Automated Security Proofs for Almost-Universal Hash for MAC verification

Martin Gagné

Saarland University, Germany

September 10, 2013
ESORICS 2013, RHUL, Egham, U.K

Joint work with Pascal Lafourcade and Yassine Lakhnech

Why make automated provers for crytpo?

Why make automated provers for crytpo?

- People make mistakes

Why make automated provers for crytpo?

- People make mistakes
- Mistakes in security proof of crypto protocols are bad

Why make automated provers for crytpo?

- People make mistakes
- Mistakes in security proof of crypto protocols are bad
- Reinforce our trust in their security

Why make automated provers for crytpo?

- People make mistakes
- Mistakes in security proof of crypto protocols are bad
- Reinforce our trust in their security
- Can lead to automated protocol synthesis

Message Authentication Code

Purpose: ensure integrity and authenticity of messages

What is a MAC?

- $MAC : key \times \{0,1\}^* \rightarrow tag$
- With (m, tag), receiver computes $tag' = MAC(k, m)$ and accepts the message as authentic if $tag = tag'$

Examples: CBC-MAC, HMAC, PMAC, VMAC, UMAC ...

Message Authentication Code

Purpose: ensure integrity and authenticity of messages

What is a MAC?

- $MAC : key \times \{0,1\}^* \rightarrow tag$
- With (m, tag), receiver computes $tag' = MAC(k, m)$ and accepts the message as authentic if $tag = tag'$

Examples: CBC-MAC, HMAC, PMAC, VMAC, UMAC ...

Applications

- Authenticity
- IND-CCA security (encrypt-then-mac construction)
- Building block for many other cryptographic protocols

MAC Security

Exp_{MAC}:

- Sample $k \xleftarrow{R} \{0,1\}^\eta$.
- $(m^*, tag) \xleftarrow{R} \mathcal{A}^{MAC(k,\cdot)}(\eta)$
- if $MAC(k, m^*) = tag$ and $\mathcal{A}$ never queried m^* to its $MAC(k, \cdot)$ oracle, return 1, else return 0

Definition

$\textsc{Adv}_{\mathcal{A}}^{UNF}(\eta) = Pr[Exp_{MAC} = 1]$

A MAC is **existentially unforgeable** if $\textsc{Adv}_{\mathcal{A}}^{UNF}(\eta)$ is negligible.

Our Approach

Automatically proving security of cryptographic primitives

1. Modeling security properties
2. Defining a language
3. Building an Hoare Logic
4. Proving the security

Our Approach

Automatically proving security of cryptographic primitives

1. Modeling security properties
2. Defining a language
3. Building an Hoare Logic
4. Proving the security

For

Our Approach

Automatically proving security of cryptographic primitives

1. Modeling security properties
2. Defining a language
3. Building an Hoare Logic
4. Proving the security

For

- Asymmetric Encryption [CDELL08]

Our Approach

Automatically proving security of cryptographic primitives

1. Modeling security properties
2. Defining a language
3. Building an Hoare Logic
4. Proving the security

For

- Asymmetric Encryption [CDELL08]
- Symmetric Encryption Modes [GLLS09]

Our Approach

Automatically proving security of cryptographic primitives

1. Modeling security properties
2. Defining a language
3. Building an Hoare Logic
4. Proving the security

For

- Asymmetric Encryption [CDELL08]
- Symmetric Encryption Modes [GLLS09]
- Block cipher and hash-based MACs [ESORICS2013]

Hoare Logic [H69]

Set of rules (R_i) : $\{P\}$ cmd $\{Q\}$

Hoare Logic [H69]

Set of rules (R_i) : $\{P\}$ cmd $\{Q\}$

$$c_1$$
$$c_2$$
$$\vdots$$
$$c_n$$

Hoare Logic [H69]

Set of rules (R_i) : $\{P\}$ cmd $\{Q\}$

$$\begin{array}{ll} \{P_0\} & c_1 \\ & c_2 \\ & \vdots \\ & c_n \end{array}$$

Hoare Logic [H69]

Set of rules (R_i) : $\{P\}$ cmd $\{Q\}$

$$\begin{array}{ll} \{P_0\} & c_1 \\ & c_2 \\ & \vdots \\ & c_n \ \{Q_n\} \ ? \end{array}$$

