Automated Analysis and Synthesis of Block-Cipher Modes of Operation

Alex J. Malozemoff$^1$  Jonathan Katz$^1$  Matthew D. Green$^2$

$^1$University of Maryland
$^2$Johns Hopkins University

Presented at the Fall Protocol eXchange, National Cryptologic Museum, Fort Meade, Maryland, USA, September 23–24, 2014.
Problem: Designing/proving crypto constructions is hard!
Problem: Designing/proving crypto constructions is hard!

(Possible) Solution: Use ideas from *program synthesis* to automate the design/proof of crypto schemes
Introduction

Problem: Designing/proving crypto constructions is hard!

(Possible) Solution: Use ideas from program synthesis to automate the design/proof of crypto schemes

Program Synthesis

- Automatically construct programs based on (small) set of rules
- Has been applied to crypto protocols (e.g., [AGHP12, BCG+13])
Problem: Designing/proving crypto constructions is hard!

(Possible) Solution: Use ideas from *program synthesis* to automate the design/proof of crypto schemes

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

**This Work:** Apply program synthesis to *modes of operation*
Background: Modes of Operation

Block-Cipher (\(=\) PRP, \(F_k\)): Encrypts \emph{fixed-length} message (e.g., AES)
Background: Modes of Operation

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

**Mode of Operation**: encrypts *arbitrary-length* messages, using block-cipher as building block
Background: Modes of Operation

**Block-Cipher (= PRP, $F_k$):** Encrypts *fixed-length* message (e.g., AES)

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

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

Want output of mode to look “random” to adversary ⇒ IND$-CPA

**What is IND$-CPA?**

Adversary $A$ has oracle access to either
- (World 1) a truly random function
- (World 2) the desired mode of operation

$A$ specifies messages to encrypt and receives resulting ciphertexts

*$A$’s Goal:* Decide whether in World 1 or World 2

*Secure:* $A$ cannot distinguish between worlds
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 1) a truly random function
- (World 2) the desired mode of operation

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

\( \mathcal{A} \)'s Goal: Decide whether in World 1 or World 2

Secure: \( \mathcal{A} \) cannot distinguish between worlds

**Note:** Explains why ECB mode (encrypt each message block by PRP) is insecure
Motivation

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
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
This Work

Model (single block of) mode as \textit{directed, acyclic graph}

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

Each edge assigned \textit{label}

- \textit{Constraints} restrict how edges can be labeled
This Work

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

Meta-Theorem: Exists valid labeling $\implies$ mode IND$\$-CPA-secure

Note: Our approach analyzes a constant size graph, yet proves security on arbitrary (polynomial) length inputs
Several prior works look at automatically analyzing modes:

- **Gagné et al. [GLLSN09, GLLSN12]:**
  - Modes described in imperative language
  - Use *compositional Hoare logic* to analyze security
  - **Drawback:** Can only reason about encryption of messages of pre-specified length

- **Courant et al. [CEL07]:**
  - Use *type system* to analyze security of modes, among others
  - **Drawback:** Similar to above

Our approach works for *arbitrary (polynomial) length* messages
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
Viewing Modes as Graphs

Init algorithm

Block algorithm
Edge Labels: Intuition

Recall: Edges denote intermediate values
Recall: Edges denote intermediate values

Intuition: Labels should capture “properties” of intermediate value
  • Does value look random to adversary?
  • Can value be output as ciphertext?
    • Only “random-looking” values should be output
  • etc.
Edge Labels: Intuition

Recall: Edges denote intermediate values

Intuition: Labels should capture “properties” of intermediate value
  • Does value look random to adversary?
  • Can value be output as ciphertext?
    • Only “random-looking” values should be output
  • etc.

Goal: If values on edges into OUT nodes look random to adversary, then mode is IND$-CPA-secure
Each edge label is a 3-tuple \((\text{fam}, \text{type}, \text{flags})\):

- **fam**: See later slide...
- **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 \(\text{OUT}\) or \(\text{PRP}\)
  - Prevents values being both output as part of ciphertext and input to \(\text{PRP}\)
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.$OUT = 1$

- **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**
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 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
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
- **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 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.$OUT = 1$
What About the \textbf{fam} Variable?

\textbf{Note:} No tracking of which intermediate values “related” to other intermediate values
What About the **fam** Variable?

