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 crypto?
Why make automated provers for crypto?

- People make mistakes
Automated Security Proofs for Almost-Universal Hash for MAC verification
Motivation and Background

Why make automated provers for crytpo?

- People make mistakes
- Mistakes in security proof of crypto protocols are bad
Why make automated provers for crypto?

- People make mistakes
- Mistakes in security proof of crypto protocols are bad
- Reinforce our trust in their security
Why make automated provers for crypto?

- 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

\( \text{Exp}_{\text{MAC}}: \)

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

**Definition**

\[
\text{ADV}_{\mathcal{A}}^{UNF}(\eta) = \Pr[\text{Exp}_{\text{MAC}} = 1]
\]

A MAC is **existentially unforgeable** if \( \text{ADV}_{\mathcal{A}}^{UNF}(\eta) \) 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 \((R_i)\) : \(\{P\} \text{ cmd } \{Q\}\)
Automated Security Proofs for Almost-Universal Hash for MAC verification

Motivation and Background

Hoare Logic [H69]

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

\[ \begin{align*}
  c_1 \\
  c_2 \\
  \vdots \\
  c_n
\end{align*} \]
Hoare Logic \([\text{H69}]\)

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

\[
\begin{align*}
\{P_0\} & \quad c_1 \\
\{P_1\} & \quad c_2 \\
\vdots & \\
\{P_n\} & \quad c_n
\end{align*}
\]
Automated Security Proofs for Almost-Universal Hash for MAC verification
Motivation and Background

Hoare Logic [H69]

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

\[
\begin{align*}
\{P_0\} & \quad c_1 \\
c_2 & \\
& \vdots \\
c_n & \quad \{Q_n\} \quad ?
\end{align*}
\]
Automated Security Proofs for Almost-Universal Hash for MAC verification

Motivation and Background

Hoare Logic \((\text{H69})\)

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

\((R_5)\{P_0\} \ c_1 \ \{Q_0\}\)

\[c_2\]

\[\vdots\]

\[c_n \ \{Q_n\} \ ?\]
Automated Security Proofs for Almost-Universal Hash for MAC verification

Motivation and Background

Hoare Logic \([H69]\)

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

\((R_5)\) \(\{P_0\}\ c_1 \ \{Q_0\}\)

\((R_2)\) \(\{P_1\}\ c_2 \ \{Q_2\}\), where \(P_1 \subseteq Q_0\)

\[
\vdots
\]

\(c_n \ \{Q_n\}\ ?\)
Automated Security Proofs for Almost-Universal Hash for MAC verification
Motivation and Background

Hoare Logic [H69]

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

\((R_5)\){\(P_0\}} \ c_1 \ {Q_0}\)
\((R_2)\){\(P_1\}} \ c_2 \ {Q_2}\), 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

⇒ We need fundamentally new trick
⇒ 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

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

Example

\[
\begin{align*}
z_1 & := k || m_1; \\
c_1 & := H(z_1); \\
\text{for } i &= 2 \text{ to } n \text{ do:;} \\
    z_i & := c_{i-1} || m_i \\
    c_i & := H(z_i) \\
    z' & := k' || c_n; \\
tag & := H(z');
\end{align*}
\]
Almost-universal hash function [CW79]

Definition

A hash function family \( \{H_k\}_{k \in \{0,1\}^n} \) 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

\[ \text{c ::= } x := \rho^i(y) \mid x := E(y) \mid x := H(y) \mid x := y \oplus z \mid x := y \Vert z \mid x := y \mid \text{for } x = i \text{ to } j \text{ do: } c_x \mid c_1; c_2 \]
Generic Hash

Definition

A generic hash function $M$ is represented by

$$Hash_M(m_1|\ldots|m_p, c_p) : \text{var } \vec{x}; \text{cmd}$$

$$Hash_{CBC}(m_1|m_2|m_3, c_3) :$$

$$\text{var } z_1, z_2, z_3;$$
$$c_1 := E(m_1);$$
$$z_2 := c_1 \oplus m_2;$$
$$c_2 := E(z_2);$$
$$z_3 := c_2 \oplus m_3;$$
$$c_3 := E(z_3);$$
Semantics of the Programming Language

