Automated Analysis and Synthesis of Modes of Operation and Authenticated Encryption Schemes

Alex J. Malozemoff

University of Maryland

Joint work with Matthew Green, Viet Tung Hoang, and Jonathan Katz

Presented at the IACR School on Computer-aided Cryptography, University of Maryland, College Park, USA, June 1 – June 4, 2015.
Problem: Designing/proving crypto constructions is hard
Problem: Designing/proving crypto constructions is hard

(Possible) Solution: Use program synthesis!
Introduction

Problem: Designing/proving crypto constructions is hard

(Possible) Solution: Use *program synthesis*!

Program Synthesis

- Automatically construct programs based on (small) set of rules
- Has been applied to crypto protocols (e.g., [AGHP12],[BCG+13], etc.)
This Work

Two results:

1. Apply program synthesis to \textit{modes of operation} (CSF 2014)
   - Joint with Matthew Green and Jonathan Katz
This Work

Two results:

1. Apply program synthesis to *modes of operation* (CSF 2014)
   - Joint with Matthew Green and Jonathan Katz

2. Extend ideas to *authenticated encryption* (new)
   - Joint with Viet Tung Hoang and Jonathan Katz
Two results:

1. Apply program synthesis to *modes of operation* (CSF 2014)
   - Joint with Matthew Green and Jonathan Katz

2. Extend ideas to *authenticated encryption* (new)
   - Joint with Viet Tung Hoang and Jonathan Katz
Background: Modes of Operation

Blockcipher (= PRP, $F_k$): Encrypts fixed-length message (e.g., AES)
Background: Modes of Operation

Blockcipher \((= PRP, F_k)\): Encrypts \textit{fixed-length} message (e.g., AES)

**Mode of Operation:** encrypts \textit{arbitrary-length} messages, using blockcipher as building block
Background: Modes of Operation

Blockcipher (\(=\) PRP, \(F_k\)): Encrypts fixed-length message (e.g., AES)

Mode of Operation: encrypts arbitrary-length messages, using blockcipher as building block

Example: Cipher-Block Chaining (CBC) Mode
Background: Security of Modes of Operation

Want output of mode to look “random” to adversary \(\Rightarrow\) IND$\$-CPA

What is IND$\$-CPA?

Adversary \(\mathcal{A}\) has oracle access to either
- (World 0) a truly random function
- (World 1) the desired mode of operation

\(\mathcal{A}\) specifies messages to encrypt and receives resulting ciphertexts

\(\mathcal{A}'\)'s Goal: Decide whether in World 0 or World 1

Secure: \(\mathcal{A}\) cannot distinguish between worlds
Lots of modes exist; some modes are complex

Each scheme requires separate security proof

- proofs occasionally omitted, sometimes wrong!
Motivation

Lots of modes exist; some modes are complex
Each scheme requires separate security proof
  • proofs occasionally omitted, sometimes wrong!

Question: Can we automate the security analysis, synthesize new modes?

Solution: Construct framework for automatically proving modes of operation secure, use this to synthesize new modes
Modes of Operation

Model (single block of) mode as *directed, acyclic graph*

- Nodes → atomic operations
  - E.g., XOR two values, apply PRP to value, etc.
- Edges → intermediate values

Meta-Theorem: Exists valid labeling = ⇒ mode IND$-CPA-secure
Modes of Operation

Model (single block of) mode as *directed, acyclic graph*
- Nodes $\rightarrow$ atomic operations
  - E.g., XOR two values, apply PRP to value, etc.
- Edges $\rightarrow$ intermediate values

Each edge assigned *label*
- *Constraints* restrict how edges can be labeled
Modes of Operation

Model (single block of) mode as *directed, acyclic graph*

- Nodes → atomic operations
  - E.g., XOR two values, apply PRP to value, etc.
- Edges → intermediate values

Each edge assigned *label*

- *Constraints* restrict how edges can be labeled

**Meta-Theorem:** Exists valid labeling $\implies$ mode IND$\$-CPA-secure
Mode of Operation: Formal Definition

Defined by two algorithms:

- **Init**(1^n) → (c_0, z_0)
- **Block**(m_i, z_{i-1}) → (c_i, z_i)

**Enc_k**(m = m_1∥⋯∥m_ℓ):

- Compute (c_0, z_0) ← Init(1^n)
- For i = 1, ..., ℓ:
  Compute (c_i, z_i) ← Block(m_i, z_{i-1})
- Output c_0∥⋯∥c_ℓ

Diagram showing the flow of data through the cryptographic functions F_k.
Viewing Modes as Graphs

\[ \begin{array}{ccc} \text{Init algorithm} & \Rightarrow & \text{Block algorithm} \\
\text{GenRand} & \text{DUP} & \text{OUT} \\
\text{DUP} & \text{NEXTIV} & \\
\text{START} & \text{M} & \\
\text{XOR} & \text{PRP} & \\
\text{DUP} & \text{OUT} & \text{NEXTIV} \\
\end{array} \]
Recall: Edges denote intermediate values
Edge Labels: Intuition

