only tapes** $\rho = (\rho_\mathsf{a}, \rho_\mathsf{h})$ for adversary and honest randomness.

We let $\mathcal{D}$ be the set of such machines.

💡 *The machine must be polynomial in the size of its input on the working tape only.*

Terms Interpretation

The **interpretation** $[\![t]\!]_{\mathbb{M}} \in \mathcal{D}$ of a term t is parameterized by a **model** $\mathbb{M}$ which provides:

- the set of random tapes $\mathbb{T}_{\mathbb{M},\eta} = \mathbb{T}^{\mathsf{a}}_{\mathbb{M},\eta} \times \mathbb{T}^{\mathsf{h}}_{\mathbb{M},\eta}$, where $\mathbb{T}^{\mathsf{a}}_{\mathbb{M},\eta}$ and $\mathbb{T}^{\mathsf{h}}_{\mathbb{M},\eta}$ are **finite** same-length set of bit-strings.
  We equip it with the **uniform** probability measure.
  ($\mathbb{T}^{\mathsf{a}}_{\mathbb{M},\eta}$ for the adversary, $\mathbb{T}^{\mathsf{h}}_{\mathbb{M},\eta}$ for honest functions)
- the semantics $(\!|\cdot|\!)_{\mathbb{M}}$ of **symbols** in $\mathcal{S}$ (details on next slides).
  (This extends the interpretation $(\!|\cdot|\!)_{\mathbb{L}}$ of symbols by a library $\mathbb{L}$.)

We may omit $\mathbb{M}$ when it is clear from context.

We define the machine $[\![t]\!]_{\mathbb{M}} \in \mathcal{D}$, by defining its behavior $[\![t]\!]^{\eta,\rho}_{\mathbb{M}}$ for every $\eta \in \mathbb{N}$ and pairs of random tapes $\rho = (\rho_{\mathsf{a}}, \rho_{\mathsf{h}}) \in \mathbb{T}_{\mathbb{M},\eta}$.

Terms Interpretation: Function Symbols

Function symbols interpretations is just **composition**.

For **function symbols** in $f \in \mathcal{F}$, we simply apply $(\!|f|\!)_{\mathbb{M}}$:

$$[\![f(t_1, \ldots, t_n)]\!]_{\mathbb{M}}^{\eta,\rho} \stackrel{\text{def}}{=} (\!|f|\!)_{\mathbb{M}}(1^\eta, [\![t_1]\!]_{\mathbb{M}}^{\eta,\rho}, \ldots, [\![t_n]\!]_{\mathbb{M}}^{\eta,\rho})$$

Adversarial function symbols $g \in \mathcal{G}$ also have access to ρ_{a}:

$$[\![g(t_1, \ldots, t_n)]\!]_{\mathbb{M}}^{\eta,\rho} \stackrel{\text{def}}{=} (\!|g|\!)_{\mathbb{M}}(1^\eta, [\![t_1]\!]_{\mathbb{M}}^{\eta,\rho}, \ldots, [\![t_n]\!]_{\mathbb{M}}^{\eta,\rho}, \rho_{\mathsf{a}})$$

Restrictions. $(\!|f|\!)_{\mathbb{M}}$ and $(\!|g|\!)_{\mathbb{M}}$ are:

- PTIME-computable;
- **deterministic** (all randomness must come explicitly, from ρ).

Terms Interpretation: Variables and Names

The interpretation $(\!|x|\!)_{\mathbb{M}}$ of a **variable** $x \in \mathcal{X}$ is an **arbitrary machine** in $\mathcal{D}$. Then:

$$[\![\mathsf{x}]\!]_{\mathbb{M}}^{\eta,\rho} \stackrel{\text{def}}{=} (\!|x|\!)_{\mathbb{M}}(1^\eta, \rho).$$

Names $\mathsf{n} \in \mathcal{G}$ are interpreted as **uniform random samplings** among bitstrings of length η, extracted from ρ_{h}:

$$[\![\mathsf{n}]\!]_{\mathbb{M}}^{\eta,\rho} \stackrel{\text{def}}{=} (\!|\mathsf{n}|\!)_{\mathbb{M}}(1^\eta, \rho_{\mathsf{h}})$$

For every pair of different names $\mathsf{n}_0, \mathsf{n}_1$, we require that $(\!|\mathsf{n}_0|\!)_{\mathbb{M}}$ and $(\!|\mathsf{n}_1|\!)_{\mathbb{M}}$ extracts disjoint parts of ρ_{h}.

