On Proving the Security of Message Authentication Codes Using λ**BLL** and Hoare Logic

Michele Dinelli

Supervisor: Prof. Ugo Dal Lago
Co-Supervisor: Dr. Rémy Cerda

Department of Computer Science and Engineering
University of Bologna

March 26, 2026

Outline

Cryptography: Symbolic vs Computational Model

- **Symbolic Model**
  - Treats cryptographic primitives as **black-box** operations.
  - Enables **automated**, scalable verification of security protocols.

Cryptography: Symbolic vs Computational Model

- **Symbolic Model**
  - Treats cryptographic primitives as **black-box** operations.
  - Enables **automated**, scalable verification of security protocols.
- **Computational Model**
  - Adveraries are Probabilistic Polynomial-Time Turing Machines (**PPTM**).
  - Keys, messages and ciphertexts are **bitstrings** and cryptographic operations are **random** functions on bitstrings.
  - Security properties are defined in terms of the **probability** and **computational complexity** of the attackers.

The Formalization Challenge

- **Symbolic Model**
  - The application of classic verification methodologies in the symbolic model is **natural**.
  - Many formal tools have been developed such as `Proverif`, `TAMARIN` and `DEEPSEC`.
- **Computational Model**
  - Involve probabilities, adversarial interaction and explicit reasoning about resource bounds and computational efficiency.

The Formalization Challenge

- **Symbolic Model**
  - The application of classic verification methodologies in the symbolic model is **natural**.
  - Many formal tools have been developed such as `Proverif`, `TAMARIN` and `DEEPSEC`.
- **Computational Model**
  - Involve probabilities, adversarial interaction and explicit reasoning about resource bounds and computational efficiency.
  - It is **harder**!

The Formalization Challenge

- **Symbolic Model**
  - The application of classic verification methodologies in the symbolic model is **natural**.
  - Many formal tools have been developed such as `Proverif`, `TAMARIN` and `DEEPSEC`.
- **Computational Model**
  - Involve probabilities, adversarial interaction and explicit reasoning about resource bounds and computational efficiency.
  - It is **harder**!
  - But still formal verification frameworks are available, including `EasyCrypt`, `Squirrel`, and `CryptoVerif`.

λBLL

Polynomials	$p ::= 1 \mid i \mid p + p \mid p \times p$
Ground types	$G ::= \mathbb{U} \mid \mathbb{B} \mid \mathbb{S}[p]$
Positive types	$P ::= G \mid P \otimes P \mid !_p A$
Types	$A ::= P \mid P \multimap A$
Variable contexts	$\Gamma ::= \varnothing \mid x : P, \Gamma$
Reference contexts	$\Theta ::= \varnothing \mid r : G, \Theta$

λBLL

- Polynomials index resources, they **bound** input sizes, number of oracle queries and overall runtime as functions of the security parameter i.
- Supports references, probabilistic choice and captures the complexity constraints through graded modalities.
- PPT adversaries modelled as **higher-order** terms with oracle access:

$$\vdash_c^{\varnothing} Adv : !_q\big(\mathbb{S}[i] \multimap \mathbb{S}[i]\big) \multimap (\mathbb{S}[i] \otimes \mathbb{S}[i])$$

- Security proofs become **equational reasoning** between **λBLL** terms, proved sound via logical relations.

$$\texttt{return}\,\mathbf{f} \sim \texttt{let}\, y = M\, \texttt{in}\, \texttt{let}\, x = \texttt{random}\, \texttt{in}\, \texttt{eq}(x, y) \quad \text{(randF)}$$

Message Authentication Codes (MAC)

MAC ensures message authentication: each party can verify that a received message was sent by the claimed sender.

Construction: PRF-Induced MAC

- *Gen* takes the security parameter 1^n as input and outputs a secret key k.

Message Authentication Codes (MAC)

MAC ensures message authentication: each party can verify that a received message was sent by the claimed sender.