\[
\begin{align*}
[x := \mathcal{E}(y)](S, S', \mathcal{E}, \mathcal{H}, \mathcal{L}_\mathcal{E}, \mathcal{L}_\mathcal{H}) &= (S\{x \mapsto \mathcal{E}(S(y))\}, S'\{x \mapsto \mathcal{E}(S'(y))\}, \mathcal{E}, \mathcal{H}, \mathcal{L}_\mathcal{E} \cup \{S(y), S'(y)\}, \mathcal{L}_\mathcal{H}) \\
[x := \mathcal{H}(y)](S, S', \mathcal{E}, \mathcal{H}, \mathcal{L}_\mathcal{E}, \mathcal{L}_\mathcal{H}) &= (S\{x \mapsto \mathcal{H}(S(y))\}, S'\{x \mapsto \mathcal{H}(S'(y))\}, \mathcal{E}, \mathcal{H}, \mathcal{L}_\mathcal{E}, \mathcal{L}_\mathcal{H} \cup \{S(y), S'(y)\}) \\
[x := y](S, S', \mathcal{E}, \mathcal{H}, \mathcal{L}_\mathcal{E}, \mathcal{L}_\mathcal{H}) &= (S\{x \mapsto S(y)\}, S'\{x \mapsto S'(y)\}, \mathcal{E}, \mathcal{H}, \mathcal{L}_\mathcal{E}, \mathcal{L}_\mathcal{H}) \\
[x := y \oplus z](S, S', \mathcal{E}, \mathcal{H}, \mathcal{L}_\mathcal{E}, \mathcal{L}_\mathcal{H}) &= (S\{x \mapsto S(y) \oplus S(z)\}, S'\{x \mapsto S'(y) \oplus S'(z)\}, \mathcal{E}, \mathcal{H}, \mathcal{L}_\mathcal{E}, \mathcal{L}_\mathcal{H}) \\
[x := y | z](S, S', \mathcal{E}, \mathcal{H}, \mathcal{L}_\mathcal{E}, \mathcal{L}_\mathcal{H}) &= (S\{x \mapsto S(y) | S(z)\}, S'\{x \mapsto S'(y) | S'(z)\}, \mathcal{E}, \mathcal{H}, \mathcal{L}_\mathcal{E}, \mathcal{L}_\mathcal{H}) \\
[x := \rho(i, y)](S, S', \mathcal{E}, \mathcal{H}, \mathcal{L}_\mathcal{E}, \mathcal{L}_\mathcal{H}) &= (S\{x \mapsto \rho(i, S(y))\}, S'\{x \mapsto \rho(i, S'(y))\}, \mathcal{E}, \mathcal{H}, \mathcal{L}_\mathcal{E}, \mathcal{L}_\mathcal{H}) \\
[\text{for } l = p \text{ to } q \text{ do: } [\text{cmd}_l]]_\gamma &= \begin{cases} [\text{cmd}_q] \circ [\text{cmd}_{q-1}] \circ \ldots \circ [\text{cmd}_p]_\gamma & \text{if } p \leq q \\ \gamma & \text{otherwise} \end{cases} \\
[c_1; c_2] &= [c_2] \circ [c_1]
\end{align*}
\]
Automated Security Proofs for Almost-Universal Hash for MAC verification

Our Strategy and Model

Semantics of the Programming Language

\[
\begin{align*}
[x := E(y)](S, S', E, H, L_E, L_H) &= (S\{x \mapsto E(S(y))\}, S'\{x \mapsto E(S'(y))\}, E, H, L_E \cup \{S(y), S'(y)\}, L_H) \\
[x := H(y)](S, S', E, H, L_E, L_H) &= (S\{x \mapsto H(S(y))\}, S'\{x \mapsto H(S'(y))\}, E, H, L_E, L_H \cup \{S(y), S'(y)\}) \\
[x := y](S, S', E, H, L_E, L_H) &= (S\{x \mapsto S(y)\}, S'\{x \mapsto S'(y)\}, E, H, L_E, L_H) \\
[x := y \oplus z](S, S', E, H, L_E, L_H) &= (S\{x \mapsto S(y) \oplus S(z)\}, S'\{x \mapsto S'(y) \oplus S'(z)\}, E, H, L_E, L_H) \\
[x := y || z](S, S', E, H, L_E, L_H) &= (S\{x \mapsto S(y) || S(z)\}, S'\{x \mapsto S'(y) || S'(z)\}, E, H, L_E, L_H) \\
[x := \rho(i, y)](S, S', E, H, L_E, L_H) &= (S\{x \mapsto \rho(i, S(y))\}, S'\{x \mapsto \rho(i, S'(y))\}, E, H, L_E, L_H) \\
[\text{for } l = p \text{ to } q \text{ do: } [\text{cmd}_l]] \gamma &= \begin{cases} [\text{cmd}_q] \circ [\text{cmd}_{q-1}] \circ \ldots \circ [\text{cmd}_p] \gamma \text{ if } p \leq q \\
\gamma \text{ otherwise} \end{cases} \\
[c_1; c_2] &= [c_2] \circ [c_1]
\end{align*}
\]
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**(*E; x; V*): the probability that the value of *x* is either in $L_E$ or in *V* is negligible.