Hoare Logic [H69]

Set of rules (R_i) : $\{P\}$ cmd $\{Q\}$

$$
\begin{array}{ll}
(R_5)\{P_0\} & c_1 \ \{Q_0\} \\
 & c_2 \\
 & \vdots \\
 & c_n \ \{Q_n\} \ ?
\end{array}
$$

Hoare Logic [H69]

Set of rules (R_i) : $\{P\}$ cmd $\{Q\}$

$(R_5)\{P_0\}\ c_1\ \{Q_0\}$
$(R_2)\{P_1\}\ c_2\ \{Q_2\}$, where $P_1 \subseteq Q_0$
$\vdots$
$c_n\ \{Q_n\}$?

Hoare Logic [H69]

Set of rules (R_i) : $\{P\}$ cmd $\{Q\}$

$(R_5)\{P_0\}\ c_1\ \{Q_0\}$
$(R_2)\{P_1\}\ c_2\ \{Q_2\}$, where $P_1 \subseteq Q_0$
$\vdots$
$(R_8)\{P_n\}\ c_n\ \{Q_n\}$?

Challenges with MACs

- Security property harder to model
- MACs are deterministic
- Common prefixes cause repeated queries to the block cipher

Challenges with MACs

- Security property harder to model
- MACs are deterministic
- Common prefixes cause repeated queries to the block cipher

$\Rightarrow$ We need fundamentally new trick

$\Rightarrow$ Analyze collisions using 2 simultaneous executions

Outline

Outline

Security Proofs of MACs

Usual method for proving MAC security

- Pseudo-random functions are good MACs

$\Rightarrow$ prove that the compressing part of the MAC is an **almost-universal hash function**

$\Rightarrow$ combine that almost-universal hash with a mixing step to get a PRF

Our Strategy

Two Step Strategy

- Hoare logic to prove 'front-end' is almost-universal
  - empty list of block cipher queries at the beginning
  - consider two simultaneous executions of the code
  - examine probability of collisions between intermediate values
- List of possible 'mixing steps'

DMAC (CBC-MAC variant)

Example

$c_1 := m_1;$
for $i = 2$ to n do:
  $z_i := c_{i-1} \oplus m_i$
  $c_i := \mathcal{E}(z_i);$
$tag := \mathcal{E}'(c_n);$

HMAC

Example

$$
\begin{aligned}
& z_1 := k \| m_1; \\
& c_1 := \mathcal{H}(z_1); \\
& \text{for } i = 2 \text{ to } n \text{ do:;} \\
& \qquad z_i := c_{i-1} \| m_i \\
& \qquad c_i := \mathcal{H}(z_i) \\
& z' := k' \| c_n; \\
& tag := \mathcal{H}(z');
\end{aligned}
$$

Almost-universal hash function [CW79]

Definition

A hash function family $\{H_k\}_{k\in\{0,1\}^\eta}$ is an **almost-universal hash function** family if for any two messages $m_0, m_1 \in \{0,1\}^*$,

$$\Pr[H_k(m_0) = H_k(m_1)] \text{ is negligible,}$$

where the probability is taken over the choice of the key.

Grammar

$$\begin{array}{lll} \mathsf{c} & ::= & x := \rho^i(y) \mid x := \mathcal{E}(y) \mid x := \mathcal{H}(y) \mid x := y \oplus z \mid x := y \| z \mid \\ & & x := y \mid \text{for } \; x = i \; \text{ to } \; j \; \text{ do: } \; c_x \mid c_1 ; c_2 \end{array}$$

Generic Hash

Definition

A generic hash function M is represented by
$Hash_M(m_1|\dots|m_p, c_p)$: **var** $\vec{x}$; cmd

$$
\begin{aligned}
&Hash_{CBC}(m_1|m_2|m_3, c_3):\\
&\textbf{var}\ z_1, z_2, z_3;\\
&c_1 := \mathcal{E}(m_1);\\
&z_2 := c_1 \oplus m_2;\\
&c_2 := \mathcal{E}(z_2);\\
&z_3 := c_2 \oplus m_3;\\
&c_3 := \mathcal{E}(z_3);
\end{aligned}
$$

