Automated Analysis and Synthesis of Authenticated Encryption Schemes

Viet Tung Hoang
University of Maryland
Georgetown University

Jonathan Katz
University of Maryland

Alex J. Malozemoff
University of Maryland

ACM CCS 2015
October 13, 2015

Authenticated Encryption (AE)

Sender → **Receiver**

K K

Privacy **Authenticity**

Encryption scheme

Authenticated Encryption:
Achieve **both** of these aims

Message Authentication Code (MAC)

Authenticated Encryption (AE)

- Lots of AE schemes
  - OCB, CCM, CAESAR candidates, etc.

Authenticated Encryption (AE)

- Lots of AE schemes
  - **OCB**, CCM, CAESAR candidates, etc.

    ✓ Most efficient
    ✘ Patented

Authenticated Encryption (AE)

- Lots of AE schemes
  - OCB, **CCM**, CAESAR candidates, etc.

✓ Not patented
✘ Slower than OCB

Authenticated Encryption (AE)

- Lots of AE schemes
  - OCB, CCM, **CAESAR candidates**, etc.

Ongoing competition
for new AE standard
$\rightarrow$ Active area of research

Authenticated Encryption (AE)

- Lots of AE schemes
  - OCB, CCM, CAESAR candidates, etc.

- Developing new AE schemes is hard
  - Complex, error-prone proofs

More systematic way to build secure AE schemes?

Our approach

based on *tweakable blockciphers*

Automatically ***analyze*** and ***synthesize*** AE schemes

Extend [MalozemoffKatzGreen14], which analyzed / synthesized *encryption modes of operation*

Captures many existing schemes:
OCB, XCBC, COPA, OTR, CCM, etc.

Our approach

Step 1: View AE scheme as graphs

- **Nodes**: operation (e.g., E_K, $\oplus$)
- **Edges**: Intermediate values

Step 2: Construct type system for graphs

- Typed graphs $\implies$ secure AE scheme

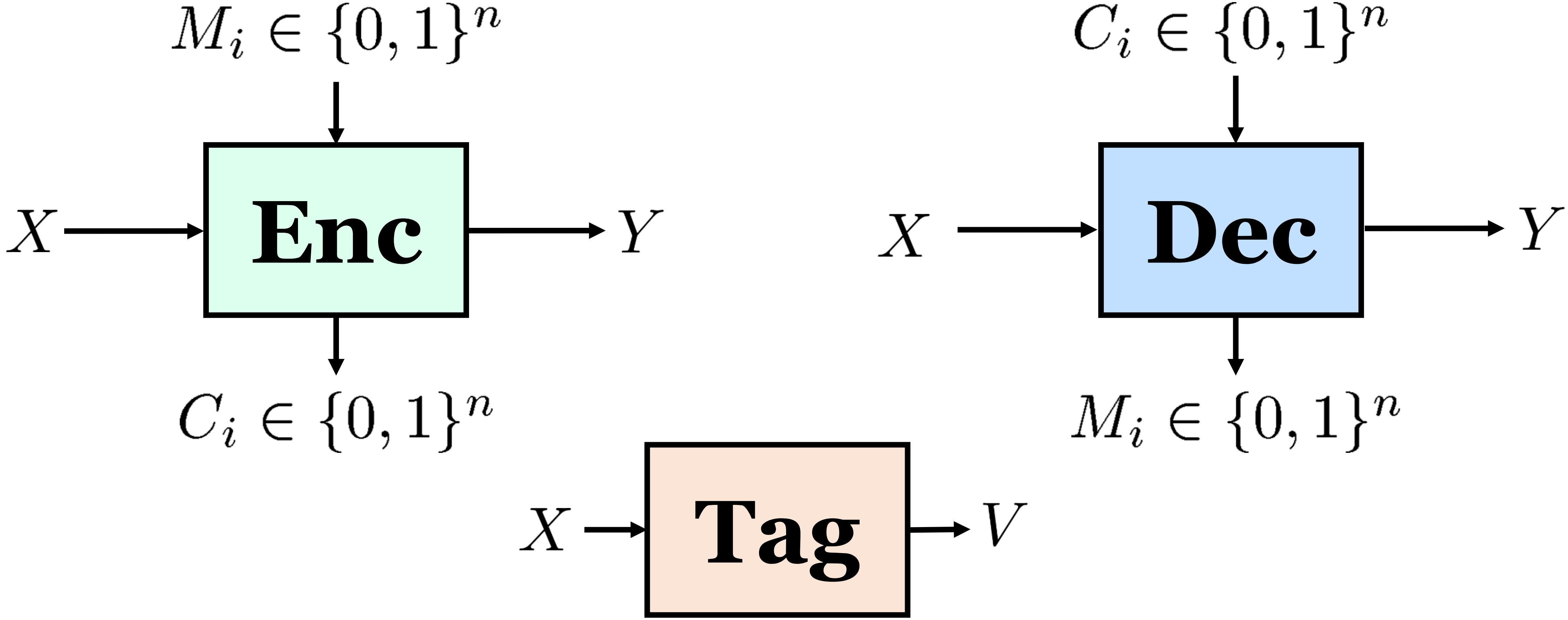
Step 3: Synthesize AE schemes using type system

