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.

Introduction

Problem: Designing/proving crypto constructions is hard

Introduction

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 *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

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

Background: Modes of Operation

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

Background: Modes of Operation

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

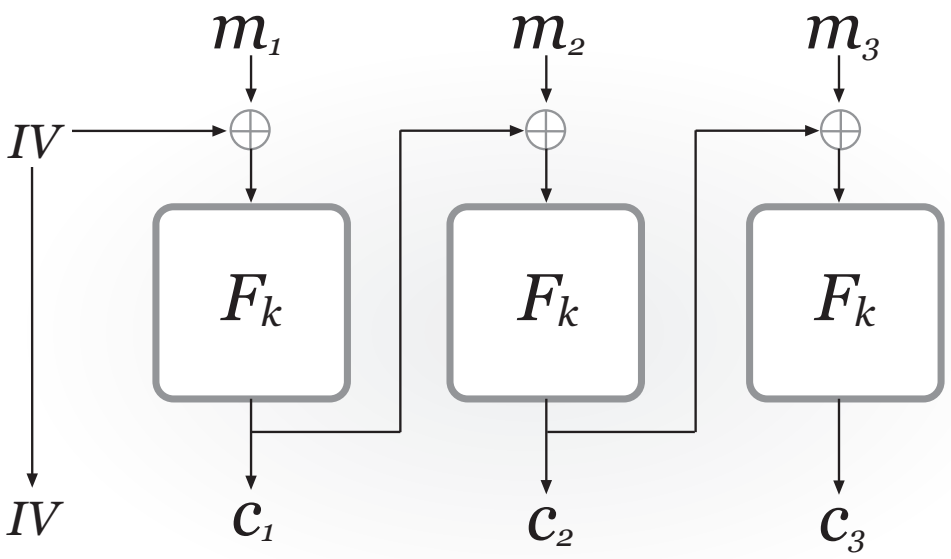
Mode of Operation: encrypts *arbitrary-length* messages, using blockcipher as building block

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

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

Modes of Operation

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

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

Modes of Operation

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

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

Each edge assigned *label*

- *Constraints* restrict how edges can be labeled

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

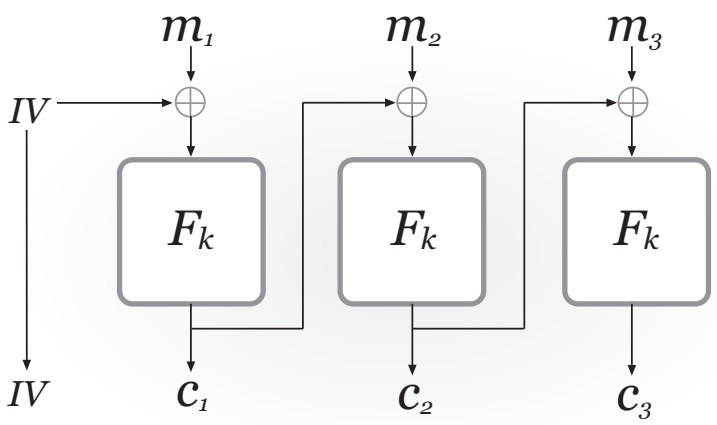
Mode of Operation: Formal Definition

Defined by two algorithms:

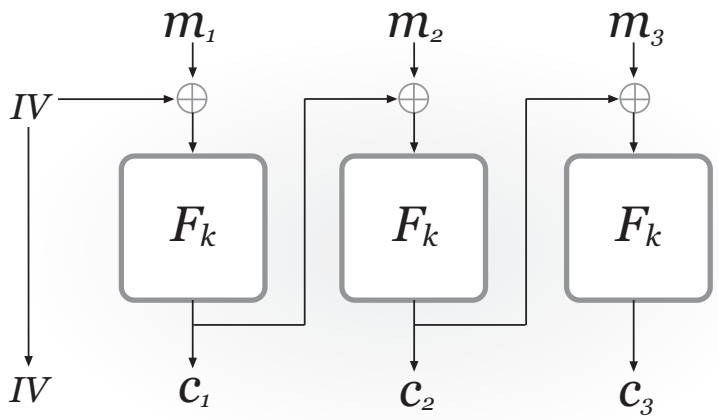
- $\mathbf{Init}(1^n) \to (c_0, z_0)$
- $\mathbf{Block}(m_i, z_{i-1}) \to (c_i, z_i)$

$\mathbf{Enc}_k(m = m_1 \| \cdots \| m_\ell)$:

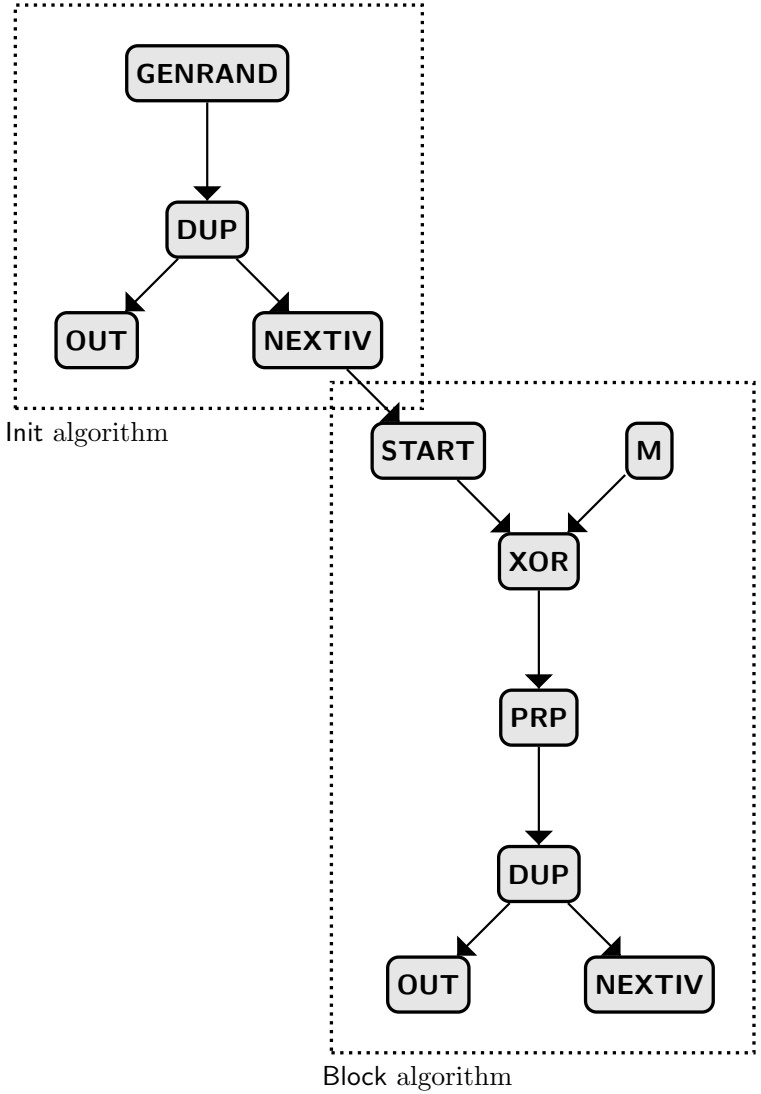
- Compute $(c_0, z_0) \leftarrow \mathbf{Init}(1^n)$
- For $i = 1, \ldots, \ell$:
  Compute $(c_i, z_i) \leftarrow \mathbf{Block}(m_i, z_{i-1})$
- Output $c_0 \| \cdots \| c_\ell$



Viewing Modes as Graphs



$\Rightarrow$



Edge Labels: Intuition

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?

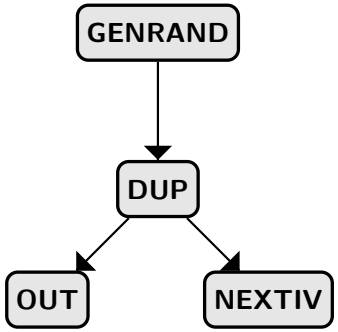
Goal: If values on edges into **OUT** nodes look random to adversary, then mode is IND$-CPA-secure

Edge Labels: Formalism (simplified)

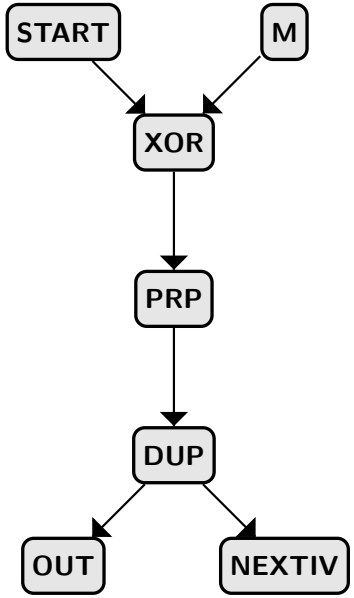
Each edge label is a tuple $\mathbf{(type, flags)}$:

- $\mathbf{type} \in \{\perp, \mathbf{R}\}$: "Type" of intermediate value
  - $\perp$: Adversarially controlled
  - **R**: Random
- $\mathbf{flags} \in \{\mathbf{0}, \mathbf{1}\}^{\mathbf{2}}$: Bit-vector denoting whether edge can be input into **OUT** or **PRP**
  - E.g., prevents both **DUP**'d values being output as ciphertext

Constraints



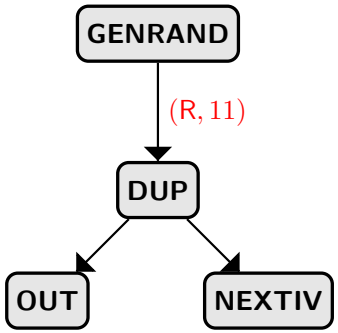
Init algorithm



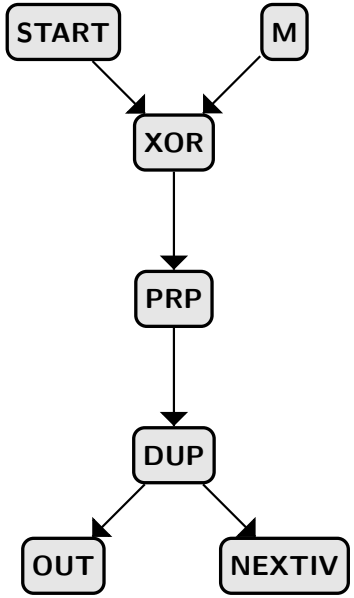
Block algorithm

Constraints on Nodes:

Constraints



Init algorithm

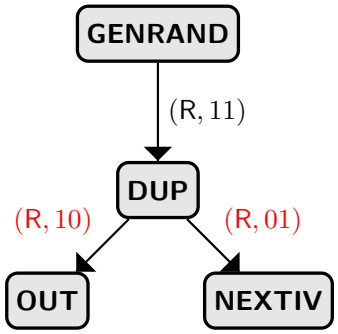


Block algorithm

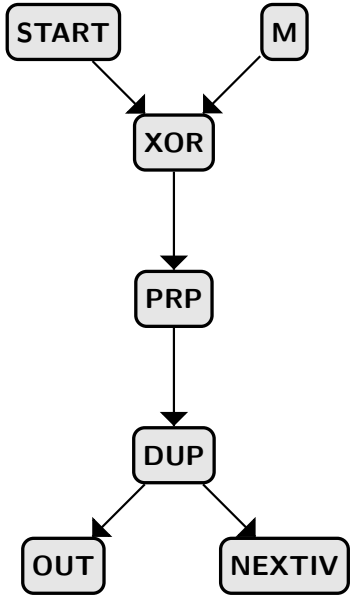
Constraints on Nodes:

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

Constraints



Init algorithm

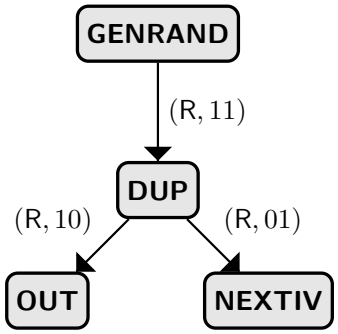


Block algorithm

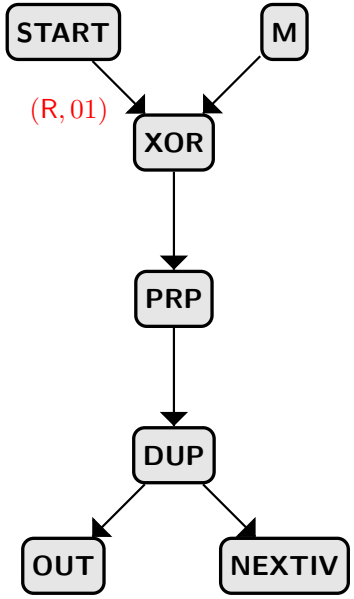
Constraints on Nodes:

- **GENRAND**: Outgoing edge gets type **R**, **flags.*PRP* = 1**, **flags.*OUT* = 1**
- **DUP**: Outgoing edges inherit ingoing edge's type, split flag bits

Constraints



Init algorithm

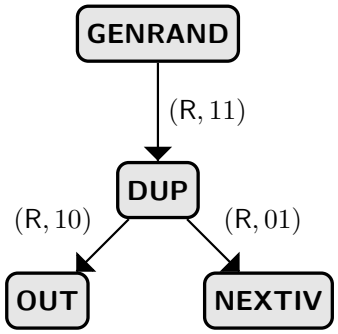


Block algorithm

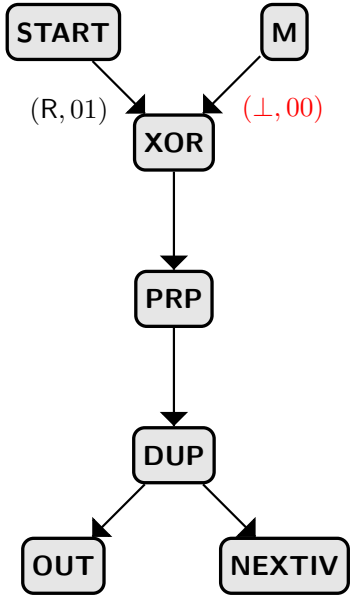
Constraints on Nodes:

- **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**

Constraints



Init algorithm

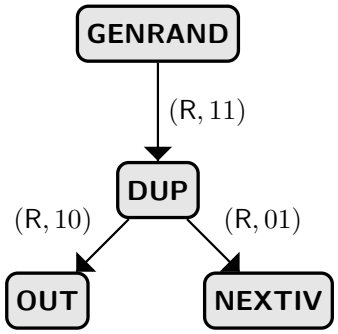


Block algorithm

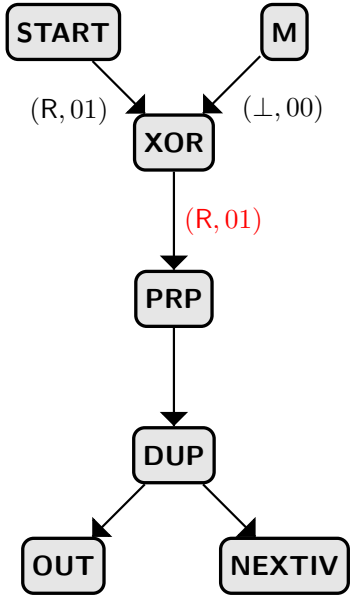
Constraints on Nodes:

- **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 ⊥, **flags.*PRP* = 0**, **flags.*OUT* = 0**

Constraints



Init algorithm

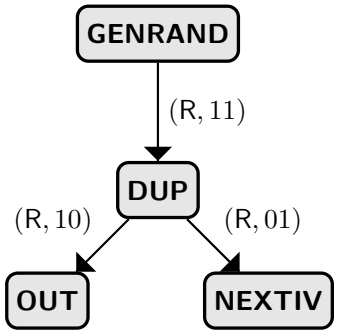


Block algorithm

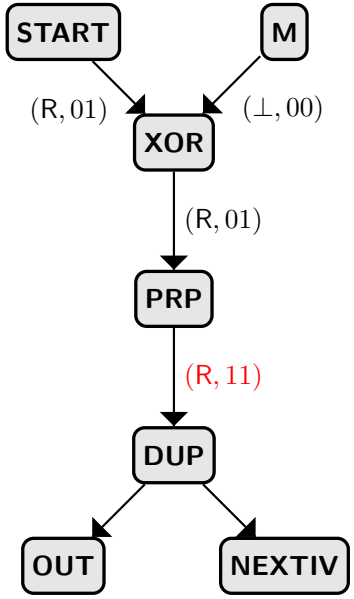
Constraints on Nodes:

- **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 ⊥, **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