**Note:** No tracking of which intermediate values “related” to other intermediate values

Consider the following graph:
What About the **fam** Variable?

**Note:** No tracking of which intermediate values “related” to other intermediate values

Consider the following graph:

![Graph](image)

The output edge is labeled \((R, 11)\) but the actual value is zero!
What About the \textbf{fam} Variable?

The \textbf{fam} variable tracks “related” edges
What About the **fam** Variable?

The **fam** variable tracks “related” edges

\[ \text{fam} \subseteq \{1, \ldots\} : \text{Set of families to which edge belongs} \]
What About the **fam** Variable?

The **fam** variable tracks “related” edges

**fam** \( \subseteq \{1, \ldots \} \): Set of *families* to which edge belongs

Two edges \( e_1, e_2 \) are *related* if \( \text{fam}_1 \cap \text{fam}_2 \neq \emptyset \)
What About the \textbf{fam} Variable?

The \textbf{fam} variable tracks “related” edges

\textbf{fam} \subseteq \{1, \ldots \}: Set of \textit{families} to which edge belongs

Two edges \(e_1, e_2\) are \textit{related} if \(\text{fam}_1 \cap \text{fam}_2 \neq \emptyset\)

New constraint on \textbf{XOR}: incoming edges cannot be related
What About the \textbf{fam} Variable?

The \textbf{fam} variable tracks “related” edges

\textbf{fam} \subseteq \{1, \ldots \}: Set of \textit{families} to which edge belongs

Two edges \( e_1, e_2 \) are \textit{related} if \( \text{fam}_1 \cap \text{fam}_2 \neq \emptyset \)

New constraint on \textbf{XOR}: incoming edges cannot be related

\[
\begin{align*}
\text{GENRAND} & \quad \{\{1\}, R, 11\} \\
\text{DUP} & \quad \{\{1\}, R, 10\}, \{\{1\}, R, 01\} \\
\text{XOR} & \quad x
\end{align*}
\]
Want to prove: Exists valid labeling $\implies$ mode is IND$\$-CPA-secure

Proof (high level): By induction:
  - $\mathcal{A}$ inputs $m = m_1 \| \ldots \| m_\ell$ to mode
Meta-Theorem

Want to prove: Exists valid labeling $\Rightarrow$ mode is IND$\$-CPA-secure

Proof (high level): By induction:

- $\mathcal{A}$ inputs $m = m_1 \parallel \ldots \parallel m_{\ell}$ to mode
- Let $G$ be connected graph containing one copy of $\text{Init}$ and $\ell$ copies of $\text{Block}$
Meta-Theorem

Want to prove: Exists valid labeling $\Rightarrow$ mode is IND$\$-CPA-secure

Proof (high level): By induction:

- $\mathcal{A}$ inputs $m = m_1\| \ldots \| m_\ell$ to mode
- Let $G$ be connected graph containing one copy of $\text{Init}$ and $\ell$ copies of $\text{Block}$
- Consider assigning values to edges in topological order $\text{step-by-step}$
Want to prove: Exists valid labeling $\Rightarrow$ mode is IND$_\text{S}$-CPA-secure

Proof (high level): By induction:

- $\mathcal{A}$ inputs $m = m_1 \parallel \ldots \parallel m_\ell$ to mode
- Let $G$ be connected graph containing one copy of $\text{Init}$ and $\ell$ copies of $\text{Block}$
- Consider assigning values to edges in topological order step-by-step
- $\text{OUT}$: set of edges on ingoing edges to $\text{OUT}$ nodes in $G$
Meta-Theorem

Want to prove: Exists valid labeling $\Rightarrow$ mode is IND$^\dagger$-CPA-secure

Proof (high level): By induction:
- $A$ inputs $m = m_1 \| \ldots \| m_\ell$ to mode
- Let $G$ be connected graph containing one copy of $\text{Init}$ and $\ell$ copies of $\text{Block}$
- Consider assigning values to edges in topological order step-by-step
- $\text{OUT}$: set of edges on ingoing edges to $\text{OUT}$ nodes in $G$
- Let $\text{val}$ be function mapping edges to values
Meta-Theorem

Want to prove: Exists valid labeling $\Rightarrow$ mode is IND$\$-CPA-secure

Proof (high level): By induction:

- $\mathcal{A}$ inputs $m = m_1 || \ldots || m_\ell$ to mode
- Let $G$ be connected graph containing one copy of $\text{Init}$ and $\ell$ copies of $\text{Block}$
- Consider assigning values to edges in topological order \textit{step-by-step}
- $\text{OUT}$: set of edges on ingoing edges to $\text{OUT}$ nodes in $G$
- Let $\text{val}$ be function mapping edges to values
- \textbf{Invariant 1}: At each step, values in $\text{val}(\text{OUT})$ are uniformly random
  - $\Rightarrow$ Output looks random to $\mathcal{A}$
  - $\Rightarrow$ Proving Invariant 1 proves theorem!
Proof (continued):

- Need additional invariants to prove Invariant 1...
Proof (continued):

- Need additional invariants to prove Invariant 1.
- Edge is *active* if it has been assigned a value but its children have not
Proof (continued):

- Need additional invariants to prove Invariant 1...
- Edge is active if it has been assigned a value but its children have not
- $PRP_a$: set of active edges of type R with flags. $PRP = 1$
Meta-Theorem

Proof (continued):

• Need additional invariants to prove Invariant 1...
• Edge is *active* if it has been assigned a value but its children have not
• \( PRP_a \): set of active edges of type \( R \) with flags. \( PRP = 1 \)
• Invariant 2: Values in \( \text{val}(PRP_a) \) are jointly uniform, even conditioned on prior inputs to \( PRP \)
  • Intuition: Enforces that inputs into \( PRP \) nodes are uniformly random
Meta-Theorem

Proof (continued):

- Need additional invariants to prove Invariant 1...
- Edge is active if it has been assigned a value but its children have not
- \( PRP_a \): set of active edges of type \( R \) with flags. \( PRP = 1 \)
- Invariant 2: Values in \( \text{val}(PRP_a) \) are jointly uniform, even conditioned on prior inputs to \( PRP \)
  - Intuition: Enforces that inputs into \( PRP \) nodes are uniformly random
- \( OUT_a \): set of active edges of type \( R \) with flags. \( OUT = 1 \)
Proof (continued):

- Need additional invariants to prove Invariant 1...
- Edge is *active* if it has been assigned a value but its children have not
- $PRP_a$: set of active edges of type $R$ with flags. $PRP = 1$
- Invariant 2: Values in $\text{val}(PRP_a)$ are jointly uniform, even conditioned on prior inputs to $PRP$
  - Intuition: Enforces that inputs into $PRP$ nodes are uniformly random
- $OUT_a$: set of active edges of type $R$ with flags. $OUT = 1$
- Invariant 3: Values in $\text{val}(OUT_a)$ are jointly uniform, even conditioned on prior inputs to $OUT$
  - Intuition: Enforces that inputs into $OUT$ nodes are uniformly random
Meta-Theorem

Proof (continued):

• Need additional invariants to prove Invariant 1...
• Edge is active if it has been assigned a value but its children have not
• $PRP_a$: set of active edges of type $R$ with flags. $PRP = 1$
• Invariant 2: Values in $val(PRP_a)$ are jointly uniform, even conditioned on prior inputs to $PRP$
  • Intuition: Enforces that inputs into $PRP$ nodes are uniformly random
• $OUT_a$: set of active edges of type $R$ with flags. $OUT = 1$
• Invariant 3: Values in $val(OUT_a)$ are jointly uniform, even conditioned on prior inputs to $OUT$
  • Intuition: Enforces that inputs into $OUT$ nodes are uniformly random

Final Step: Considering each node type, prove (by induction) that Invariants hold

• See paper for details...
What About CTR Mode?

\[ F_k \]

\[ m_1 \rightarrow c_1 \]
\[ m_2 \rightarrow c_2 \]
\[ m_3 \rightarrow c_3 \]
What About CTR Mode?