Outline of rest of talk

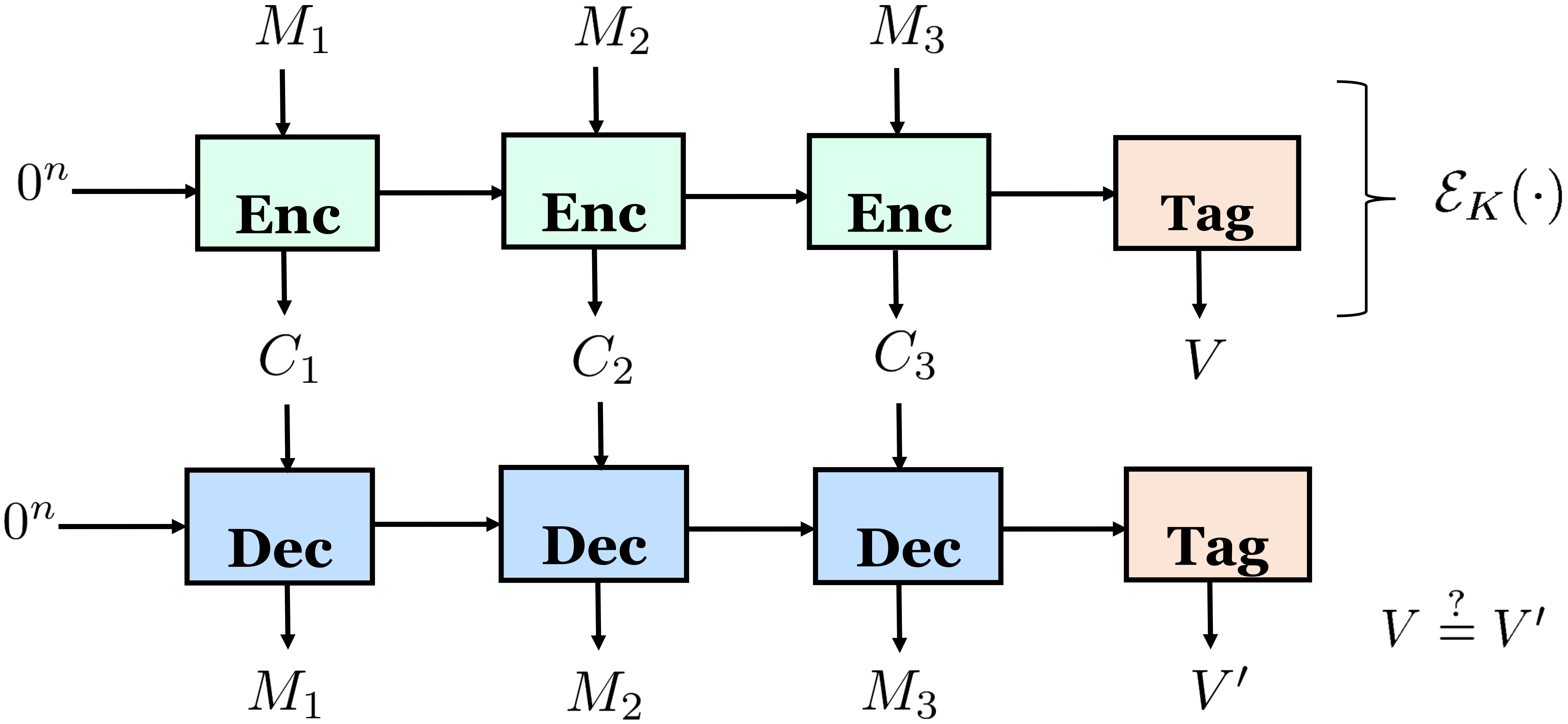
1. Template for AE schemes
2. Viewing AE schemes as graphs
3. Type system for graphs
4. Implementation + results