Semantics of the Programming Language

$$
\begin{aligned}
&[\![x := \mathcal{E}(y)]\!](S, S', \mathcal{E}, \mathcal{H}, \mathcal{L}_{\mathcal{E}}, \mathcal{L}_{\mathcal{H}}) = \\
&\quad (S\{x \mapsto \mathcal{E}(S(y))\}, S'\{x \mapsto \mathcal{E}(S'(y))\}, \mathcal{E}, \mathcal{H}, \mathcal{L}_{\mathcal{E}} \cup \{S(y), S'(y)\}, \mathcal{L}_{\mathcal{H}}) \\
&[\![x := \mathcal{H}(y)]\!](S, S', \mathcal{E}, \mathcal{H}, \mathcal{L}_{\mathcal{E}}, \mathcal{L}_{\mathcal{H}}) = \\
&\quad (S\{x \mapsto \mathcal{H}(S(y))\}, S'\{x \mapsto \mathcal{H}(S'(y))\}, \mathcal{E}, \mathcal{H}, \mathcal{L}_{\mathcal{E}}, \mathcal{L}_{\mathcal{H}} \cup \{S(y), S'(y)\}) \\
&[\![x := y]\!](S, S', \mathcal{E}, \mathcal{H}, \mathcal{L}_{\mathcal{E}}, \mathcal{L}_{\mathcal{H}}) = (S\{x \mapsto S(y)\}, S'\{x \mapsto S'(y)\}, \mathcal{E}, \mathcal{H}, \mathcal{L}_{\mathcal{E}}, \mathcal{L}_{\mathcal{H}}) \\
&[\![x := y \oplus z]\!](S, S', \mathcal{E}, \mathcal{H}, \mathcal{L}_{\mathcal{E}}, \mathcal{L}_{\mathcal{H}}) = \\
&\quad (S\{x \mapsto S(y) \oplus S(z)\}, S'\{x \mapsto S'(y) \oplus S'(z)\}, \mathcal{E}, \mathcal{H}, \mathcal{L}_{\mathcal{E}}, \mathcal{L}_{\mathcal{H}}) \\
&[\![x := y||z]\!](S, S', \mathcal{E}, \mathcal{H}, \mathcal{L}_{\mathcal{E}}, \mathcal{L}_{\mathcal{H}}) = \\
&\quad (S\{x \mapsto S(y)||S(z)\}, S'\{x \mapsto S'(y)||S'(z)\}, \mathcal{E}, \mathcal{H}, \mathcal{L}_{\mathcal{E}}, \mathcal{L}_{\mathcal{H}}) \\
&[\![x := \rho(i, y)]\!](S, S', \mathcal{E}, \mathcal{H}, \mathcal{L}_{\mathcal{E}}, \mathcal{L}_{\mathcal{H}}) = \\
&\quad (S\{x \mapsto \rho(i, S(y))\}, S'\{x \mapsto \rho(i, S'(y))\}, \mathcal{E}, \mathcal{H}, \mathcal{L}_{\mathcal{E}}, \mathcal{L}_{\mathcal{H}}) \\
&[\![\text{for } l = p \text{ to } q \text{ do: } [\text{cmd}_l]]\!]\gamma = \begin{cases} [\![\text{cmd}_q]\!] \circ [\![\text{cmd}_{q-1}]\!] \circ \ldots \circ [\![\text{cmd}_p]\!]\gamma \text{ if } p \leq q \\ \gamma \text{ otherwise} \end{cases} \\
&[\![c_1; c_2]\!] = [\![c_2]\!] \circ [\![c_1]\!]
\end{aligned}
$$