Init algorithm

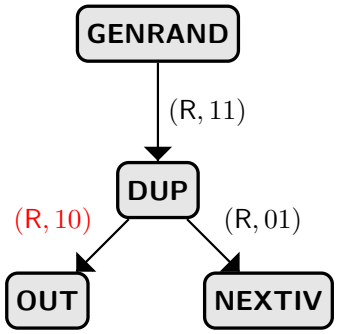


Block algorithm

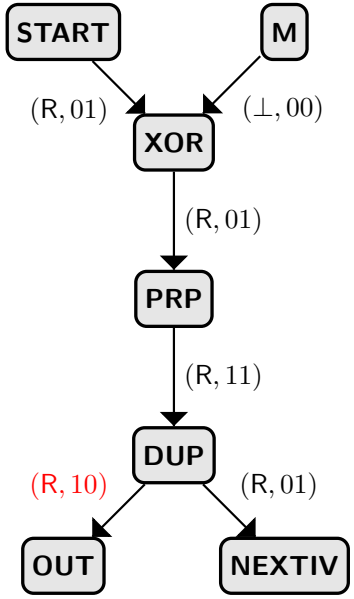
Constraints on Nodes:

- **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 ⊥, **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



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 ⊥, **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**

Implementation

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

Synthesis Results

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

# Instructions	Valid	Decryptable	Secure
1–6	0	0	0
7	50	30	5
8	549	281	31
9	3130	1304	150
10	5107	1770	184
Total	8836	3383	**370**

Synthesis Results

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

# Instructions	Valid	Decryptable	Secure
1–6	0	0	0
7	50	30	5
8	549	281	31
9	3130	1304	150
10	5107	1770	184
Total	8836	3383	**370**

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

Background: Authenticated Encryption

AE scheme: Tuple of algorithms $\mathbf{\Pi} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$, 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

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"):

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

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"):

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

Simplifies analysis:

- If tweak is unique, output of TBC is random

Background: Tweakable Blockciphers

We restrict ourselves to AE schemes using tweakable blockciphers

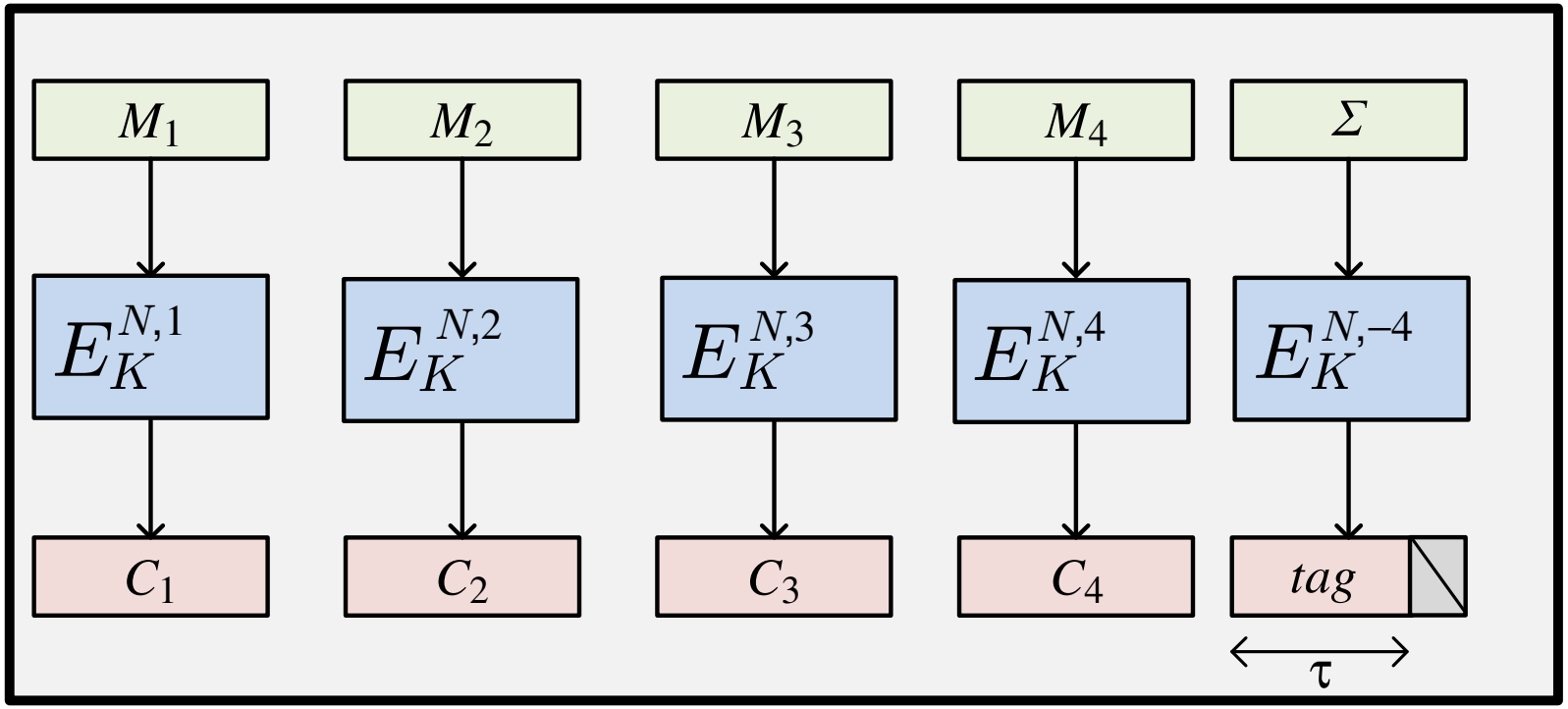
Tweakable Blockcipher (TBC): Like (standard) blockcipher, except takes an extra argument (the "tweak"):