Need to add **INC** instruction: increments value by 1

⇒ Need new type (\(U\) ⇒ “unique”), flag bits

⇒ Requires additional constraints

⇒ Complicates proof, but is possible (see paper)
Implemented model checker + synthesizer in OCaml

**Model Checker:**

1. Checks whether an input mode is secure
   - **Recall:** Valid labeling $\implies$ mode is secure
   - $\implies$ Determining secure mode is a constraint-satisfaction problem
   - $\implies$ 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
(1) Encoding Example: **DUP**

Recall: Edge label: *(fam, type, flags)*, flags.**OUT**, flags.**PRP**

**DUP** rule: outgoing edges inherit ingoing edge’s type, split flag bits

---

**Z3 Encoding:**

```lisp
(declare-const dup_l_type Int)
(declare-const dup_l_flag_out Bool)
(declare-const dup_l_flag_prp Bool)
(declare-const dup_r_type Int)
(declare-const dup_r_flag_out Bool)
(declare-const dup_r_flag_prp Bool)
(assert (= dup_l_type dup_r_type genrand_type))
(assert (= (and dup_l_flag_out dup_r_flag_out) false))
(assert (= (and dup_l_flag_prp dup_r_flag_prp) false))
(assert (= (or dup_l_flag_out dup_r_flag_out) genrand_flag_out))
(assert (= (or dup_l_flag_prp dup_r_flag_prp) genrand_flag_prp))
```
(1) Encoding Example: DUP

Recall: Edge label: 
(fam, type, flags), flags.**OUT**, flags.**PRP**

**DUP** rule: outgoing edges inherit ingoing edge's type, split flag bits

Z3 Encoding:

(declare-const dup_l_type Int)  
(declare-const dup_l_flag_out Bool)  
(declare-const dup_l_flag_prp Bool)  
(declare-const dup_r_type Int)  
(declare-const dup_r_flag_out Bool)  
(declare-const dup_r_flag_prp Bool)  
(assert (= dup_l_type dup_r_type genrand_type))  
(assert (= (and dup_l_flag_out dup_r_flag_out) false))  
(assert (= (and dup_l_flag_prp dup_r_flag_prp) false))  
(assert (= (or dup_l_flag_out dup_r_flag_out) genrand_flag_out))  
(assert (= (or dup_l_flag_prp dup_r_flag_prp) genrand_flag_prp))
(1) Encoding Example: **DUP**

Recall: Edge label: 
(fam, type, flags), flags.**OUT**, flags.**PRP**

**DUP** rule: outgoing edges inherit ingoing edge’s type, split flag bits

**Z3 Encoding:**

(declare-const dup_l_type Int)
(declare-const dup_l_flag_out Bool)
(declare-const dup_l_flag_prp Bool)
(declare-const dup_r_type Int)
(declare-const dup_r_flag_out Bool)
(declare-const dup_r_flag_prp Bool)
(assert (= dup_l_type dup_r_type genrand_type))
(assert (= (and dup_l_flag_out dup_r_flag_out) false))
(assert (= (and dup_l_flag_prp dup_r_flag_prp) false))
(assert (= (or dup_l_flag_out dup_r_flag_out) genrand_flag_out))
(assert (= (or dup_l_flag_prp dup_r_flag_prp) genrand_flag_prp))
Encoding Example: DUP

Recall: Edge label:

(fam, type, flags),
flags.\textit{OUT}, flags.\textit{PRP}

\textbf{DUP} rule: outgoing edges inherit ingoing edge’s type, split flag bits

\textbf{Z3 Encoding:}

(declare-const dup_l_type Int)
(declare-const dup_l_flag_out Bool)
(declare-const dup_l_flag_prp Bool)
(declare-const dup_r_type Int)
(declare-const dup_r_flag_out Bool)
(declare-const dup_r_flag_prp Bool)
(assert (= dup_l_type dup_r_type genrand_type))
(assert (= (and dup_l_flag_out dup_r_flag_out) false))
(assert (= (and dup_l_flag_prp dup_r_flag_prp) false))
(assert (= (or dup_l_flag_out dup_r_flag_out) genrand_flag_out))
(assert (= (or dup_l_flag_prp dup_r_flag_prp) genrand_flag_prp))
(2) Checking Decryptability

Given ciphertext, can we recover message?
(2) Checking Decryptability

Given ciphertext, can we recover message?

**Three Steps:**
Given ciphertext, can we recover message?

Three Steps:

Step 1: Given values of outgoing edge of **START** and incoming edge to **OUT** in **Block**, can we recover **M**?
- I.e., given ciphertext block and previous state info, can we recover plaintext block?