Construction: PRF-Induced MAC

- Gen takes the security parameter 1^n as input and outputs a secret key k.
- Mac takes a key k and message m as input and outputs a tag $t = F_k(m)$, where F is a pseudorandom function.

Message Authentication Codes (MAC)

MAC ensures message authentication: each party can verify that a received message was sent by the claimed sender.

Construction: PRF-Induced MAC

- Gen takes the security parameter 1^n as input and outputs a secret key k.
- Mac takes a key k and message m as input and outputs a tag $t = F_k(m)$, where F is a pseudorandom function.
- $Vrfy$ takes a key k, message m, and tag t as input and outputs a bit b. The tag is valid if $Vrfy_k(m, Mac_k(m)) = 1$.

Message Authentication Codes (MAC)

MAC ensures message authentication: each party can verify that a received message was sent by the claimed sender.

Construction: PRF-Induced MAC

- Gen takes the security parameter 1^n as input and outputs a secret key k.
- Mac takes a key k and message m as input and outputs a tag $t = F_k(m)$, where F is a pseudorandom function.
- $Vrfy$ takes a key k, message m, and tag t as input and outputs a bit b. The tag is valid if $Vrfy_k(m, Mac_k(m)) = 1$.

Theorem: PRF-Induced MAC Security

If F is a pseudorandom function, then Construction above is a secure fixed-length MAC for messages of length n.

MacForge Experiment

$$
\begin{aligned}
&\texttt{MacForge}_{A,\Pi}(n): \\
&\quad k \leftarrow Gen(1^n) \\
&\quad (m,t) \leftarrow A^{Mac_k(\cdot)}(1^n) \\
&\quad \mathbb{Q} \leftarrow \{m \mid A \text{ queries } Mac_k(m)\} \\
&\quad \texttt{return } (m \notin \mathbb{Q} \wedge Vrfy_k(m,t) = 1)
\end{aligned}
$$

MacForge Experiment

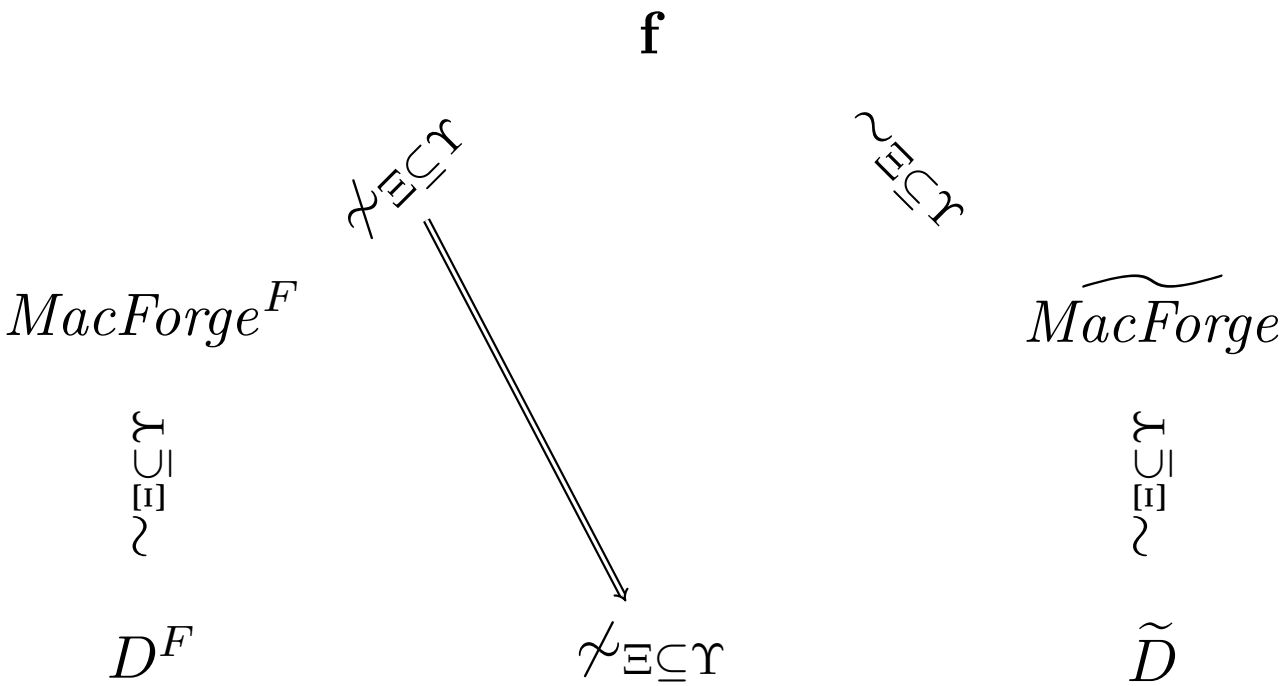
$$
\begin{aligned}
&\texttt{MacForge}_{A,\Pi}(n): \\
&\quad k \leftarrow Gen(1^n) \\
&\quad (m,t) \leftarrow A^{Mac_k(\cdot)}(1^n) \\
&\quad \mathbb{Q} \leftarrow \{m \mid A \text{ queries } Mac_k(m)\} \\
&\quad \texttt{return } (m \notin \mathbb{Q} \wedge Vrfy_k(m,t) = 1)
\end{aligned}
$$