**H**(*H, x; V*): the probability that the value of *x* is either in $L_H$ or in *V* is negligible.

**Empty**: that the probability that $L_E$ 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 $P(m_1||\ldots||m_n, c : \texttt{var } \vec{x}; \texttt{cmd})$ is an almost-universal hash function if for all $n$, $\{(\texttt{init})\} \texttt{cmd} \{\texttt{UNIV}(n)\}$ holds:

$$\left( \bigwedge_{i=1}^{n-1} \text{Unequal}(c_n, c_i) \land \bigwedge_{i=1}^{n} \text{Equal}(m_i, m_i) \right) \lor \bigwedge_{i=1}^{n} \text{Unequal}(c_n, c_i)$$
Outline
Some of our 40 Rules

**XOR**

(X4) \( \{ \text{Equal}(y, y) \land \text{Unequal}(z, z) \} \ x := y \oplus z \)

\( \{ \text{Unequal}(x, x) \} \) if \( y \neq z \)

**Concatenation**

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

**Assignment**

(A1) \( \{ \text{true} \} \ x := m_i \ \{ (\text{Equal}(m_i, m_i) \land \text{Equal}(x, x)) \lor \text{Unequal}(x, x) \} \)
Some of our 40 Rules

**XOR**

<table>
<thead>
<tr>
<th>Rule (X4)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>{Equal(y, y) \land Unequal(z, z)} \ x := y \oplus z {Unequal(x, x)} if y \neq z</td>
<td></td>
</tr>
</tbody>
</table>

**Concatenation**

<table>
<thead>
<tr>
<th>Rule (C6)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>{Indis(y; V, L_H; V')} \ x := y</td>
<td>z {H(H, x; V)}</td>
</tr>
</tbody>
</table>

**Assignment**

<table>
<thead>
<tr>
<th>Rule (A1)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>{true} \ x := m_i {(Equal(m_i, m_i) \land Equal(x, x)) \lor Unequal(x, x)}</td>
<td></td>
</tr>
</tbody>
</table>
Some of our 40 Rules

Block Cipher

(B1) \{\text{Empty}\} \ x := \mathcal{E}(m_i)
\{\text{Unequal}(x, x) \land \text{Indis}(x; \text{Var}, \mathcal{L}_\mathcal{E}, \mathcal{L}_\mathcal{H}; \text{Var})\} \lor
\{\text{Equal}(m_i, m_i) \land \text{Equal}(x, x) \land
\text{Indis}(x; \text{Var}, \mathcal{L}_\mathcal{E}, \mathcal{L}_\mathcal{H}; \text{Var} - x)\}\}

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

(B3) \{\mathcal{E}(\mathcal{E}; y; \emptyset) \land \text{Equal}(y, y)\} \ x := \mathcal{E}(y)
\{\text{Indis}(x; \text{Var}, \mathcal{L}_\mathcal{E}, \mathcal{L}_\mathcal{H}; \text{Var} - x) \land \text{Equal}(x, x)\}
Example: CBC-MAC