Semantics of the Programming Language

$$
\begin{aligned}
&[\![x := \mathcal{E}(y)]\!](S, S', \mathcal{E}, \mathcal{H}, \mathcal{L}_{\mathcal{E}}, \mathcal{L}_{\mathcal{H}}) = \\
&\quad (S\{x \mapsto \mathcal{E}(S(y))\}, S'\{x \mapsto \mathcal{E}(S'(y))\}, \mathcal{E}, \mathcal{H}, \mathcal{L}_{\mathcal{E}} \cup \{S(y), S'(y)\}, \mathcal{L}_{\mathcal{H}}) \\
&[\![x := \mathcal{H}(y)]\!](S, S', \mathcal{E}, \mathcal{H}, \mathcal{L}_{\mathcal{E}}, \mathcal{L}_{\mathcal{H}}) = \\
&\quad (S\{x \mapsto \mathcal{H}(S(y))\}, S'\{x \mapsto \mathcal{H}(S'(y))\}, \mathcal{E}, \mathcal{H}, \mathcal{L}_{\mathcal{E}}, \mathcal{L}_{\mathcal{H}} \cup \{S(y), S'(y)\}) \\
&[\![x := y]\!](S, S', \mathcal{E}, \mathcal{H}, \mathcal{L}_{\mathcal{E}}, \mathcal{L}_{\mathcal{H}}) = (S\{x \mapsto S(y)\}, S'\{x \mapsto S'(y)\}, \mathcal{E}, \mathcal{H}, \mathcal{L}_{\mathcal{E}}, \mathcal{L}_{\mathcal{H}}) \\
&[\![x := y \oplus z]\!](S, S', \mathcal{E}, \mathcal{H}, \mathcal{L}_{\mathcal{E}}, \mathcal{L}_{\mathcal{H}}) = \\
&\quad (S\{x \mapsto S(y) \oplus S(z)\}, S'\{x \mapsto S'(y) \oplus S'(z)\}, \mathcal{E}, \mathcal{H}, \mathcal{L}_{\mathcal{E}}, \mathcal{L}_{\mathcal{H}}) \\
&[\![x := y||z]\!](S, S', \mathcal{E}, \mathcal{H}, \mathcal{L}_{\mathcal{E}}, \mathcal{L}_{\mathcal{H}}) = \\
&\quad (S\{x \mapsto S(y)||S(z)\}, S'\{x \mapsto S'(y)||S'(z)\}, \mathcal{E}, \mathcal{H}, \mathcal{L}_{\mathcal{E}}, \mathcal{L}_{\mathcal{H}}) \\
&[\![x := \rho(i, y)]\!](S, S', \mathcal{E}, \mathcal{H}, \mathcal{L}_{\mathcal{E}}, \mathcal{L}_{\mathcal{H}}) = \\
&\quad (S\{x \mapsto \rho(i, S(y))\}, S'\{x \mapsto \rho(i, S'(y))\}, \mathcal{E}, \mathcal{H}, \mathcal{L}_{\mathcal{E}}, \mathcal{L}_{\mathcal{H}}) \\
&[\![\text{for } l = p \text{ to } q \text{ do: } [\text{cmd}_l]]\!]\gamma = \begin{cases} [\![\text{cmd}_q]\!] \circ [\![\text{cmd}_{q-1}]\!] \circ \ldots \circ [\![\text{cmd}_p]\!]\gamma \text{ if } p \le q \\ \gamma \text{ otherwise} \end{cases} \\
&[\![c_1; c_2]\!] = [\![c_2]\!] \circ [\![c_1]\!]
\end{aligned}
$$

Predicates

Equal(x, y): the probability that $S(x) \neq S'(y)$ is negligible.

Unequal(x, y): the probability that $S(x) = S'(y)$ is negligible.

E$(\mathcal{E}; x; V)$: the probability that the value of x is either in $\mathcal{L}_{\mathcal{E}}$ or in V is negligible.

H$(\mathcal{H}, x; V)$: the probability that the value of x is either in $\mathcal{L}_{\mathcal{H}}$ or in V is negligible.

Empty: that the probability that $\mathcal{L}_{\mathcal{E}}$ contains an element is negligible.

Indis$(x; V; V')$: the value of x is indistinguishable from a random value given the values of the variables in V in this execution and the values of the variables in V' from the parallel execution.