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

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



Authenticated Encryption: Formal Definition

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

- $\mathbf{Enc}^{E_K}(T, X, M_1 M_2) \rightarrow (Y, C_1 C_2)$
- $\mathbf{Dec}^{E_K, E_K^{-1}}(T, Y, C_1 C_2) \rightarrow (X, M_1 M_2)$
- $\mathbf{Tag}^{E_K}(T, X) \rightarrow V$

Authenticated Encryption: Formal Definition

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

- $\mathbf{Enc}^{E_K}(T, X, M_1 M_2) \to (Y, C_1 C_2)$
- $\mathbf{Dec}^{E_K, E_K^{-1}}(T, Y, C_1 C_2) \to (X, M_1 M_2)$
- $\mathbf{Tag}^{E_K}(T, X) \to V$

Encryption procedure:

$$
\begin{array}{l}
X_0 \leftarrow 0^{2n};\ v \leftarrow 1;\ M_1 \cdots M_{2m} \leftarrow M \\
\textbf{for } i = 1 \textbf{ to } m \textbf{ do} \\
\quad T \leftarrow (N, v) \\
\quad (X_i, C_{2i-1} C_{2i}) \leftarrow \mathbf{Enc}^{E_K}(T, X_{i-1}, M_{2i-1} M_{2i}) \\
\quad v \leftarrow v + \mathbf{Cost}(\Pi) \\
T \leftarrow (N, 1 - v) \\
V \leftarrow \mathbf{Tag}^{E_K}(T, X) \\
\textbf{return } C_1 \cdots C_{2m} \parallel V[1, \tau]
\end{array}
$$

Authenticated Encryption: Formal Definition

Decryption procedure:

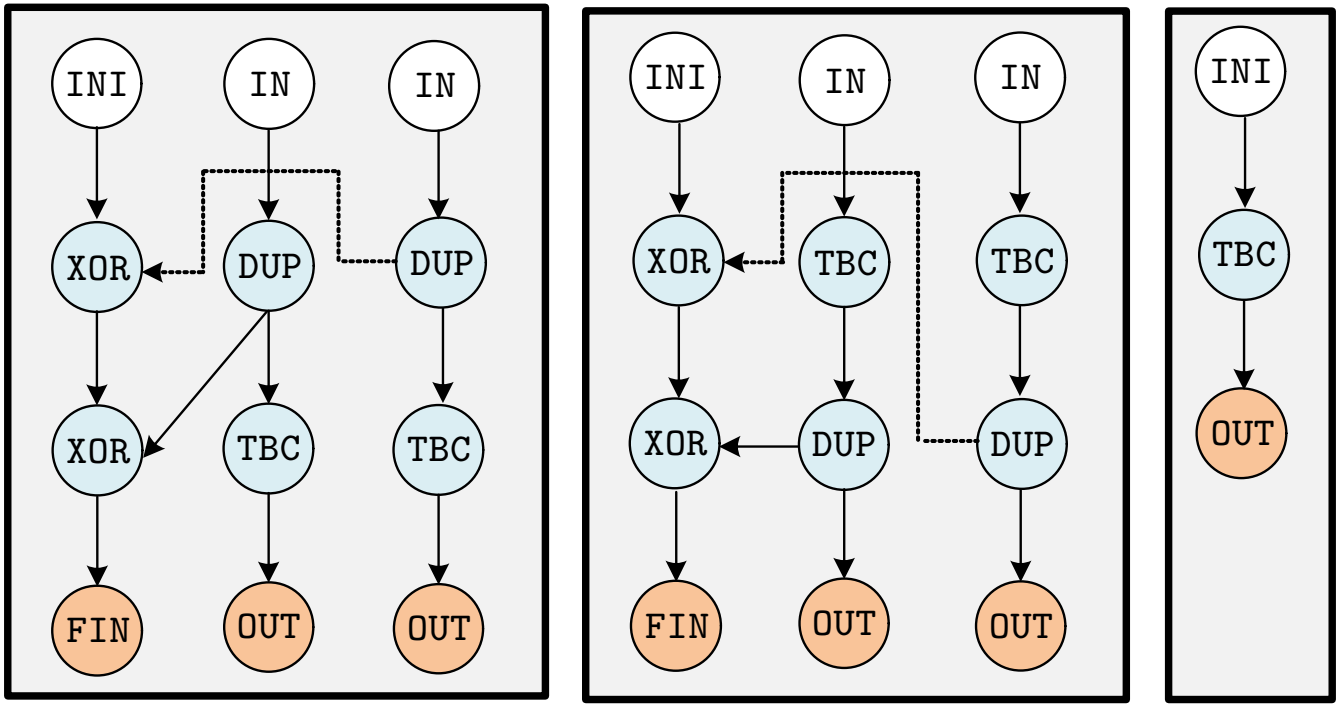
$$
\begin{array}{l}
C_1 \cdots C_{2m} \parallel \mathrm{tag} \leftarrow C \\
X_0 \leftarrow 0^{2n};\ v \leftarrow 1 \\
\textbf{for } i = 1 \textbf{ to } m \textbf{ do} \\
\quad T \leftarrow (N, v) \\
\quad (X_i, M_{2i-1}M_{2i}) \leftarrow \mathrm{Dec}^{E_K, E_K^{-1}}(T, X_{i-1}, C_{2i-1}C_{2i}) \\
\quad v \leftarrow v + \mathrm{Cost}(\Pi) \\
T \leftarrow (N, 1 - v) \\
V \leftarrow \mathrm{Tag}^{E_K}(T, X_m) \\
\textbf{if } \mathrm{tag} \neq V[1, \tau] \textbf{ then return } \perp \textbf{ else return } M_1 \cdots M_{2m}
\end{array}
$$

Viewing AE Schemes as Graphs

Idea of modeling modes as graphs extends to AE setting:

Viewing AE Schemes as Graphs

Idea of modeling modes as graphs extends to AE setting:



Graph representations of **Enc** (left), **Dec** (middle), and **Tag** (right) for OCB mode.