\[
c_1 := \mathcal{E}(m_1); \quad \text{(Init)} \quad \{\text{Empty}\}
\]
\[
\begin{align*}
c_1 &:= \mathcal{E}(m_1); \quad \text{(B1)} \quad \{(\text{Unequal}(c_1, c_1) \land \text{Indis}(c_1; \text{Var}, \mathcal{E}, \mathcal{H}; \text{Var})) \lor \\
&\quad (\text{Equal}(m_1, m_1) \land \text{Equal}(c_1, c_1) \land \\
&\quad \text{Indis}(c_1; \text{Var}, \mathcal{E}, \mathcal{H}; \text{Var} - c_1))\}\end{align*}
\]
\[
z_2 := c_1 \oplus m_2; \quad \text{(G5)(X2)} \quad \{(\text{Indis}(c_1; \text{Var} - z_2, \mathcal{E}, \mathcal{H}; \text{Var}) \land \\
&\quad \text{Indis}(z_2; \text{Var} - c_1, \mathcal{E}, \mathcal{H}; \text{Var})) \lor \\
&\quad (\text{Equal}(m_1, m_1) \land \text{Indis}(c_1; \text{Var} - z_2, \mathcal{E}, \mathcal{H}; \text{Var} - c_1 - z_2) \land \\
&\quad \text{Unequal}(z_2, z_2) \land \text{Indis}(z_2; \text{Var} - c_1, \mathcal{E}, \mathcal{H}; \text{Var} - c_1 - z_2)) \lor \\
&\quad (\text{Equal}(m_1, m_1) \land \text{Equal}(m_2, m_2) \land \text{Equal}(z_2, z_2) \land \\
&\quad \text{Indis}(c_1; \text{Var} - z_2, \mathcal{E}, \mathcal{H}; \text{Var} - c_1 - z_2) \land \\
&\quad \text{Indis}(z_2; \text{Var} - c_1, \mathcal{E}, \mathcal{H}; \text{Var} - c_1 - z_2))\}\end{align*}
\]
\[
c_2 := \mathcal{E}(z_2) \quad \text{(B2)(B4)} \quad \{(\text{Indis}(c_1; \text{Var} - z_2, \mathcal{H}; \text{Var}) \land \\
&\quad \text{Indis}(c_2; \text{Var}, \mathcal{E}, \mathcal{H}; \text{Var})) \lor \\
&\quad (\text{Equal}(m_1, m_1) \land \text{Indis}(c_1; \text{Var} - z_2, \mathcal{H}; \text{Var} - c_1 - z_2) \land \\
&\quad \text{Indis}(c_2; \text{Var}, \mathcal{E}, \mathcal{H}; \text{Var})) \lor \\
&\quad (\text{Equal}(m_1, m_1) \land \text{Equal}(m_2, m_2) \land \\
&\quad \text{Indis}(c_1; \text{Var} - z_2, \mathcal{H}; \text{Var} - c_1 - z_2) \land \\
&\quad \text{Indis}(c_2; \text{Var}, \mathcal{E}, \mathcal{H}; \text{Var} - c_2))\}\end{align*}
\]
Example: CBC-MAC

\[
\begin{align*}
c_1 &:= E(m_1); \quad \text{(Init)} \quad \{\text{Empty}\} \\
\{\text{Unequal}(c_1, c_1) \land \text{Indis}(c_1; \text{Var}, \mathcal{L}_E, \mathcal{L}_H; \text{Var})\} \lor \\
\{\text{Equal}(m_1, m_1) \land \text{Equal}(c_1, c_1) \land \\
\quad \text{Indis}(c_1; \text{Var}, \mathcal{L}_E, \mathcal{L}_H; \text{Var} - c_1)\}\} \\
z_2 &:= c_1 \oplus m_2; \quad \text{(G5)(X2)} \quad \{\text{Indis}(c_1; \text{Var} - z_2, \mathcal{L}_E, \mathcal{L}_H; \text{Var}) \land \\
\quad \text{Indis}(z_2; \text{Var} - c_1, \mathcal{L}_E, \mathcal{L}_H; \text{Var})\} \lor \\
\{\text{Equal}(m_1, m_1) \land \text{Indis}(c_1; \text{Var} - z_2, \mathcal{L}_E, \mathcal{L}_H; \text{Var} - c_1 - z_2) \land \\
\quad \text{Indis}(c_2; \text{Var} - c_1, \mathcal{L}_E, \mathcal{L}_H; \text{Var} - c_1 - z_2)\} \\
c_2 &:= E(z_2) \quad \text{(B2)(B4)} \quad \{\text{Indis}(c_1; \text{Var} - z_2, \mathcal{L}_H; \text{Var}) \land \\
\quad \text{Indis}(c_2; \text{Var}, \mathcal{L}_E, \mathcal{L}_H; \text{Var})\} \lor \\
\{\text{Equal}(m_1, m_1) \land \text{Indis}(c_1; \text{Var} - z_2, \mathcal{L}_H; \text{Var} - c_1 - z_2) \land \\
\quad \text{Indis}(c_2; \text{Var}, \mathcal{L}_E, \mathcal{L}_H; \text{Var})\} \lor \\
\{\text{Equal}(m_1, m_1) \land \text{Equal}(m_2, m_2) \land \\
\quad \text{Indis}(c_1; \text{Var} - z_2, \mathcal{L}_H; \text{Var} - c_1 - z_2) \land \\
\quad \text{Indis}(c_2; \text{Var} - c_1, \mathcal{L}_E, \mathcal{L}_H; \text{Var} - c_2)\}\}
\end{align*}
\]
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 too many OR clauses
Implementation