Recall: Edges denote intermediate values

Intuition: Labels should capture “properties” of intermediate value
  • E.g., does value look random to adversary?
Edge Labels: Intuition

**Recall:** Edges denote intermediate values

**Intuition:** Labels should capture “properties” of intermediate value
  - E.g., does value look random to adversary?

**Goal:** If values on edges into **OUT** nodes look random to adversary, then mode is IND$-CPA-secure
Each edge label is a tuple \( (\text{type}, \text{flags}) \): 

- **type** \( \in \{\bot, R\} \): “Type” of intermediate value
  - \( \bot \): Adversarially controlled
  - \( R \): Random
- **flags** \( \in \{0, 1\}^2 \): Bit-vector denoting whether edge can be input into **OUT** or **PRP**
  - E.g., prevents both **DUP**’d values being output as ciphertext
Constraints

Constraints on Nodes:

- **GENRAND**: Outgoing edge gets type $R$, flags $PRP = 1$, flags $OUT = 1$
- **DUP**: Outgoing edges inherit ingoing edge's type, split flag bits
- **OUT**: Ingoing edge must have type $R$ and flags $PRP = 1$; Outgoing edge same as $GENRAND$
- **NEXTIV**: Ingoing edge must have type $R$ and flags $OUT = 1$
- **START**: Inherits type and flag bits of ingoing edge to $NEXTIV$
- **M**: Outgoing edge gets type $\perp$, flags $PRP = 0$, flags $OUT = 0$
- **XOR**: At least one ingoing edge of type $R$; Outgoing edge gets type $R$ and OR of ingoing edges' flags
- **PRP**: Ingoing edge must have type $R$ and flags $PRP = 1$; Outgoing edge same as $GENRAND$

Init algorithm

Block algorithm
Constraints on Nodes:

- **GENRAND**: Outgoing edge gets type $R$, flags $PRP = 1$, flags $OUT = 1$

- **DUP**: Outgoing edges inherit ingoing edge's type, split flag bits

- **START**: Inherits type and flag bits of ingoing edge to **NEXTIV**

- **M**: Outgoing edge gets type $\perp$, flags $PRP = 0$, flags $OUT = 0$

- **XOR**: At least one ingoing edge of type $R$; Outgoing edge gets type $R$ and OR of ingoing edges' flags

- **PRP**: Ingoing edge must have type $R$ and flags $PRP = 1$; Outgoing edge same as **GENRAND**

- **OUT**: Ingoing edge must have type $R$ and flags $OUT = 1$
Constraints on Nodes:

- **GENRAND**: Outgoing edge gets type \( R \), flags.\( PRP = 1 \), flags.\( OUT = 1 \)
- **DUP**: Outgoing edges inherit ingoing edge’s type, split flag bits
Constraints on Nodes:

- **GENRAND**: Outgoing edge gets type $R$, flags.$PRP = 1$, flags.$OUT = 1$
- **DUP**: Outgoing edges inherit ingoing edge’s type, split flag bits
- **START**: Inherits type and flag bits of ingoing edge to $NEXTIV$
- **$M$**: Outgoing edge gets type $\bot$, flags.$PRP = 0$, flags.$OUT = 0$
- **$XOR$**: At least one ingoing edge of type $R$; Outgoing edge gets type $R$ and OR of ingoing edges’ flags
Constraints on Nodes:

- **GENRAND**: Outgoing edge gets type $R$, flags.$PRP = 1$, flags.$OUT = 1$
- **DUP**: Outgoing edges inherit ingoing edge’s type, split flag bits
- **START**: Inherits type and flag bits of ingoing edge to NEXTIV
- **M**: Outgoing edge gets type $\perp$, flags.$PRP = 0$, flags.$OUT = 0$
Constraints on Nodes:

- **GENRAND**: Outgoing edge gets type $\mathbf{R}$, flags.$\mathbf{PRP} = 1$, flags.$\mathbf{OUT} = 1$
- **DUP**: Outgoing edges inherit ingoing edge’s type, split flag bits
- **START**: Inherits type and flag bits of ingoing edge to NEXTIV
- **M**: Outgoing edge gets type $\perp$, flags.$\mathbf{PRP} = 0$, flags.$\mathbf{OUT} = 0$
- **XOR**: At least one ingoing edge of type $\mathbf{R}$; Outgoing edge gets type $\mathbf{R}$ and OR of ingoing edges’ flags
Constraints on Nodes:

- **GENRAND**: Outgoing edge gets type $R$, flags.$PRP = 1$, flags.$OUT = 1$
- **DUP**: Outgoing edges inherit ingoing edge’s type, split flag bits
- **START**: Inherits type and flag bits of ingoing edge to NEXTIV
- **M**: Outgoing edge gets type $\perp$, flags.$PRP = 0$, flags.$OUT = 0$
- **XOR**: At least one ingoing edge of type $R$; Outgoing edge gets type $R$ and OR of ingoing edges’ flags
- **PRP**: Ingoing edge must have type $R$ and flags.$PRP = 1$; Outgoing edge same as GENRAND
Constraints on Nodes:

- **GENRAND**: Outgoing edge gets type \( R \), flags.\( PRP = 1 \), flags.\( OUT = 1 \)
- **DUP**: Outgoing edges inherit ingoing edge’s type, split flag bits
- **START**: Inherits type and flag bits of ingoing edge to \( \text{NEXTIV} \)
- **M**: Outgoing edge gets type \( \perp \), flags.\( PRP = 0 \), flags.\( OUT = 0 \)
- **XOR**: At least one ingoing edge of type \( R \); Outgoing edge gets type \( R \) and OR of ingoing edges’ flags
- **PRP**: Ingoing edge must have type \( R \) and flags.\( PRP = 1 \); Outgoing edge same as GENRAND
- **OUT**: Ingoing edge must have type \( R \) and flags.\( OUT = 1 \)
Implemented model checker + synthesizer in OCaml

**Model Checker:**

(1) Checks whether an input mode is secure
   - **Recall:** Valid labeling $\Rightarrow$ mode is secure
   $\Rightarrow$ Determining secure mode is a constraint-satisfaction problem
   $\Rightarrow$ Can use SMT solver (e.g., Z3)

(2) Secure modes need to be decryptable
   - Implement algorithm to check decryptability of mode

**Synthesizer:**

Can simply iterate over all possible graphs
   - Use simple rules to reduce search space
Ran model checker for modes with \( \leq 10 \) instructions

<table>
<thead>
<tr>
<th># Instructions</th>
<th>Valid</th>
<th>Decryptable</th>
<th>Secure</th>
</tr>
</thead>
<tbody>
<tr>
<td>1–6</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>50</td>
<td>30</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>549</td>
<td>281</td>
<td>31</td>
</tr>
<tr>
<td>9</td>
<td>3130</td>
<td>1304</td>
<td>150</td>
</tr>
<tr>
<td>10</td>
<td>5107</td>
<td>1770</td>
<td>184</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>8836</td>
<td>3383</td>
<td><strong>370</strong></td>
</tr>
</tbody>
</table>
Synthesis Results

Ran model checker for modes with $\leq 10$ instructions

<table>
<thead>
<tr>
<th># Instructions</th>
<th>Valid</th>
<th>Decryptable</th>
<th>Secure</th>
</tr>
</thead>
<tbody>
<tr>
<td>1–6</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>50</td>
<td>30</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>549</td>
<td>281</td>
<td>31</td>
</tr>
<tr>
<td>9</td>
<td>3130</td>
<td>1304</td>
<td>150</td>
</tr>
<tr>
<td>10</td>
<td>5107</td>
<td>1770</td>
<td>184</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>8836</strong></td>
<td><strong>3383</strong></td>
<td><strong>370</strong></td>
</tr>
</tbody>
</table>

We are able to synthesize all standard (secure) modes
- E.g., CBC, OFB, CFB, CTR, PCBC
Results

Two results:

1. Apply program synthesis to *modes of operation* (CSF 2014)
2. Extend previous ideas to *authenticated encryption* (new)
Background: Authenticated Encryption

Previous approach gave modes with privacy but not authenticity
Background: Authenticated Encryption

Previous approach gave modes with privacy but not authenticity

**Authenticated Encryption (AE):** mode of operation which encrypts and authenticates arbitrary-length messages
AE scheme: Tuple of algorithms $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$, where $\mathcal{E}(\cdot, \cdot, \cdot)$ and $\mathcal{D}(\cdot, \cdot, \cdot)$ are deterministic and take nonce as input, in addition to key and plaintext

Adversary has access to oracle $\mathcal{E}(K, \cdot, \cdot)$ and must never repeat a nonce

Privacy: Similar to previous case

Authenticity: Adversary wins if it can output $(N, C)$ such that $\mathcal{D}(K, N, C) \neq \perp$ and $C$ is not an oracle output
Background: Tweakable Blockciphers

We restrict ourselves to AE schemes using tweakable blockciphers.
We restrict ourselves to AE schemes using **tweakable blockciphers**

**Tweakable Blockcipher (TBC):** Like (standard) blockcipher, except takes an extra argument (the "tweak"):

- \( E : \mathcal{K} \times \mathcal{T} \times \{0, 1\}^n \rightarrow \{0, 1\}^n \), where \( E_K(T, \cdot) \) is a PRP on \( \{0, 1\}^n \) for every \( K \in \mathcal{K} \) and \( T \in \mathcal{T} \)
We restrict ourselves to AE schemes using **tweakable blockciphers**