Defintion: MAC Security

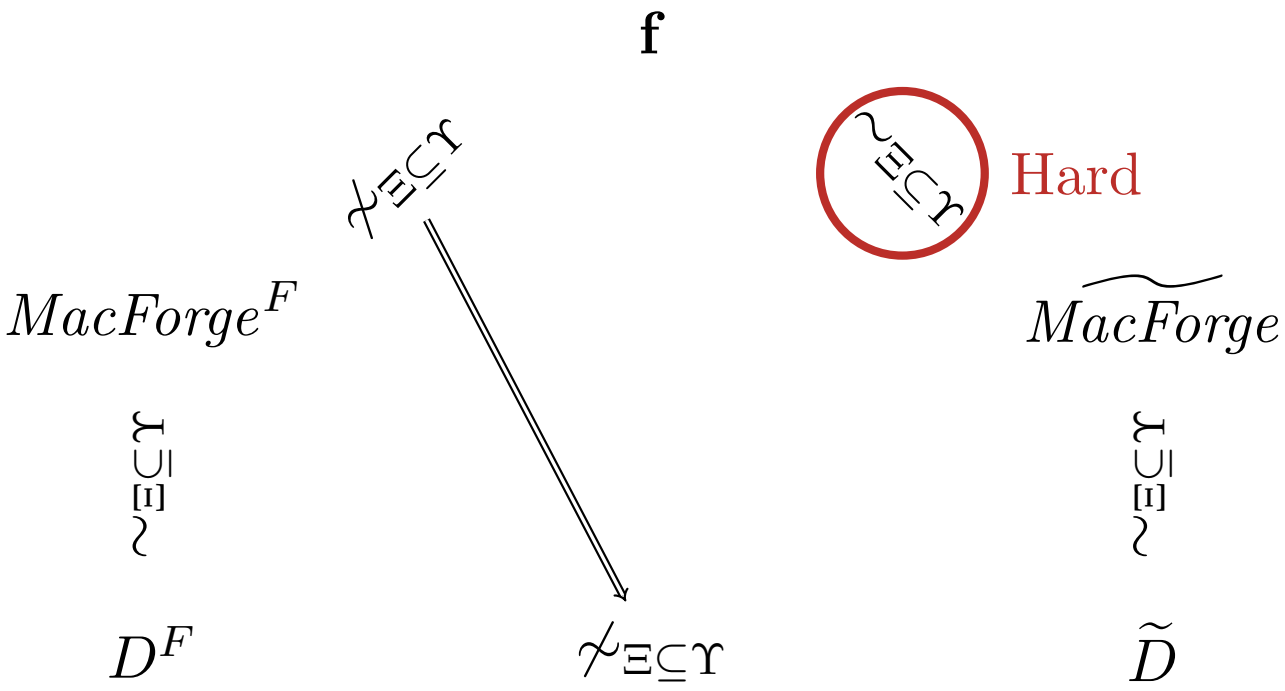
A MAC Π *is secure if for all PPT adversaries* A*, there is a negligible function* ϵ *such that:*