Step 2: Given value of incoming edge to **OUT** in **Init**, can we recover **NEXTIV**?
- I.e., given ciphertext, can we recover state info (in **Init**)?

Step 3: Given value of incoming edge to **OUT** in **Block**, can we recover **NEXTIV**?
- I.e., given ciphertext, can we recover state info (in **Block**)?
(2) Checking Decryptability

Given ciphertext, can we recover message?

**Three Steps:**

**Step 1:** Given values of outgoing edge of **START** and incoming edge to **OUT** in **Block**, can we recover **M**?
- I.e., given ciphertext block and previous state info, can we recover plaintext block?

**Step 2:** Given value of incoming edge to **OUT** in **Init**, can we recover **NEXTIV**?
- I.e., given ciphertext, can we recover state info (in **Init**)?
Given ciphertext, can we recover message?

**Three Steps:**

**Step 1:** Given values of outgoing edge of START and incoming edge to OUT in Block, can we recover $M$?
  - I.e., given ciphertext block and previous state info, can we recover plaintext block?

**Step 2:** Given value of incoming edge to OUT in Init, can we recover NEXTIV?
  - I.e., given ciphertext, can we recover state info (in Init)?

**Step 3:** Given value of incoming edge to OUT in Block, can we recover NEXTIV?
  - I.e., given ciphertext, can we recover state info (in Block)?
Step 1: Given values of outgoing edge of \textbf{START} and incoming edge to \textbf{OUT} in \textbf{Block}, can we recover \textbf{M}?
(2) Checking Decryptability

Step 1: Given values of outgoing edge of START and incoming edge to OUT in Block, can we recover M?
(2) Checking Decryptability

Step 1: Given values of outgoing edge of **START** and incoming edge to **OUT** in **Block**, can we recover **M**?
(2) Checking Decryptability

**Step 1:** Given values of outgoing edge of **START** and incoming edge to **OUT** in **Block**, can we recover **M**?

![Diagram](image)

- **GENRAND**
- **DUP**
- **OUT**
- **NEXTIV**
- **START**
- **M**
- **XOR**
- **PRP**
- **DUP**
- **OUT**
- **NEXTIV**

Init algorithm

Block algorithm
(2) Checking Decryptability

**Step 2:** Given value of incoming edge to **OUT** in **Init**, can we recover **NEXTIV**?
Step 2: Given value of incoming edge to OUT in Init, can we recover NEXTIV?
(2) Checking Decryptability

**Step 3:** Given value of incoming edge to **OUT** in **Block**, can we recover **NEXTIV**?
(2) Checking Decryptability

**Step 3:** Given value of incoming edge to \textbf{OUT} in \textbf{Block}, can we recover \textbf{NEXTIV}?
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>559</td>
<td>282</td>
<td>20</td>
</tr>
<tr>
<td>9</td>
<td>3544</td>
<td>1361</td>
<td>87</td>
</tr>
<tr>
<td>10</td>
<td>8862</td>
<td>2101</td>
<td>243</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>13015</td>
<td>3774</td>
<td><strong>355</strong></td>
</tr>
</tbody>
</table>

**Note:** Numbers subject to change (bug in decryptability checker currently being fixed)
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>559</td>
<td>282</td>
<td>20</td>
</tr>
<tr>
<td>9</td>
<td>3544</td>
<td>1361</td>
<td>87</td>
</tr>
<tr>
<td>10</td>
<td>8862</td>
<td>2101</td>
<td>243</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>13015</td>
<td>3774</td>
<td><strong>355</strong></td>
</tr>
</tbody>
</table>

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

Note: Numbers subject to change (bug in decryptability checker currently being fixed)
Conclusion

Introduced method for reasoning about modes of operation

- Uses only “local” analysis of single block

Meta-theorem: Validly labeled mode is secure

- Can use SMT solver to *automatically* prove modes secure

**Future Work:**

- Handle additional operations (field operations, etc)
- Combine with EasyCrypt for (1) further security assurances and (2) concrete security bounds
- Can similar approach work for message authentication codes (authenticity), authenticated encryption (confidentiality *and* authenticity), etc?
Any questions?

E-mail: amaloz@cs.umd.edu
URL: https://www.cs.umd.edu/~amaloz
Code: https://github.com/amaloz/modes-generator
Full Version: Coming soon on https://eprint.iacr.org