**Tweakable Blockcipher (TBC):** Like (standard) blockcipher, except takes an extra argument (the “tweak”):

- \( E : \mathcal{K} \times \mathcal{T} \times \{0, 1\}^n \to \{0, 1\}^n \), where \( E_K(T, \cdot) \) is a PRP on \( \{0, 1\}^n \) for every \( K \in \mathcal{K} \) and \( T \in \mathcal{T} \)

Simplifies analysis:

- If tweak is unique, output of TBC is random
Background: Tweakable Blockciphers

We restrict ourselves to AE schemes using tweakable blockciphers

**Tweakable Blockcipher (TBC):** Like (standard) blockcipher, except takes an extra argument (the “tweak”):

- \( E : \mathcal{K} \times \mathcal{T} \times \{0, 1\}^n \rightarrow \{0, 1\}^n \), where \( E_K(T, \cdot) \) is a PRP on \( \{0, 1\}^n \) for every \( K \in \mathcal{K} \) and \( T \in \mathcal{T} \)

Simplifies analysis:

- If tweak is unique, output of TBC is random
- Tweak depends on nonce, and nonce must be fresh on every query \( \Rightarrow \) output of TBC is *always* random
Example: Offset Codebook (OCB) Mode

\[ M_1 \rightarrow E_K^{N,1} \rightarrow C_1 \]
\[ M_2 \rightarrow E_K^{N,2} \rightarrow C_2 \]
\[ M_3 \rightarrow E_K^{N,3} \rightarrow C_3 \]
\[ M_4 \rightarrow E_K^{N,4} \rightarrow C_4 \]
\[ \Sigma \rightarrow E_K^{N,-4} \]

\[ \tau \rightarrow \text{tag} \]
Authenticated Encryption: Formal Definition

Defined by three algorithms (operating over two blocks at a time):

- $\text{Enc}^E_K (T, X, M_1 M_2) \rightarrow (Y, C_1 C_2)$
- $\text{Dec}^{E_K, E_K^{-1}} (T, Y, C_1 C_2) \rightarrow (X, M_1 M_2)$
- $\text{Tag}^E_K (T, X) \rightarrow V$
Authenticated Encryption: Formal Definition

Defined by three algorithms (operating over two blocks at a time):

- $\text{Enc}^{E_K}(T, X, M_1M_2) \rightarrow (Y, C_1C_2)$
- $\text{Dec}^{E_K,E_K^{-1}}(T, Y, C_1C_2) \rightarrow (X, M_1M_2)$
- $\text{Tag}^{E_K}(T, X) \rightarrow V$

Encryption procedure:

\[
\begin{align*}
X_0 & \leftarrow 0^{2n}; \quad \nu \leftarrow 1; \quad M_1 \cdots M_{2m} \leftarrow M \\
\text{for } i = 1 \text{ to } m \text{ do} & \\
& \quad T \leftarrow (N, \nu) \\
& \quad (X_i, C_{2i-1}C_{2i}) \leftarrow \text{Enc}^{E_K}(T, X_{i-1}, M_{2i-1}M_{2i}) \\
& \quad \nu \leftarrow \nu + \text{Cost}(\Pi) \\
& \quad T \leftarrow (N, 1 - \nu) \\
& \quad V \leftarrow \text{Tag}^{E_K}(T, X) \\
\text{return } C_1 \cdots C_{2m} \parallel V[1, \tau] & 
\end{align*}
\]
Authenticated Encryption: Formal Definition

Decryption procedure:

\[ C_1 \cdots C_{2m} \parallel \text{tag} \leftarrow C \]
\[ X_0 \leftarrow 0^{2^n}; \ v \leftarrow 1 \]
for \( i = 1 \) to \( m \) do
\[ T \leftarrow (N, v) \]
\[ (X_i, M_{2i-1}M_{2i}) \leftarrow \text{Dec}^{E_K,E_K^{-1}}(T, X_{i-1}, C_{2i-1}C_{2i}) \]
\[ v \leftarrow v + \text{Cost}(\Pi) \]
\[ T \leftarrow (N, 1 - v) \]
\[ V \leftarrow \text{Tag}^{E_K}(T, X_m) \]
if \( \text{tag} \neq V[1, \tau] \) then return \( \perp \) else return \( M_1 \cdots M_{2m} \)
Idea of modeling modes as graphs extends to AE setting:
Idea of modeling modes as graphs extends to AE setting:

Graph representations of **Enc** (left), **Dec** (middle), and **Tag** (right) for OCB mode.
Each vertex label is a value $\text{type} \in \{\$, \perp, 0, 1\}$:

- $\$$: Random
- $\perp$: Adversarially controlled
- 0 and 1: Used to compare values on the same vertex in two runs of the Dec graph
  - 0: The two values are the same
  - 1: The two values are different
Constraints generally follow “intuition” and prior approach. E.g.,

- $\$ \oplus \perp \rightarrow \$ 
- $\text{TBC}(\$ \text{ or } 1) \rightarrow \$ 
- $\text{TBC}(0) \rightarrow 0$ 
- $\text{TBC}(\perp) \rightarrow \perp$
Verifying Privacy and Authenticity