$$\Pr\left[\texttt{MacForge}_{A,\Pi}(n) = 1\right] \leq \epsilon(n)$$

Security Proof of MAC in λBLL



Security Proof of MAC in λBLL



Hoare Logic

- Programs behaviour expressed by **partial correctness specifications**, written as triples of the form

$$\{P\}\, e\, \{Q\}.$$

- If execution of e starts in a program state satisfying P, then every **terminating** execution of e produces a state satisfying Q.
- **Propositions** used in Hoare triples are generated by the following grammar:

$$\Phi ::= \phi \mid \Phi \wedge \Phi \mid \Phi \vee \Phi$$

Hoare Logic

$$\frac{r : G \in \Theta}{\Gamma \vdash_c^\Theta \{P(r)\}\,\mathtt{get}\,r\,\{\lambda x.P(x)\} : G}\ \text{H-GET} \qquad \frac{\Gamma \vdash_v^\Theta Z : G \quad r : G \in \Theta}{\Gamma \vdash_c^\Theta \{P[Z/r]\}\,\mathtt{set}\,r\,Z\,\{P\} : \mathbb{U}}\ \text{H-SET}$$

$$\frac{\Gamma \vdash_c^\Theta \{P\}M\{\lambda v.Q(v)\} : A \quad Q \models P}{p * \Gamma \vdash_v^\Theta\ !M : !_p^{P,Q} A}\ \text{H-BANG}$$

$$\frac{\Gamma \vdash_c^\Theta \{P\}M\{\lambda f.Q(f)\} : A \underset{R,S}{\multimap} B \quad \Delta \vdash_v^\Theta Z : A \quad Q \models R}{\Gamma \boxplus \Delta \vdash_c^\Theta \{P\}\,MZ\,\{\lambda v.S(v)\} : B}\ \text{H-APP}$$

$$\frac{\Gamma \vdash_c^\Theta \{P\}\,M\,\{\lambda v.Q(v)\} : A \quad \Delta, x : A \vdash_c^\Theta \{Q(x)\}\,N\,\{S\} : B}{\Gamma \boxplus \Delta \vdash_c^\Theta \{P\}\,\mathtt{let}\,x = M\,\mathtt{in}\,N\,\{S\} : B}\ \text{H-LET}$$

$$\frac{P' \models P \quad Q \models Q' \quad \Gamma \vdash_c^\Theta \{P\}\,M\,\{\lambda v.Q(v)\} : A}{\Gamma \vdash_c^\Theta \{P'\}\,M\,\{\lambda v.Q'(v)\} : A}\ \text{H-CONS}$$

Formalization of MAC Security in λBLL

$$
\begin{array}{l}
MacForge \triangleq \\
\quad \texttt{let}\, k = Gen \star\, \texttt{in} \\
\quad \texttt{let}\, o = Oracle\, k\, \texttt{in} \\
\quad \texttt{let}\, \langle m, tag \rangle = Adv\, o\, \texttt{in} \\
\quad \texttt{let}\, queries = \texttt{get}\, Q\, \texttt{in} \\
\quad \texttt{let}\, c = \texttt{contains}(queries, m)\, \texttt{in} \\
\quad \texttt{if}\, c\, \texttt{then}\, \texttt{return}\, \texttt{f} \\
\quad \texttt{else}\, \texttt{let}\, x = \texttt{der}(o)\, m\, \texttt{in}\, \texttt{eq}(x, tag)
\end{array}
$$