Proving that Generic Hash is Almost-Universal

Theorem

A generic hash $P(m_1 \| \ldots \| m_n, c : \textbf{var}\ \vec{x}; \mathrm{cmd})$ is an almost-universal hash function if for all n, $\{(init)\}$ cmd $\{UNIV(n)\}$ holds:

$$\left(\bigwedge_{i=1}^{n-1} \mathrm{Unequal}(c_n, c_i) \wedge \bigwedge_{i=1}^{n} \mathrm{Equal}(m_i, m_i)\right) \vee \bigwedge_{i=1}^{n} \mathrm{Unequal}(c_n, c_i)$$

Outline

Some of our 40 Rules

XOR

(X4) $\{\text{Equal}(y, y) \wedge \text{Unequal}(z, z)\}\ x := y \oplus z$
$\{\text{Unequal}(x, x)\}$ if $y \neq z$

Concatenation

(C6) $\{\text{Indis}(y; V, \mathcal{L}_{\mathcal{H}}; V')\}\ x := y \| z\ \{\text{H}(\mathcal{H}, x; V)\}$

Assignment

(A1) $\{true\}\ x := m_i\ \{(\text{Equal}(m_i, m_i) \wedge \text{Equal}(x, x)) \vee$
$\text{Unequal}(x, x)\}$

Some of our 40 Rules

XOR

(X4) $\{\mathsf{Equal}(y,y) \wedge \mathsf{Unequal}(z,z)\}\ x := y \oplus z$
$\{\mathsf{Unequal}(x,x)\}$ if $y \neq z$

Concatenation

(C6) $\{\mathsf{Indis}(y; V, \mathcal{L}_{\mathcal{H}}; V')\}\ x := y \| z\ \{\mathsf{H}(\mathcal{H}, x; V)\}$

Assignment

(A1) $\{true\}\ x := m_i\ \{(\mathsf{Equal}(m_i, m_i) \wedge \mathsf{Equal}(x,x)) \vee$
$\mathsf{Unequal}(x,x)\}$

Some of our 40 Rules

Block Cipher

(B1) $\{\mathrm{Empty}\}\ x := \mathcal{E}(m_i)$
$\{(\mathrm{Unequal}(x,x) \wedge \mathrm{Indis}(x; \mathrm{Var}, \mathcal{L}_{\mathcal{E}}, \mathcal{L}_{\mathcal{H}}; \mathrm{Var})) \vee$
$(\mathrm{Equal}(m_i, m_i) \wedge \mathrm{Equal}(x,x) \wedge$
$\mathrm{Indis}(x; \mathrm{Var}, \mathcal{L}_{\mathcal{E}}, \mathcal{L}_{\mathcal{H}}; \mathrm{Var} - x))\}$

(B2) $\{\mathrm{E}(\mathcal{E}; y; \emptyset) \wedge \mathrm{Unequal}(y,y)\}\ x := \mathcal{E}(y)$
$\{\mathrm{Indis}(x; \mathrm{Var}, \mathcal{L}_{\mathcal{E}}, \mathcal{L}_{\mathcal{H}}; \mathrm{Var})\}$

(B3) $\{\mathrm{E}(\mathcal{E}; y; \emptyset) \wedge \mathrm{Equal}(y,y)\}\ x := \mathcal{E}(y)$
$\{\mathrm{Indis}(x; \mathrm{Var}, \mathcal{L}_{\mathcal{E}}, \mathcal{L}_{\mathcal{H}}; \mathrm{Var} - x) \wedge \mathrm{Equal}(x,x)\}$