To show: Validly labeled graph $\Rightarrow$ privacy and authenticity
To show: Validly labeled graph $\Rightarrow$ privacy and authenticity

Privacy:
- Outputs typed $\$$ when inputs typed $\perp$

Authenticity:
Recall security definition:
- Adversary has access to oracle $E(K, \cdot, \cdot)$ and must never repeat a nonce
- Adversary wins if it can output $(N, C)$ such that $D(K, N, C) \neq \perp$ and $C$ is not an oracle output

Consider forgery query $(N, C)$
- Want to show: Tag produced when running $D(K, N, C)$ is random
- $D(K, N, C) = \perp$ w.h.p.

Interesting Case:
- Suppose prior oracle query $(N, M)$ produced ciphertext $C'$ ($\neq C$)
- $C = C_1 \cdots C_m$ and $C' = C'_1 \cdots C'_m$ must differ in some block
- I.e., $C_i \neq C'_i$ for some $i$
To show: Validly labeled graph $\Rightarrow$ privacy and authenticity

Privacy:
- Outputs typed \$ when inputs typed \(\bot\)

Authenticity: Recall security definition:

Adversary has access to oracle \(\mathcal{E}(K, \cdot, \cdot)\) and must never repeat a nonce
Adversary wins if it can output \((N, C)\) such that \(\mathcal{D}(K, N, C) \neq \bot\) and \(C\) is not an oracle output
Verifying Privacy and Authenticity

To show: Validly labeled graph $\Rightarrow$ privacy and authenticity

Privacy:
- Outputs typed $\$ when inputs typed $\bot$

Authenticity: Recall security definition:

Adversary has access to oracle $E(K, \cdot, \cdot)$ and must never repeat a nonce
Adversary wins if it can output $(N, C)$ such that $D(K, N, C) \neq \bot$ and $C$ is not an oracle output

- Consider forgery query $(N, C)$
  - Want to show: Tag produced when running $D(K, N, C)$ is random $\Rightarrow D(K, N, C) = \bot$ w.h.p.
Verifying Privacy and Authenticity

To show: Validly labeled graph \( \Rightarrow \) privacy and authenticity

Privacy:
- Outputs typed \( \$ \) when inputs typed \( \bot \)

Authenticity: Recall security definition:

Adversary has access to oracle \( \mathcal{E}(K, \cdot, \cdot) \) and must never repeat a nonce. Adversary wins if it can output \((N, C)\) such that \( \mathcal{D}(K, N, C) \neq \bot \) and \( C \) is not an oracle output.

- Consider forgery query \((N, C)\)
  - Want to show: Tag produced when running \( \mathcal{D}(K, N, C) \) is random
    \( \Rightarrow \) \( \mathcal{D}(K, N, C) = \bot \) w.h.p.
  - Interesting Case: Suppose prior oracle query \((N, M)\) produced ciphertext \( C' \) (\( \neq C \))
    - \( C = C_1 \cdots C_m \) and \( C' = C'_1 \cdots C'_m \) must differ in some block
    - I.e., \( C_i \neq C'_i \) for some \( i \)
Verifying Privacy and Authenticity

To show: Validly labeled graph ⇒ privacy and authenticity

Privacy:
- Outputs typed $ when inputs typed ⊥

Authenticity: Recall security definition:

Adversary has access to oracle $\mathcal{E}(K, \cdot, \cdot)$ and must never repeat a nonce
Adversary wins if it can output $(N, C)$ such that $\mathcal{D}(K, N, C) \neq \bot$ and $C$ is not an oracle output

- Consider forgery query $(N, C)$
  - Want to show: Tag produced when running $\mathcal{D}(K, N, C)$ is random
    ⇒ $\mathcal{D}(K, N, C) = \bot$ w.h.p.
  - **Interesting Case:** Suppose prior oracle query $(N, M)$ produced ciphertext $C'$ ($\neq C$)
    - $C = C_1 \cdots C_m$ and $C' = C'_1 \cdots C'_m$ must differ in some block
      - i.e., $C_i \neq C'_i$ for some $i$
    - Want to show: This difference ⇒ tag is random
Verifying Authenticity

**Authenticity:**

1. Consider Dec graph on first two-block chunk $C_i C_{i+1}$ and $C'_i C'_{i+1}$ where $C$ and $C'$ differ

2. Consider Dec graph on blocks $C_i+2 C_{i+3}$ and $C'_i+2 C'_{i+3}$

3. Consider Tag graph

Given INI = $\$, check that FIN = $\$ \Rightarrow$ State is random after this point

Given INI = $\$, check that OUT = $\$ \Rightarrow$ Tag is random
Verifying Authenticity

Authenticity:

1. Consider Dec graph on first two-block chunk $C_i C_{i+1}$ and $C'_i C'_{i+1}$ where $C$ and $C'$ differ
   - Suppose $C_i \neq C'_i$:
     - Given $IN_1 = 0$, $IN_1 = 1$, and $IN_2 = 0$, check that $FIN = $
Verifying Authenticity

Authenticity:

1. Consider Dec graph on first two-block chunk $C_iC_{i+1}$ and $C'_iC'_{i+1}$ where $C$ and $C'$ differ
   - Suppose $C_i \neq C'_i$:
     - Given $INI = 0$, $IN_1 = 1$, and $IN_2 = 0$, check that $FIN = $
     - Similar checks for cases where (1) $C_{i+1} \neq C'_{i+1}$ and (2) both $C_i \neq C'_i$ and $C_{i+1} \neq C'_{i+1}$
Verifying Authenticity

Authenticity:

1. Consider Dec graph on first two-block chunk \( C_i C_{i+1} \) and \( C'_i C'_{i+1} \) where \( C \) and \( C' \) differ
   - Suppose \( C_i \neq C'_i \):
     - Given \( \text{INI} = 0 \), \( \text{IN}_1 = 1 \), and \( \text{IN}_2 = 0 \), check that \( \text{FIN} = $ \)
     - Similar checks for cases where (1) \( C_{i+1} \neq C'_{i+1} \) and (2) both \( C_i \neq C'_i \) and \( C_{i+1} \neq C'_{i+1} \)
   \( \Rightarrow \) State is random after this point
Verifying Authenticity

**Authenticity:**

1. Consider $\text{Dec}$ graph on first two-block chunk $C_iC_{i+1}$ and $C'_iC'_{i+1}$ where $C$ and $C'$ differ
   - Suppose $C_i \neq C'_i$:
     - Given $\text{INI} = 0$, $\text{IN}_1 = 1$, and $\text{IN}_2 = 0$, check that $\text{FIN} = $ \$
     - Similar checks for cases where (1) $C_{i+1} \neq C'_{i+1}$ and (2) both $C_i \neq C'_i$ and $C_{i+1} \neq C'_{i+1}$
   \[\Rightarrow\] State is random after this point

2. Consider $\text{Dec}$ graph on blocks $C_{i+2}C_{i+3}$ and $C'_{i+2}C'_{i+3}$
Verifying Authenticity

Authenticity:

1. Consider Dec graph on first two-block chunk $C_i C_{i+1}$ and $C'_i C'_{i+1}$ where $C$ and $C'$ differ
   - Suppose $C_i \neq C'_i$:
     - Given $INI = 0$, $IN_1 = 1$, and $IN_2 = 0$, check that $FIN = $ 
     - Similar checks for cases where (1) $C_{i+1} \neq C'_{i+1}$ and (2) both $C_i \neq C'_i$ and $C_{i+1} \neq C'_{i+1}$
   $\Rightarrow$ State is random after this point

2. Consider Dec graph on blocks $C_{i+2} C_{i+3}$ and $C'_{i+2} C'_{i+3}$
   - Given $INI = $, check that $FIN = $
Verifying Authenticity

Authenticity:

1. Consider Dec graph on first two-block chunk $C_i C_{i+1}$ and $C'_i C'_{i+1}$ where $C$ and $C'$ differ
   • Suppose $C_i \neq C'_i$:
     • Given $INI = 0$, $IN_1 = 1$, and $IN_2 = 0$, check that $FIN =$
     • Similar checks for cases where (1) $C_{i+1} \neq C'_{i+1}$ and (2) both $C_i \neq C'_i$ and $C_{i+1} \neq C'_{i+1}$
   $\Rightarrow$ State is random after this point

2. Consider Dec graph on blocks $C_{i+2} C_{i+3}$ and $C'_{i+2} C'_{i+3}$
   • Given $INI = $, check that $FIN =$
   $\Rightarrow$ State continues to be random
Verifying Authenticity

Authenticity:

1. Consider Dec graph on first two-block chunk $C_i C_{i+1}$ and $C'_i C'_{i+1}$ where $C$ and $C'$ differ
   - Suppose $C_i \neq C'_i$:
     - Given $INI = 0$, $IN_1 = 1$, and $IN_2 = 0$, check that $FIN = $
     - Similar checks for cases where (1) $C_{i+1} \neq C'_{i+1}$ and (2) both $C_i \neq C'_i$ and $C_{i+1} \neq C'_{i+1}$
   ⇒ State is random after this point

2. Consider Dec graph on blocks $C_{i+2} C_{i+3}$ and $C'_{i+2} C'_{i+3}$
   - Given $INI = $, check that $FIN = $
   ⇒ State continues to be random