MAC Algorithms as λBLL-Terms

$\widetilde{Gen} \triangleq$

$\lambda u.$

$\texttt{let}\, q = \texttt{initQ}\, \texttt{in}$

$\texttt{set}\, Q\, q;$

$\texttt{let}\, tab = \texttt{initTable}\, \texttt{in}$

$\texttt{set}\, rand\, tab;$

$\texttt{let}\, z = \texttt{zero}\, \texttt{in}$

$\texttt{return}\, z$

$\widetilde{Mac} \triangleq$

$\lambda e.$

$\texttt{let}\, \langle k, m \rangle = e\, \texttt{in}$

$\texttt{let}\, tab = \texttt{get}\, rand\, \texttt{in}$

$\texttt{let}\, b = \texttt{isdefined}(tab, m)\, \texttt{in}$

$\texttt{if}\, b\, \texttt{then}$

$\quad \texttt{let}\, x = \texttt{getValue}(tab, m)\, \texttt{in}$

$\quad \texttt{return}\, x$

$\texttt{else}$

$\quad \texttt{let}\, x = \texttt{random}\, \texttt{in}$

$\quad \texttt{let}\, newtab = \texttt{modify}(tab, m, x)\, \texttt{in}$

$\quad \texttt{set}\, rand\, newtab;$

$\quad \texttt{return}\, x$

MAC Algorithms as λBLL-Terms

$$\widetilde{Oracle} \triangleq$$
$$\texttt{return}\,(\lambda k.\texttt{return}\,!(\texttt{return}\,\lambda m.$$
$$\texttt{let}\,q = \texttt{get}\,Q\,\texttt{in}$$
$$\texttt{let}\,t = \texttt{contains}(q, m)\,\texttt{in}$$
$$\texttt{if}\,\texttt{not}(t)\,\texttt{then}\,\texttt{set}\,Q\,\texttt{modifyQ}(q, m)$$
$$\texttt{else}\,\texttt{skip};$$
$$\widetilde{Mac}\langle k, m\rangle))$$

$$D \triangleq$$
$$\texttt{return}\,(\lambda o.$$
$$\texttt{let}\,\langle m, tag\rangle = Adv\,o\,\texttt{in}$$
$$\texttt{let}\,queries = \texttt{get}\,Q\,\texttt{in}$$
$$\texttt{let}\,c = \texttt{contains}(queries, m)\,\texttt{in}$$
$$\texttt{if}\,c\,\texttt{then}\,\texttt{return}\,\texttt{f}$$
$$\texttt{else}\,\texttt{let}\,x = \texttt{der}(o)\,m\,\texttt{in}\,\texttt{eq}(x, tag))$$

$$D^F \triangleq \texttt{let}\,k = Gen^F\,\star\,\texttt{in}\,\texttt{let}\,o = M^F\,k\,\texttt{in}\,\texttt{let}\,e = MacOracle\,o\,\texttt{in}\,D\,e$$
$$\widetilde{D} \triangleq \texttt{let}\,k = \widetilde{Gen}\,\star\,\texttt{in}\,\texttt{let}\,o = \widetilde{M}k\,\texttt{in}\,\texttt{let}\,e = MacOracle\,o\,\texttt{in}\,D\,e$$

From $\widetilde{D}$ to $\widetilde{MacForge}$

Lemma 1

$$\texttt{let}\, o = \widetilde{M}\, k\, \texttt{in}\, MacOracle\, o \sim_{\Xi \subseteq \Upsilon} \widetilde{Oracle}\, k$$