Hence different names are ***independent*** *random samplings.*

Terms Interpretation: Names

Examples

- If (n, n_0 : `message`) then:

$$
\begin{aligned}
[\![\mathsf{n}]\!]^\eta &=_{\text{distr.}} \text{sample } w \text{ in } \{0,1\}^\eta \\
[\![(\mathsf{n}, \mathsf{n}_0)]\!]^\eta &=_{\text{distr.}} \text{sample } w \text{ in } \{0,1\}^\eta \\
&\qquad\quad \text{sample } w' \text{ in } \{0,1\}^\eta \textbf{ independently} \\
&\qquad\quad \text{build } (w, w') \\
[\![(\mathsf{n}, \mathsf{n})]\!]^\eta &=_{\text{distr.}} \text{sample } w \text{ in } \{0,1\}^\eta \\
&\qquad\quad \text{build } (w, w)
\end{aligned}
$$

Indeed:

$$[\![(\mathsf{n}, \mathsf{n})]\!]^{\eta,\rho} = ([\![\mathsf{n}]\!]^{\eta,\rho}, [\![\mathsf{n}]\!]^{\eta,\rho}) = (w, w) \quad \text{(where } w = (\!|\mathsf{n}|\!)(1^\eta, \rho_{\mathsf{h}}))$$

Terms Interpretation: Modeling and Randomness

$\neq$ in **how randomness** is sampled:

- **In the "real-world"**, the adversary $\mathcal{A}$ samples randomness on-the-fly, as needed.
  $\Rightarrow$ possibly $P(\eta)$ random bits, where P is the (polynomial) running-time of $\mathcal{A}$.
- **In the logic**, we restrict $\mathbb{T}_{\mathbb{M},\eta} = \mathbb{T}^{\mathsf{a}}_{\mathbb{M},\eta} \times \mathbb{T}^{\mathsf{h}}_{\mathbb{M},\eta}$ to be finite and fixed by $\mathbb{M}$.
  $\Rightarrow$ all randomness sampled eagerly according to $\mathbb{M}$, independently of the adversary $\mathcal{A}$.

This $\neq$ of behaviors is not an issue, i.e. the logic can **soundly model** real-world adversaries:

- Indeed, for any adversary $\mathcal{A}$, there exists a model $\mathbb{M}$ with enough randomness.

Outline

Verification of Cryptographic Protocols

$$\forall \mathcal{A} \in \mathcal{C}.\ (\mathcal{A} \parallel \mathsf{P}) \models \Phi$$

✓ • Model the **adversary/protocol interaction** as **symbolic terms**.

- Express the **security property** Φ using a **logical formula**.
- Capture the **cryptographic arguments** $\models$ as **reasoning rules**.

References i

[1] D. Adrian, K. Bhargavan, Z. Durumeric, P. Gaudry, M. Green, J. A. Halderman, N. Heninger, D. Springall, E. Thomé, L. Valenta, B. VanderSloot, E. Wustrow, S. Zanella-Béguelin, and P. Zimmermann.
Imperfect forward secrecy: how diffie-hellman fails in practice.
Commun. ACM, 62(1):106–114, 2019.

[2] N. Aviram, S. Schinzel, J. Somorovsky, N. Heninger, M. Dankel, J. Steube, L. Valenta, D. Adrian, J. A. Halderman, V. Dukhovni, E. Käsper, S. Cohney, S. Engels, C. Paar, and Y. Shavitt.
DROWN: breaking TLS using sslv2.
In *USENIX Security Symposium*, pages 689–706. USENIX Association, 2016.

[3] D. Baelde, S. Delaune, C. Jacomme, A. Koutsos, and J. Lallemand.
The squirrel prover and its logic.
ACM SIGLOG News, 11(2):62–83, 2024.

[4] D. Baelde, S. Delaune, C. Jacomme, A. Koutsos, and S. Moreau.
An interactive prover for protocol verification in the computational model.
In *SP*, pages 537–554. IEEE, 2021.

[5] G. Bana and H. Comon-Lundh.
A computationally complete symbolic attacker for equivalence properties.
In *CCS*, pages 609–620. ACM, 2014.

[6] G. Barthe, F. Dupressoir, B. Grégoire, C. Kunz, B. Schmidt, and P. Strub.
Easycrypt: A tutorial.
In *FOSAD*, volume 8604 of *Lecture Notes in Computer Science*, pages 146–166. Springer, 2013.

References ii

[7] G. Barthe, B. Grégoire, and S. Zanella-Béguelin.
Probabilistic relational hoare logics for computer-aided security proofs.
In *MPC*, volume 7342 of *Lecture Notes in Computer Science*, pages 1–6. Springer, 2012.

[8] D. A. Basin, R. Sasse, and J. Toro-Pozo.
Card brand mixup attack: Bypassing the PIN in non-visa cards by using them for visa transactions.
In *USENIX Security Symposium*, pages 179–194. USENIX Association, 2021.

[9] D. A. Basin, R. Sasse, and J. Toro-Pozo.
The EMV standard: Break, fix, verify.
In *SP*, pages 1766–1781. IEEE, 2021.

[10] K. Bhargavan, A. Delignat-Lavaud, C. Fournet, A. Pironti, and P. Strub.
Triple handshakes and cookie cutters: Breaking and fixing authentication over TLS.
In *IEEE Symposium on Security and Privacy*, pages 98–113. IEEE Computer Society, 2014.

[11] B. Blanchet.
A computationally sound mechanized prover for security protocols.
IEEE Trans. Dependable Secur. Comput., 5(4):193–207, 2008.

[12] M. Vanhoef and F. Piessens.
Key reinstallation attacks: Forcing nonce reuse in WPA2.
In *CCS*, pages 1313–1328. ACM, 2017.

[13] M. Vanhoef and F. Piessens.
Release the kraken: New kracks in the 802.11 standard.
In *CCS*, pages 299–314. ACM, 2018.