Example: CBC-MAC

$$
\begin{array}{lll}
 & (\mathsf{Init}) & \{\mathsf{Empty}\} \\
c_1 := \mathcal{E}(m_1); & (\mathsf{B1}) & \{(\mathsf{Unequal}(c_1, c_1) \wedge \mathsf{Indis}(c_1; \mathsf{Var}, \mathcal{L}_{\mathcal{E}}, \mathcal{L}_{\mathcal{H}}; \mathsf{Var})) \vee \\
 & & \quad (\mathsf{Equal}(m_1, m_1) \wedge \mathsf{Equal}(c_1, c_1) \wedge \\
 & & \qquad \mathsf{Indis}(c_1; \mathsf{Var}, \mathcal{L}_{\mathcal{E}}, \mathcal{L}_{\mathcal{H}}; \mathsf{Var} - c_1))\} \\
z_2 := c_1 \oplus m_2; & (\mathsf{G5})(\mathsf{X2}) & \{(\mathsf{Indis}(c_1; \mathsf{Var} - z_2, \mathcal{L}_{\mathcal{E}}, \mathcal{L}_{\mathcal{H}}; \mathsf{Var}) \wedge \\
 & & \qquad \mathsf{Indis}(z_2; \mathsf{Var} - c_1, \mathcal{L}_{\mathcal{E}}, \mathcal{L}_{\mathcal{H}}; \mathsf{Var})) \vee \\
 & (\mathsf{G1})(\mathsf{X1}) & \quad (\mathsf{Equal}(m_1, m_1) \wedge \mathsf{Indis}(c_1; \mathsf{Var} - z_2, \mathcal{L}_{\mathcal{E}}, \mathcal{L}_{\mathcal{H}}; \mathsf{Var} - c_1 - z_2) \wedge \\
 & & \qquad \mathsf{Unequal}(z_2, z_2) \wedge \mathsf{Indis}(z_2; \mathsf{Var} - c_1, \mathcal{L}_{\mathcal{E}}, \mathcal{L}_{\mathcal{H}}; \mathsf{Var} - c_1 - z_2) \\
 & & \quad (\mathsf{Equal}(m_1, m_1) \wedge \mathsf{Equal}(m_2, m_2) \wedge \mathsf{Equal}(z_2, z_2) \wedge \\
 & & \qquad \mathsf{Indis}(c_1; \mathsf{Var} - z_2, \mathcal{L}_{\mathcal{E}}, \mathcal{L}_{\mathcal{H}}; \mathsf{Var} - c_1 - z_2) \wedge \\
 & & \qquad \mathsf{Indis}(z_2; \mathsf{Var} - c_1, \mathcal{L}_{\mathcal{E}}, \mathcal{L}_{\mathcal{H}}; \mathsf{Var} - c_1 - z_2))\} \\
c_2 := \mathcal{E}(z_2) & (\mathsf{B2})(\mathsf{B4}) & \{(\mathsf{Indis}(c_1; \mathsf{Var} - z_2, \mathcal{L}_{\mathcal{H}}; \mathsf{Var}) \wedge \\
 & & \qquad \mathsf{Indis}(c_2; \mathsf{Var}, \mathcal{L}_{\mathcal{E}}, \mathcal{L}_{\mathcal{H}}; \mathsf{Var})) \vee \\
 & (\mathsf{G1}) & \quad (\mathsf{Equal}(m_1, m_1) \wedge \mathsf{Indis}(c_1; \mathsf{Var} - z_2, \mathcal{L}_{\mathcal{H}}; \mathsf{Var} - c_1 - z_2) \wedge \\
 & & \qquad \mathsf{Indis}(c_2; \mathsf{Var}, \mathcal{L}_{\mathcal{E}}, \mathcal{L}_{\mathcal{H}}; \mathsf{Var})) \vee \\
 & (\mathsf{B3}) & \quad (\mathsf{Equal}(m_1, m_1) \wedge \mathsf{Equal}(m_2, m_2) \wedge \\
 & & \qquad \mathsf{Indis}(c_1; \mathsf{Var} - z_2, \mathcal{L}_{\mathcal{H}}; \mathsf{Var} - c_1 - z_2) \wedge \\
 & & \qquad \mathsf{Indis}(c_2; \mathsf{Var}, \mathcal{L}_{\mathcal{E}}, \mathcal{L}_{\mathcal{H}}; \mathsf{Var} - c_2))\}
\end{array}
$$