$$\begin{aligned}
\widetilde{D} \quad &\triangleq \quad \texttt{let}\, k = \widetilde{Gen} \star\, \texttt{in}\, \texttt{let}\, o = \widetilde{M} k\, \texttt{in}\, \texttt{let}\, e = MacOracle\, o\, \texttt{in}\, D\, e \\
&\overset{\text{letAss}}{\sim_{\Xi \subseteq \Upsilon}} \quad \texttt{let}\, k = \widetilde{Gen} \star\, \texttt{in}\, \texttt{let}\, e = (\texttt{let}\, o = \widetilde{M}\, k\, \texttt{in}\, MacOracle\, o)\, \texttt{in}\, D\, e \\
&\overset{\text{Lemma 1}}{\sim_{\Xi \subseteq \Upsilon}} \quad \begin{array}{l} \texttt{let}\, k = \widetilde{Gen} \star\, \texttt{in} \\ \texttt{let}\, e = \widetilde{Oracle}\, k\, \texttt{in} \\ D\, e \end{array}
\end{aligned}$$

From D^F to $MacForge^F$

Lemma 2

$$\texttt{let}\, o = M^F\, \texttt{in}\, MacOracle\, o \sim_{\Xi \subseteq \Upsilon} Oracle^F\, k$$

$$\begin{aligned}
D^F \quad &\triangleq \quad \texttt{let}\, k = Gen^F\, \star\, \texttt{in}\, \texttt{let}\, o = M^F\, k\, \texttt{in}\, \texttt{let}\, e = MacOracle\, o\, \texttt{in}\, D\, e \\
&\overset{\text{letAss}}{\sim_{\Xi \subseteq \Upsilon}} \quad \texttt{let}\, k = Gen^F\, \star\, \texttt{in}\, \texttt{let}\, e = (\texttt{let}\, o = M^F\, k\, \texttt{in}\, MacOracle\, o)\, \texttt{in}\, D\, e \\
&\overset{\text{Lemma 2}}{\sim_{\Xi \subseteq \Upsilon}} \quad \begin{array}{l} \texttt{let}\, k = Gen^F\, \star\, \texttt{in} \\ \texttt{let}\, e = Oracle^F\, k\, \texttt{in} \\ D\, e \end{array}
\end{aligned}$$

From $\widetilde{MacForge}$ to $\mathbf{f}$ - The Role of Assertions

$\texttt{let}\, k = \widetilde{Gen}\, \star\, \texttt{in}$

$\ldots$

$\texttt{let}\, queries = \texttt{get}\, Q\, \texttt{in}$

$\texttt{let}\, c = \texttt{contains}(queries, m)\, \texttt{in}$

$\texttt{if}\, c\, \texttt{then return f}$

$\texttt{else}$

$\quad \texttt{let}\, q = \texttt{get}\, Q\, \texttt{in}$

$\quad \texttt{let}\, t = \texttt{contains}(q, m)\, \texttt{in}$

$\quad \{\texttt{coherent}(q, rand) \wedge (q = Q) \wedge (m \notin q) \wedge t = (m \in q)\}$

$\quad \texttt{if not}(t)\, \texttt{then set}\, Q\, \texttt{modifyQ}(q, m)$

$\quad \texttt{else skip};$

$\ldots$

From $\widetilde{MacForge}$ to $\mathbf{f}$ - The Role of Assertions

$\texttt{let}\, k = \widetilde{Gen}\, \star\, \texttt{in}$

$\ldots$

$\texttt{let}\, queries = \texttt{get}\, Q\, \texttt{in}$

$\texttt{let}\, c = \texttt{contains}(queries, m)\, \texttt{in}$

$\texttt{if}\, c\, \texttt{then return}\, \mathbf{f}$

$\texttt{else}$

$\quad \texttt{let}\, q = \texttt{get}\, Q\, \texttt{in}$

$\quad \texttt{let}\, t = \texttt{contains}(q, m)\, \texttt{in}$

$\quad \{\texttt{coherent}(q, rand) \wedge (q = Q) \wedge (m \notin q) \wedge t = (m \in q)\}$