Vertex Labels

Each vertex label is a value **type** $\in \{\mathbf{\$}, \perp, \mathbf{0}, \mathbf{1}\}$:

- **$**: 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

Constraints generally follow "intuition" and prior approach. E.g.,

- $\mathbf{\$} \oplus \bot \rightarrow \mathbf{\$}$
- TBC$(\mathbf{\$}$ or $\mathbf{1}) \rightarrow \mathbf{\$}$
- TBC$(\mathbf{0}) \rightarrow \mathbf{0}$
- TBC$(\bot) \rightarrow \bot$

Verifying Privacy and Authenticity

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

Verifying Privacy and Authenticity

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

Privacy:

- Outputs typed **$** when inputs typed $\perp$

Verifying 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 $\mathcal{E}(K, \cdot, \cdot)$ and *must never repeat a nonce*

Adversary wins if it can output (N, C) such that $\mathcal{D}(K, N, C) \neq \perp$ 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 $\perp$

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 $\mathcal{D}(K, N, C) \neq \perp$ 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

  $\Rightarrow$ $\mathcal{D}(K, N, C) = \perp$ w.h.p.

Verifying 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 $\mathcal{E}(K, \cdot, \cdot)$ and *must never repeat a nonce*

Adversary wins if it can output (N, C) such that $\mathcal{D}(K, N, C) \neq \perp$ 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

  $\Rightarrow$ $\mathcal{D}(K, N, C) = \perp$ 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

Verifying 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 $\mathcal{E}(K, \cdot, \cdot)$ and *must never repeat a nonce*

Adversary wins if it can output (N, C) such that $\mathcal{D}(K, N, C) \neq \perp$ 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

  $\Rightarrow$ $\mathcal{D}(K, N, C) = \perp$ 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
  - Want to show: This difference $\Rightarrow$ tag is random

Verifying Authenticity

Authenticity:

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

Verifying Authenticity

Authenticity:

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

- Suppose $\boldsymbol{C_i \neq C'_i}$:
  - Given $\mathbf{INI} = \mathbf{0}$, $\mathbf{IN_1} = \mathbf{1}$, and $\mathbf{IN_2} = \mathbf{0}$, check that $\mathbf{FIN} = \mathbf{\$}$

Verifying Authenticity

Authenticity:

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

- Suppose $\boldsymbol{C_i \neq C'_i}$:
  - Given $\mathbf{INI} = \mathbf{0}$, $\mathbf{IN_1} = \mathbf{1}$, and $\mathbf{IN_2} = \mathbf{0}$, check that $\mathbf{FIN} = \mathbf{\$}$
- Similar checks for cases where (1) $\boldsymbol{C_{i+1} \neq C'_{i+1}}$ and (2) both $\boldsymbol{C_i \neq C'_i}$ and $\boldsymbol{C_{i+1} \neq C'_{i+1}}$

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 $\mathbf{INI} = \mathbf{0}$, $\mathbf{IN}_1 = \mathbf{1}$, and $\mathbf{IN}_2 = \mathbf{0}$, check that $\mathbf{FIN} = \$$
- 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 **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 $\mathbf{INI} = \mathbf{0}$, $\mathbf{IN}_1 = \mathbf{1}$, and $\mathbf{IN}_2 = \mathbf{0}$, check that $\mathbf{FIN} = \$$
- 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 $C'_{i+2} C'_{i+3}$

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 $\mathbf{INI} = \mathbf{0}$, $\mathbf{IN}_1 = \mathbf{1}$, and $\mathbf{IN}_2 = \mathbf{0}$, check that $\mathbf{FIN} = \mathbf{\$}$
- 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 $\mathbf{INI} = \mathbf{\$}$, check that $\mathbf{FIN} = \mathbf{\$}$

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 $\mathbf{INI} = \mathbf{0}$, $\mathbf{IN}_1 = \mathbf{1}$, and $\mathbf{IN}_2 = \mathbf{0}$, check that $\mathbf{FIN} = \mathbf{\$}$
- 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 $\mathbf{INI} = \mathbf{\$}$, check that $\mathbf{FIN} = \mathbf{\$}$

$\Rightarrow$ 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 $\mathbf{INI} = \mathbf{0}$, $\mathbf{IN}_1 = \mathbf{1}$, and $\mathbf{IN}_2 = \mathbf{0}$, check that $\mathbf{FIN} = \mathbf{\$}$
- 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 $\mathbf{INI} = \mathbf{\$}$, check that $\mathbf{FIN} = \mathbf{\$}$

$\Rightarrow$ State *continues* to be random

3. Consider **Tag** graph

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 $\mathbf{INI} = \mathbf{0}$, $\mathbf{IN}_1 = \mathbf{1}$, and $\mathbf{IN}_2 = \mathbf{0}$, check that $\mathbf{FIN} = \mathbf{\$}$
- 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 $\mathbf{INI} = \mathbf{\$}$, check that $\mathbf{FIN} = \mathbf{\$}$