3. Consider Tag graph
Verifying Authenticity

Authenticity:
1. Consider \textbf{Dec} graph on first two-block chunk $C_iC_{i+1}$ and $C'_iC'_{i+1}$ where $C$ and $C'$ differ
   - Suppose $C_i \neq C'_i$:
     - Given $\text{INI} = 0$, $\text{IN}_1 = 1$, and $\text{IN}_2 = 0$, check that $\text{FIN} = $
     - Similar checks for cases where (1) $C_{i+1} \neq C'_{i+1}$ and (2) both $C_i \neq C'_i$ and $C_{i+1} \neq C'_{i+1}$
   \Rightarrow State is random after this point
2. Consider \textbf{Dec} graph on blocks $C_{i+2}C_{i+3}$ and $C'_{i+2}C'_{i+3}$
   - Given $\text{INI} = $, check that $\text{FIN} = $
   \Rightarrow State \textit{continues} to be random
3. Consider \textbf{Tag} graph
   - Given $\text{INI} = $, check that $\text{OUT} = $
   \Rightarrow Tag is random
Verifying Authenticity: Example

Recall OCB:

\[
\begin{align*}
& M_1 \quad M_2 \quad M_3 \quad M_4 \quad \Sigma \\
& E_K^{N,1} \quad E_K^{N,2} \quad E_K^{N,3} \quad E_K^{N,4} \quad E_K^{N,-4} \\
& C_1 \quad C_2 \quad C_3 \quad C_4 \quad \text{tag} \quad \tau
\end{align*}
\]

Suppose two queries are:

- \( C = C_1 C_2 C_3 C_4 \parallel \text{tag} \)
- \( C' = C_1 C_2' C_3 C_4 \parallel \text{tag} \)

Need to show:

1. Dec graph with \( \text{INI} = 0 \), \( \text{IN}_1 = 0 \), \( \text{IN}_2 = 1 \) \( \Rightarrow \) \( \text{FIN} = \$ \)
2. Dec graph with \( \text{INI} = \$ \), \( \text{IN}_1 = 0 \), \( \text{IN}_2 = 0 \) \( \Rightarrow \) \( \text{FIN} = \$ \)
3. Tag graph with \( \text{INI} = \$ \) \( \Rightarrow \) \( \text{OUT} = \$ \)
Recall OCB:

Suppose two queries are:

\[ C = C_1 C_2 C_3 C_4 \parallel \text{tag} \]
\[ C' = C_1 C'_2 C_3 C_4 \parallel \text{tag} \]
Verifying Authenticity: Example

Recall OCB:

Suppose two queries are:

\[ C = C_1 C_2 C_3 C_4 \| \text{tag} \]

\[ C' = C_1 C'_2 C_3 C_4 \| \text{tag} \]

Need to show:

1. Dec graph with INI = 0, IN_1 = 0, IN_2 = 1 ⇒ FIN = $
Verifying Authenticity: Example

Recall OCB:

Suppose two queries are:

\[ C = C_1 C_2 C_3 C_4 \parallel \text{tag} \]
\[ C' = C_1 C'_2 C_3 C_4 \parallel \text{tag} \]

Need to show:

1. **Dec** graph with \( \text{INI} = 0, \text{IN}_1 = 0, \text{IN}_2 = 1 \Rightarrow \text{FIN} = \$ \)
2. **Dec** graph with \( \text{INI} = \$, \text{IN}_1 = 0, \text{IN}_2 = 0 \Rightarrow \text{FIN} = \$ \)
Verifying Authenticity: Example

Recall OCB:

\[
M_1 \\ E_K^{N,1} \\ C_1 \\
M_2 \\ E_K^{N,2} \\ C_2 \\
M_3 \\ E_K^{N,3} \\ C_3 \\
M_4 \\ E_K^{N,4} \\ C_4 \\
\Sigma \\
\tau \\
tag
\]

Suppose two queries are:

\[
C = C_1 C_2 C_3 C_4 \| \text{tag} \\
C' = C_1 C_2' C_3 C_4 \| \text{tag}
\]

Need to show:

1. **Dec** graph with INI = 0, IN\(_1\) = 0, IN\(_2\) = 1 \(\Rightarrow\) FIN = $
2. **Dec** graph with INI = $, IN\(_1\) = 0, IN\(_2\) = 0 \(\Rightarrow\) FIN = $
3. **Tag** graph with INI = $ \(\Rightarrow\) OUT = $
Implemented model checker + synthesizer in OCaml

Model Checker:

Checks whether input mode (given as Enc graph) is secure
• Need to generate Dec graph from Enc graph to check privacy
• Checks simple enough that SMT solver not needed!
• Also include check for parallelizability of mode

Synthesizer:

As before, can simply iterate over all possible graphs
• Use various techniques to reduce search space
Ran model checker for modes with $\leq 16$ instructions