$\quad\quad \Longrightarrow\ \{t = \mathbf{f}\}$

$\quad \texttt{if not}(t)\, \texttt{then set}\, Q\, \texttt{modifyQ}(q, m)$

$\quad \texttt{else skip};$

$\ldots$

From $\widetilde{MacForge}$ to $\mathbf{f}$ - The Role of Assertions

$\texttt{let}\, k = \widetilde{Gen}\, \star\, \texttt{in}$

$\ldots$

$\texttt{let}\, queries = \texttt{get}\, Q\, \texttt{in}$

$\texttt{if}\, c\, \texttt{then return}\, \mathbf{f}$

$\texttt{else}$

$\ldots$

$\quad \texttt{let}\, tab = \texttt{get}\, rand\, \texttt{in}$

$\quad \texttt{let}\, b = \texttt{isdefined}(tab, m)\, \texttt{in}$

$\quad \{\texttt{coherent}(q, rand) \wedge (Q = q \cup \{m\}) \wedge (m \notin q)$

$\qquad \wedge\, (tab = rand) \wedge (b = (m \in tab))\}$

$\quad \texttt{if}\, b\, \texttt{then}$

$\qquad \texttt{let}\, x = \texttt{getValue}(tab, m)\, \texttt{in}$

$\qquad \texttt{eq}(x, tag)$

$\quad \texttt{else}$

$\ldots$

From $\widetilde{MacForge}$ to $\mathbf{f}$ - The Role of Assertions

$\texttt{let}\ k = \widetilde{Gen}\ \star\ \texttt{in}$

$\ldots$

$\texttt{let}\ queries = \texttt{get}\ Q\ \texttt{in}$

$\texttt{if}\ c\ \texttt{then return}\ \mathbf{f}$

$\texttt{else}$

$\ldots$

$\quad \texttt{let}\ tab = \texttt{get}\ rand\ \texttt{in}$

$\quad \texttt{let}\ b = \texttt{isdefined}(tab, m)\ \texttt{in}$

$\quad \{\texttt{coherent}(q, rand) \wedge (Q = q \cup \{m\}) \wedge (m \notin q)$

$\qquad \wedge\ (tab = rand) \wedge (b = (m \in tab))\}$

$\qquad\qquad \Longrightarrow\ \{b = \mathbf{f}\}$

$\quad \texttt{if}\ b\ \texttt{then}$

$\qquad \texttt{let}\ x = \texttt{getValue}(tab, m)\ \texttt{in}$

$\qquad \texttt{eq}(x, tag)$

$\quad \texttt{else}$

$\ldots$

From $\widetilde{MacForge}$ to $\mathbf{f}$

$$\texttt{let}\ \langle m, tag\rangle = Adv\ O\ \texttt{in}$$
$$\texttt{let}\ queries = \texttt{get}\ Q\ \texttt{in}$$
$$\texttt{let}\ c = \texttt{contains}(queries, m)\ \texttt{in}$$
$$\texttt{if}\ c\ \texttt{then return}\ \mathbf{f}$$
$$\texttt{else}$$
$$\quad \texttt{let}\ x = \texttt{random}\ \texttt{in}$$
$$\quad \texttt{eq}(x, tag)$$

$$\overset{\text{randF}}{\sim_{\Xi \subseteq \Upsilon}}$$

$$\texttt{let}\ \langle m, tag\rangle = Adv\ O\ \texttt{in}$$
$$\texttt{let}\ queries = \texttt{get}\ Q\ \texttt{in}$$
$$\texttt{let}\ c = \texttt{contains}(queries, m)\ \texttt{in}$$
$$\texttt{if}\ c\ \texttt{then return}\ \mathbf{f}$$
$$\texttt{else return}\ \mathbf{f}$$

Conclusions

Summary

- **λBLL** is a higher order calculus with probabilistic effects and references. A powerful formal framework in the computational model.
- Hoare extension enables reasoning about state and invariants.
- MAC security proof have been carried out in a purely **equational** way with the help of assertions and an extended type systems.

Future work

- Formalize the soundness of the Hoare extension.