$\Rightarrow$ State *continues* to be random

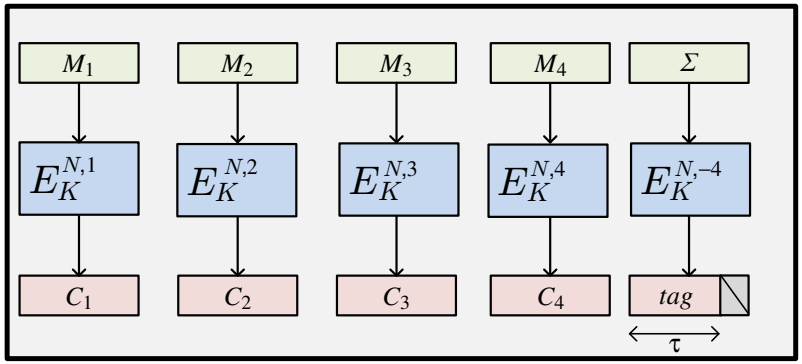
3. Consider **Tag** graph

- Given $\mathbf{INI} = \mathbf{\$}$, check that $\mathbf{OUT} = \mathbf{\$}$

$\Rightarrow$ Tag is random

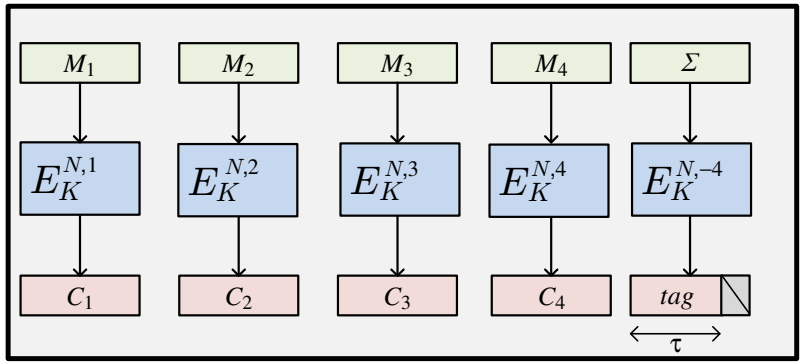
Verifying Authenticity: Example

Recall OCB:



Verifying Authenticity: Example

Recall OCB:

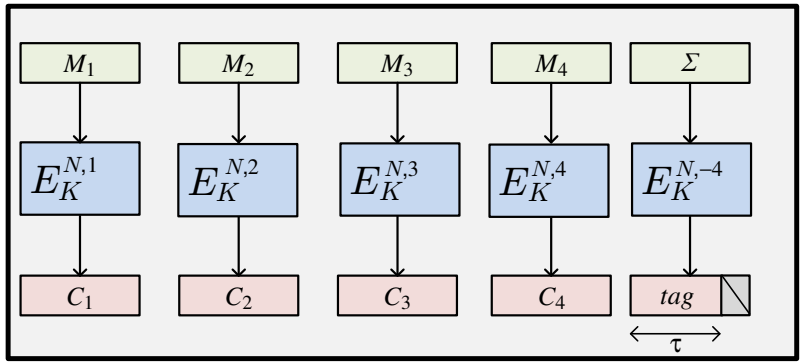


Suppose two queries are:

$$C = C_1 C_2 C_3 C_4 \| \mathrm{tag}$$
$$C' = C_1 C_2' C_3 C_4 \| \mathrm{tag}$$

Verifying Authenticity: Example

Recall OCB:



Suppose two queries are:

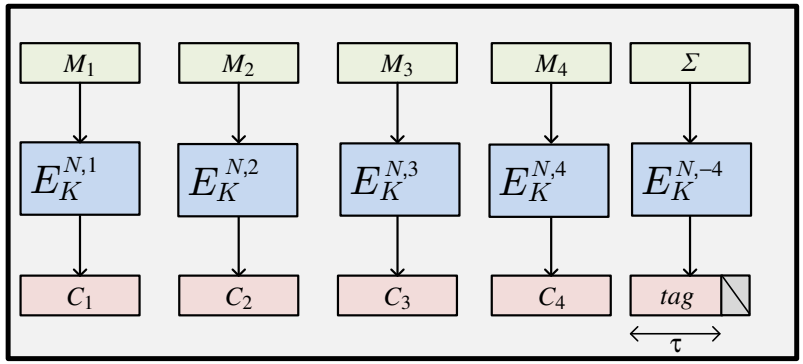
$$C = C_1 C_2 C_3 C_4 \| \mathrm{tag}$$
$$C' = C_1 C_2' C_3 C_4 \| \mathrm{tag}$$

Need to show:

1. **Dec** graph with $\mathbf{INI} = 0$, $\mathbf{IN_1} = 0$, $\mathbf{IN_2} = 1 \Rightarrow \mathbf{FIN} = \$$

Verifying Authenticity: Example

Recall OCB:



Suppose two queries are:

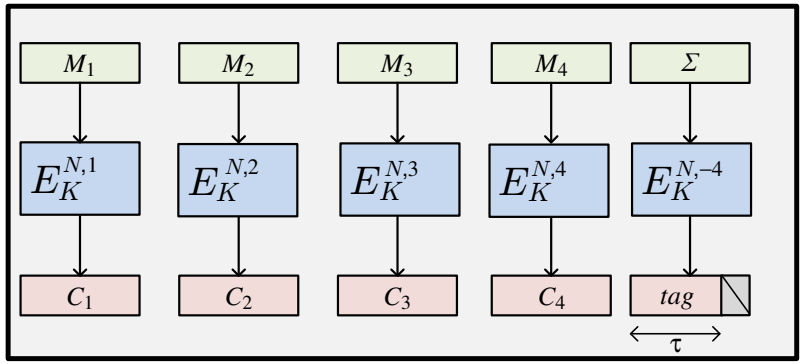
$$C = C_1 C_2 C_3 C_4 \| \mathrm{tag}$$
$$C' = C_1 C_2' C_3 C_4 \| \mathrm{tag}$$

Need to show:

1. **Dec** graph with $\mathbf{INI} = 0$, $\mathbf{IN_1} = 0$, $\mathbf{IN_2} = 1 \Rightarrow \mathbf{FIN} = \$$
2. **Dec** graph with $\mathbf{INI} = \$$, $\mathbf{IN_1} = 0$, $\mathbf{IN_2} = 0 \Rightarrow \mathbf{FIN} = \$$