- Go through program from beginning to end, applying every rule possible at each step
- Lots of unnecessary predicates and too 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 too many OR clauses
  ⇒ 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 $\wedge_{i=1}^{n-1}$ Unequal($c_n, c_i$) from $UNIV(n)$ will be implied by $\text{Indis}(c_n; \emptyset; \{c_1, \ldots, c_{n-1}\})$. 
Invariants of \textbf{for} loops

\begin{itemize}
  \item Block Cipher
  \begin{align*}
    \text{(F1) } & \{ \psi(p - 1) \} \text{ for } l = p \text{ to } q \text{ do: } [\text{cmd}_l] \{ \psi(q) \} \text{ provided} \\
    & \{ \psi(l - 1) \} \text{ cmd}_l \{ \psi(l) \} \text{ for } p \leq l \leq q
  \end{align*}
\end{itemize}
Invariants of *for* loops

<table>
<thead>
<tr>
<th>Block Cipher</th>
</tr>
</thead>
<tbody>
<tr>
<td>(F1) {ψ(p − 1)} for (l = p) to (q) do: [cmd(_l)] {ψ(q)} provided {ψ(l − 1)} cmd(_l) {ψ(l)} for (p \leq l \leq q)</td>
</tr>
</tbody>
</table>

- express formula as \(φ_0(l)\)
- find \(φ_1(l), φ_2(l)\) and \(φ_3(l)\) such that
  \(\{φ_0(l − 1)\} cmd\(_l\) \{φ_1(l)\}\),
  \(\{φ_1(l − 1)\} cmd\(_l\) \{φ_2(l)\}\),
  \(\{φ_2(l − 1)\} cmd\(_l\) \{φ_3(l)\}\)
Invariants of for loops

Block Cipher

(F1) \{\psi(p - 1)\} for \(l = p\) to \(q\) do: \([\text{cmd}_l]\) \{\psi(q)\} provided
\{\psi(l - 1)\} \text{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)\} \text{cmd}_l \{\phi_1(l)\},
  \{\phi_1(l - 1)\} \text{cmd}_l \{\phi_2(l)\},
  \{\phi_2(l - 1)\} \text{cmd}_l \{\phi_3(l)\}
- extrapolate!
Several Options for Mixing Steps

<table>
<thead>
<tr>
<th>Prop</th>
</tr>
</thead>
<tbody>
<tr>
<td>• $\text{MAC}_1(m) = \mathcal{E}(h_i(m))$ is a secure MAC with key $sk = (i, \mathcal{E})$.</td>
</tr>
<tr>
<td>• $\text{MAC}_2(m) = \mathcal{G}(l|h_i(m))$ is a secure MAC with key $sk = (i, k)$.</td>
</tr>
</tbody>
</table>
| • $\text{MAC}_3(m) =$  
  \[
  \begin{cases} 
  \mathcal{E}_1(h_i(m')), & m' = \text{pad}(m) \text{ if } m's \text{ length is not a multiple of } \eta \\
  \mathcal{E}_2(h_i(m)) & \text{if } m's \text{ length is a multiple of } \eta 
  \end{cases}
  \]
  is a secure MAC with key $sk = (i, \mathcal{E}_1, \mathcal{E}_2)$. |
| • $\text{MAC}_4(m) =$  
  \[
  \begin{cases} 
  \mathcal{E}(h_\mathcal{E}(m') \oplus k_1), & m' = \text{pad}(m) \text{ if } m's \text{ length is not a multiple of } \eta \\
  \mathcal{E}(h_\mathcal{E}(m) \oplus k_2) & \text{if } m's \text{ length is a multiple of } \eta 
  \end{cases}
  \]
  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 ?