<table>
<thead>
<tr>
<th># Nodes</th>
<th>Secure</th>
<th>“Optimal”</th>
<th>Parallel</th>
<th>Running Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>13</td>
<td>13</td>
<td>5</td>
<td>1 min</td>
</tr>
<tr>
<td>13</td>
<td>142</td>
<td>0</td>
<td>0</td>
<td>5.5 min</td>
</tr>
<tr>
<td>14</td>
<td>583</td>
<td>172</td>
<td>5</td>
<td>30.5 min</td>
</tr>
<tr>
<td>15</td>
<td>2229</td>
<td>39</td>
<td>5</td>
<td>2 hours*</td>
</tr>
<tr>
<td>16</td>
<td>2123</td>
<td>35</td>
<td>1</td>
<td>2 hours*</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>5090</strong></td>
<td><strong>259</strong></td>
<td><strong>16</strong></td>
<td></td>
</tr>
</tbody>
</table>
Synthesis Results

Ran model checker for modes with \( \leq 16 \) instructions

<table>
<thead>
<tr>
<th># Nodes</th>
<th>Secure</th>
<th>“Optimal”</th>
<th>Parallel</th>
<th>Running Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>13</td>
<td>13</td>
<td>5</td>
<td>1 min</td>
</tr>
<tr>
<td>13</td>
<td>142</td>
<td>0</td>
<td>0</td>
<td>5.5 min</td>
</tr>
<tr>
<td>14</td>
<td>583</td>
<td>172</td>
<td>5</td>
<td>30.5 min</td>
</tr>
<tr>
<td>15</td>
<td>2229</td>
<td>39</td>
<td>5</td>
<td>2 hours*</td>
</tr>
<tr>
<td>16</td>
<td>2123</td>
<td>35</td>
<td>1</td>
<td>2 hours*</td>
</tr>
<tr>
<td>Total</td>
<td>5090</td>
<td>259</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>

Among 5 parallel modes of size 12, only one previously known (OCB)
Synthesis Results: Example Schemes
Implemented these (three) schemes, and compared with running time of OCB (fastest known AE scheme):

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Enc</th>
<th>Dec</th>
</tr>
</thead>
<tbody>
<tr>
<td>OCB</td>
<td>0.5864 ± 0.0036</td>
<td>0.6333 ± 0.0012</td>
</tr>
<tr>
<td>1</td>
<td>0.5991 ± 0.0037</td>
<td>0.6233 ± 0.0029</td>
</tr>
<tr>
<td>2</td>
<td>0.5915 ± 0.0011</td>
<td>0.6326 ± 0.0012</td>
</tr>
<tr>
<td>3</td>
<td>0.6616 ± 0.0011</td>
<td>2.2855 ± 0.0023</td>
</tr>
</tbody>
</table>

Results in cycles per bytes with 95% confidence intervals over 100 runs of each scheme.
Synthesis Results: Example Schemes

Implemented these (three) schemes, and compared with running time of OCB (fastest known AE scheme):

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Enc</th>
<th>Dec</th>
</tr>
</thead>
<tbody>
<tr>
<td>OCB</td>
<td>0.5864 ± 0.0036</td>
<td>0.6333 ± 0.0012</td>
</tr>
<tr>
<td>1</td>
<td>0.5991 ± 0.0037</td>
<td>0.6233 ± 0.0029</td>
</tr>
<tr>
<td>2</td>
<td>0.5915 ± 0.0011</td>
<td>0.6326 ± 0.0012</td>
</tr>
<tr>
<td>3</td>
<td>0.6616 ± 0.0011</td>
<td>2.2855 ± 0.0023</td>
</tr>
</tbody>
</table>

Results in cycles per bytes with 95% confidence intervals over 100 runs of each scheme

Synthesized schemes competitive with OCB!
Conclusion

1. Introduced method for reasoning about *modes of operation*
   - Meta-theorem: Validly labeled mode is secure
     ⇒ Can use SMT solver to *automatically* prove modes secure
1. Introduced method for reasoning about *modes of operation*
   - Meta-theorem: Validly labeled mode is secure
     \[\Rightarrow\] Can use SMT solver to *automatically* prove modes secure

2. Extend to reasoning about (class of) *authenticated encryption schemes*
   - Uses same graph idea
   - **Key insight:** Use *tweakable blockciphers* to simplify analysis
   - Captures (simplified variant of) many modes: OCB, XCBC, COPA, OTR, CCM
   - Synthesized new schemes as efficient as fastest known AE scheme
1. Introduced method for reasoning about *modes of operation*
   - Meta-theorem: Validly labeled mode is secure
     \[\Rightarrow\text{Can use SMT solver to} \text{ automatically prove modes secure}\]

2. Extend to reasoning about (class of) *authenticated encryption schemes*
   - Uses same graph idea
   - **Key insight:** Use *tweakable blockciphers* to simplify analysis
   - Captures (simplified variant of) many modes: OCB, XCBC, COPA, OTR, CCM
   - Synthesized new schemes as efficient as fastest known AE scheme

3. **Open Problem:** Can we apply this approach to other primitives?
Thank You

Any questions?

Full Versions:
Modes of operation: http://eprint.iacr.org/2014/774
AE schemes: To appear shortly

Code:
Modes of operation: https://github.com/amaloz/modes-generator
AE schemes: https://github.com/amaloz/ae-generator