Verifying Authenticity: Example

Recall OCB:



Suppose two queries are:

$$C = C_1 C_2 C_3 C_4 \| \mathrm{tag}$$
$$C' = C_1 C_2' C_3 C_4 \| \mathrm{tag}$$

Need to show:

1. **Dec** graph with $\mathbf{INI} = 0$, $\mathbf{IN_1} = 0$, $\mathbf{IN_2} = 1 \Rightarrow \mathbf{FIN} = \$$
2. **Dec** graph with $\mathbf{INI} = \$$, $\mathbf{IN_1} = 0$, $\mathbf{IN_2} = 0 \Rightarrow \mathbf{FIN} = \$$
3. **Tag** graph with $\mathbf{INI} = \$ \Rightarrow \mathbf{OUT} = \$$

Implementation

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

Synthesis Results

Ran model checker for modes with $\leq$ **16** instructions

# Nodes	Secure	"Optimal"	Parallel	Running Time
12	13	13	5	1 min
13	142	0	0	5.5 min
14	583	172	5	30.5 min
15	2229	39	5	2 hours*
16	2123	35	1	2 hours*
Total	5090	259	16	

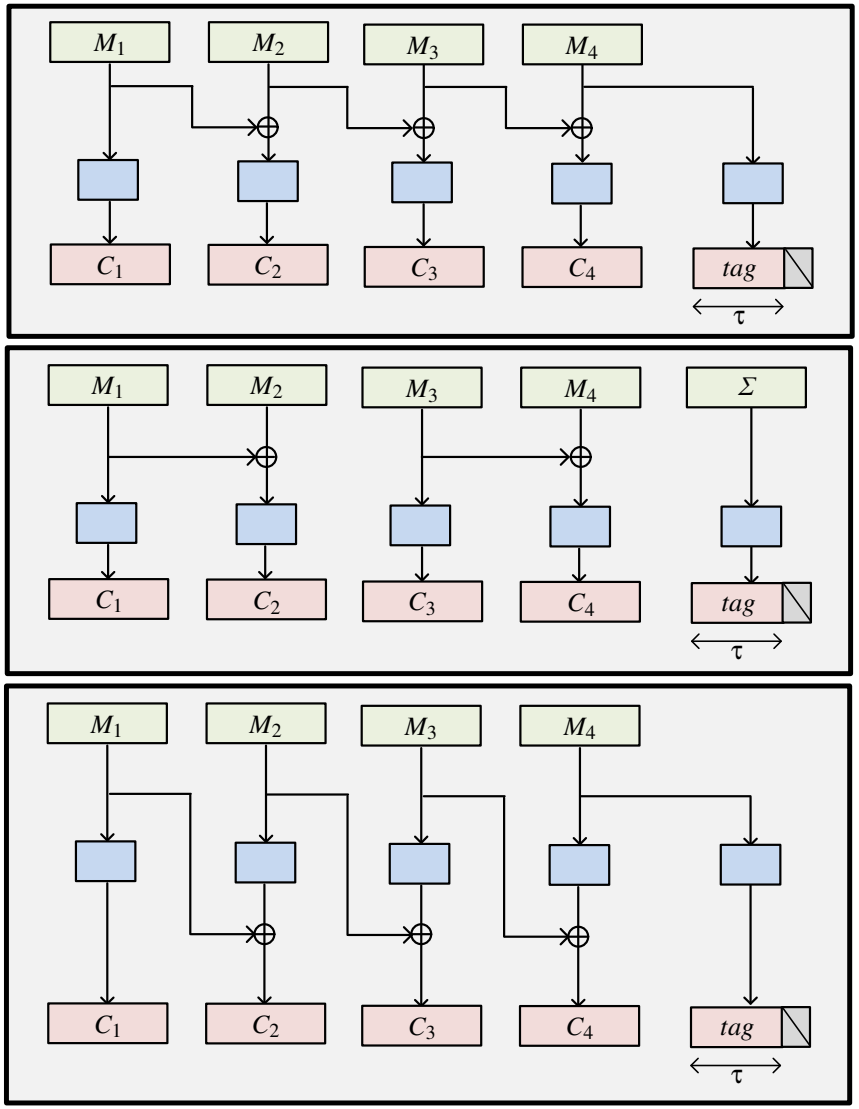
Synthesis Results

Ran model checker for modes with $\leq$ **16** instructions

# Nodes	Secure	"Optimal"	Parallel	Running Time
12	13	13	5	1 min
13	142	0	0	5.5 min
14	583	172	5	30.5 min
15	2229	39	5	2 hours*
16	2123	35	1	2 hours*
Total	5090	259	16	

Among **5** parallel modes of size **12**, only one previously known (OCB)

Synthesis Results: Example Schemes



Synthesis Results: Example Schemes

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

Scheme	**Enc**	**Dec**
OCB	0.5864 ± 0.0036	0.6333 ± 0.0012
1	0.5991 ± 0.0037	0.6233 ± 0.0029
2	0.5915 ± 0.0011	0.6326 ± 0.0012
3	0.6616 ± 0.0011	2.2855 ± 0.0023

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):

Scheme	**Enc**	**Dec**
OCB	0.5864 ± 0.0036	0.6333 ± 0.0012
1	0.5991 ± 0.0037	0.6233 ± 0.0029
2	0.5915 ± 0.0011	0.6326 ± 0.0012
3	0.6616 ± 0.0011	2.2855 ± 0.0023

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

   $\Rightarrow$ Can use SMT solver to *automatically* prove modes secure

Conclusion

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

Conclusion

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
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`