Example: CBC-MAC

$$
\begin{array}{lll}
 & \text{(Init)} & \{\text{Empty}\} \\
c_1 := \mathcal{E}(m_1); & \text{(B1)} & \{(\text{Unequal}(c_1, c_1) \wedge \text{Indis}(c_1; \text{Var}, \mathcal{L}_{\mathcal{E}}, \mathcal{L}_{\mathcal{H}}; \text{Var})) \vee \\
 & & \quad (\text{Equal}(m_1, m_1) \wedge \text{Equal}(c_1, c_1) \wedge \\
 & & \quad\quad \text{Indis}(c_1; \text{Var}, \mathcal{L}_{\mathcal{E}}, \mathcal{L}_{\mathcal{H}}; \text{Var} - c_1))\} \\
z_2 := c_1 \oplus m_2; & \text{(G5)(X2)} & \{(\text{Indis}(c_1; \text{Var} - z_2, \mathcal{L}_{\mathcal{E}}, \mathcal{L}_{\mathcal{H}}; \text{Var}) \wedge \\
 & & \quad\quad \text{Indis}(z_2; \text{Var} - c_1, \mathcal{L}_{\mathcal{E}}, \mathcal{L}_{\mathcal{H}}; \text{Var})) \vee \\
 & \text{(G1)(X1)} & \quad (\text{Equal}(m_1, m_1) \wedge \text{Indis}(c_1; \text{Var} - z_2, \mathcal{L}_{\mathcal{E}}, \mathcal{L}_{\mathcal{H}}; \text{Var} - c_1 - z_2) \wedge \\
 & & \quad\quad \text{Unequal}(z_2, z_2) \wedge \text{Indis}(z_2; \text{Var} - c_1, \mathcal{L}_{\mathcal{E}}, \mathcal{L}_{\mathcal{H}}; \text{Var} - c_1 - z_2) \\
 & & \quad (\text{Equal}(m_1, m_1) \wedge \text{Equal}(m_2, m_2) \wedge \text{Equal}(z_2, z_2) \wedge \\
 & & \quad\quad \text{Indis}(c_1; \text{Var} - z_2, \mathcal{L}_{\mathcal{E}}, \mathcal{L}_{\mathcal{H}}; \text{Var} - c_1 - z_2) \wedge \\
 & & \quad\quad \text{Indis}(z_2; \text{Var} - c_1, \mathcal{L}_{\mathcal{E}}, \mathcal{L}_{\mathcal{H}}; \text{Var} - c_1 - z_2))\} \\
c_2 := \mathcal{E}(z_2) & \text{(B2)(B4)} & \{(\text{Indis}(c_1; \text{Var} - z_2, \mathcal{L}_{\mathcal{H}}; \text{Var}) \wedge \\
 & & \quad\quad \text{Indis}(c_2; \text{Var}, \mathcal{L}_{\mathcal{E}}, \mathcal{L}_{\mathcal{H}}; \text{Var})) \vee \\
 & \text{(G1)} & \quad (\text{Equal}(m_1, m_1) \wedge \text{Indis}(c_1; \text{Var} - z_2, \mathcal{L}_{\mathcal{H}}; \text{Var} - c_1 - z_2) \wedge \\
 & & \quad\quad \text{Indis}(c_2; \text{Var}, \mathcal{L}_{\mathcal{E}}, \mathcal{L}_{\mathcal{H}}; \text{Var})) \vee \\
 & \text{(B3)} & \quad (\text{Equal}(m_1, m_1) \wedge \text{Equal}(m_2, m_2) \wedge \\
 & & \quad\quad \text{Indis}(c_1; \text{Var} - z_2, \mathcal{L}_{\mathcal{H}}; \text{Var} - c_1 - z_2) \wedge \\
 & & \quad\quad \text{Indis}(c_2; \text{Var}, \mathcal{L}_{\mathcal{E}}, \mathcal{L}_{\mathcal{H}}; \text{Var} - c_2))\}
\end{array}
$$

Implementation

- Go through program from beginning to end, applying every rule possible at each step

Implementation

- Go through program from beginning to end, applying every rule possible at each step
- Lots of unnecessary predicates and to many OR clauses

Implementation

- Go through program from beginning to end, applying every rule possible at each step
- Lots of unnecessary predicates and to many OR clauses
  $\Rightarrow$ Predicate filter

Implementation

- Go through program from beginning to end, applying every rule possible at each step
- Lots of unnecessary predicates and to many OR clauses
  $\Rightarrow$ Predicate filter
- Discovery of loop invariants

Predicate Filter

Theorem

Predicates on variables that do not appear in what is left of the program and that do not imply any term in $UNIV(n)$ are not necessary to obtain a proof.

Predicate Filter

Theorem

Predicates on variables that do not appear in what is left of the program and that do not imply any term in $UNIV(n)$ are not necessary to obtain a proof.

- Heuristic: the term $\bigwedge_{i=1}^{n-1} \mathrm{Unequal}(c_n, c_i)$ from $UNIV(n)$ will be implied by $\mathrm{Indis}(c_n; \emptyset; \{c_1, \ldots, c_{n-1}\})$.

Invariants of **for** loops

Block Cipher

(F1) $\{\psi(p-1)\}$ for $l = p$ to q do: $[\mathsf{cmd}_l]$ $\{\psi(q)\}$ provided $\{\psi(l-1)\}$ cmd_l $\{\psi(l)\}$ for $p \leq l \leq q$

Invariants of **for** loops

Block Cipher

(F1) $\{\psi(p-1)\}$ for $l = p$ to q do: $[\mathrm{cmd}_l]$ $\{\psi(q)\}$ provided $\{\psi(l-1)\}$ cmd_l $\{\psi(l)\}$ for $p \leq l \leq q$

- express formula as $\phi_0(l)$
- find $\phi_1(l)$, $\phi_2(l)$ and $\phi_3(l)$ such that
  $\{\phi_0(l-1)\}$ cmd_l $\{\phi_1(l)\}$,
  $\{\phi_1(l-1)\}$ cmd_l $\{\phi_2(l)\}$,
  $\{\phi_2(l-1)\}$ cmd_l $\{\phi_3(l)\}$

Invariants of **for** loops

Block Cipher

(F1) $\{\psi(p-1)\}$ for $l = p$ to q do: $[\mathsf{cmd}_l]$ $\{\psi(q)\}$ provided $\{\psi(l-1)\}$ cmd_l $\{\psi(l)\}$ for $p \leq l \leq q$

- express formula as $\phi_0(l)$
- find $\phi_1(l)$, $\phi_2(l)$ and $\phi_3(l)$ such that
  $\{\phi_0(l-1)\}$ cmd_l $\{\phi_1(l)\}$,
  $\{\phi_1(l-1)\}$ cmd_l $\{\phi_2(l)\}$,
  $\{\phi_2(l-1)\}$ cmd_l $\{\phi_3(l)\}$
- extrapolate!

Several Options for Mixing Steps

Prop

- $MAC_1(m) = \mathcal{E}(h_i(m))$ *is a secure MAC with key* $sk = (i, \mathcal{E})$.
- $MAC_2(m) = \mathcal{G}(l \| h_i(m))$ *is a secure MAC with key* $sk = (i, k)$.
- $MAC_3(m) =$
  $$\begin{cases} \mathcal{E}_1(h_i(m')),\ m' = pad(m) \text{ if } m\text{'s length is not a multiple of } \eta \\ \mathcal{E}_2(h_i(m)) \text{ if } m\text{'s length is a multiple of } \eta \end{cases}$$
  is a secure MAC with key $sk = (i, \mathcal{E}_1, \mathcal{E}_2)$.
- $MAC_4(m) =$
  $$\begin{cases} \mathcal{E}(h_{\mathcal{E}}(m') \oplus k_1),\ m' = pad(m) \text{ if } m\text{'s length is not a multiple of } \eta \\ \mathcal{E}(h_{\mathcal{E}}(m) \oplus k_2) \text{ if } m\text{'s length is a multiple of } \eta \end{cases}$$
  is a secure MAC with key $sk = (\mathcal{E}, k_1, k_2)$

Mixing Steps

4 Options for Mixing Steps

- DMAC uses MAC_1, HMAC uses MAC_2
- ECBC and FCBC use MAC_3
- XCBC and PMAC use MAC_4

Summary

Method for automatically proving security of MACs

1. Modeling security properties
2. Defining a language
3. Building an Hoare Logic
4. Proving the security

Summary

Method for automatically proving security of MACs

1. Modeling security properties
2. Defining a language
3. Building an Hoare Logic
4. Proving the security

Filter that removes unnecessary predicates.

Heuristics for invariant generation for loop.

Summary

Method for automatically proving security of MACs

1. Modeling security properties
2. Defining a language
3. Building an Hoare Logic
4. Proving the security

Filter that removes unnecessary predicates.

Heuristics for invariant generation for loop.

Prototype (∼2000 lines OCaml)

`http://www.infsec.cs.uni-saarland.de/~gagne/macChecker/macChecker.html`

Future Works

- Make our prototype more user-friendly (SOON!)
- Integration of mixing step in the logic
- IND-CCA and authenticated encryption
- Better treatment of tweakable block ciphers
- Interaction or integration of our prototype with existing tools.

Thanks for your Attention

Questions ?