Template for AE schemes

AE scheme defined by three algorithms:
Enc, **Dec**, **Tag**



Template for AE schemes

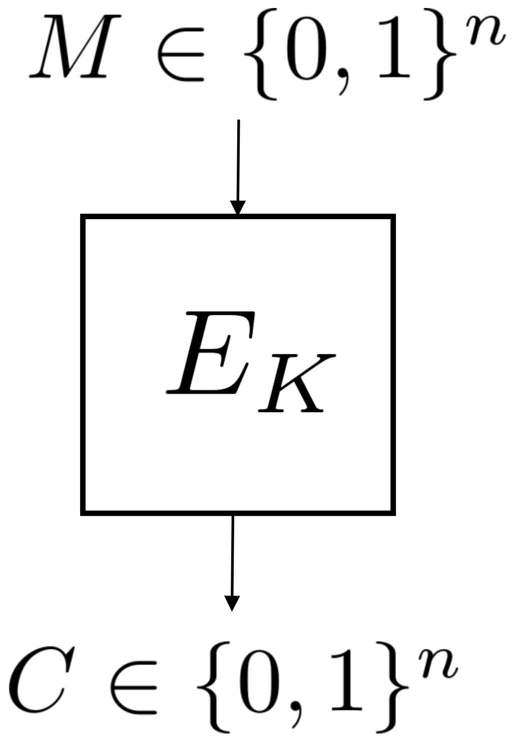


Note: Ignoring nonce/associated data

Restricted class of AE schemes

Consider AE schemes using *tweakable blockciphers (TBCs)*

(Standard) blockcipher:

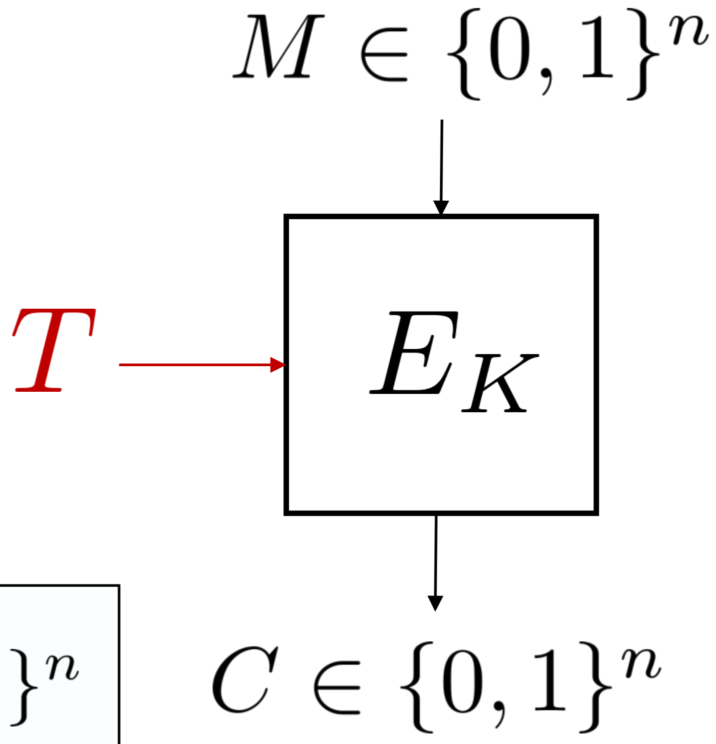


E_K: Permutation on $\{0,1\}^n$

Restricted class of AE schemes

Consider AE schemes using *tweakable blockciphers (TBCs)*

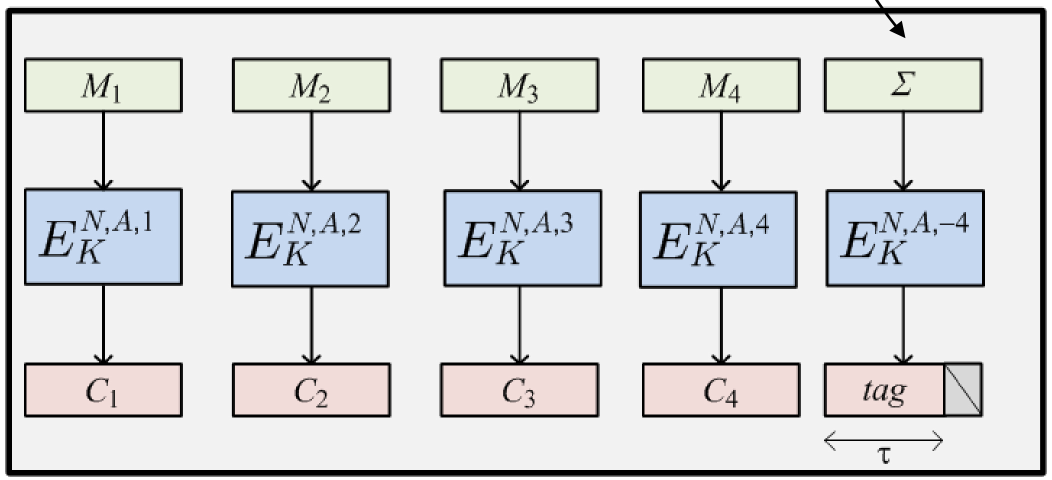
Tweakable blockcipher:

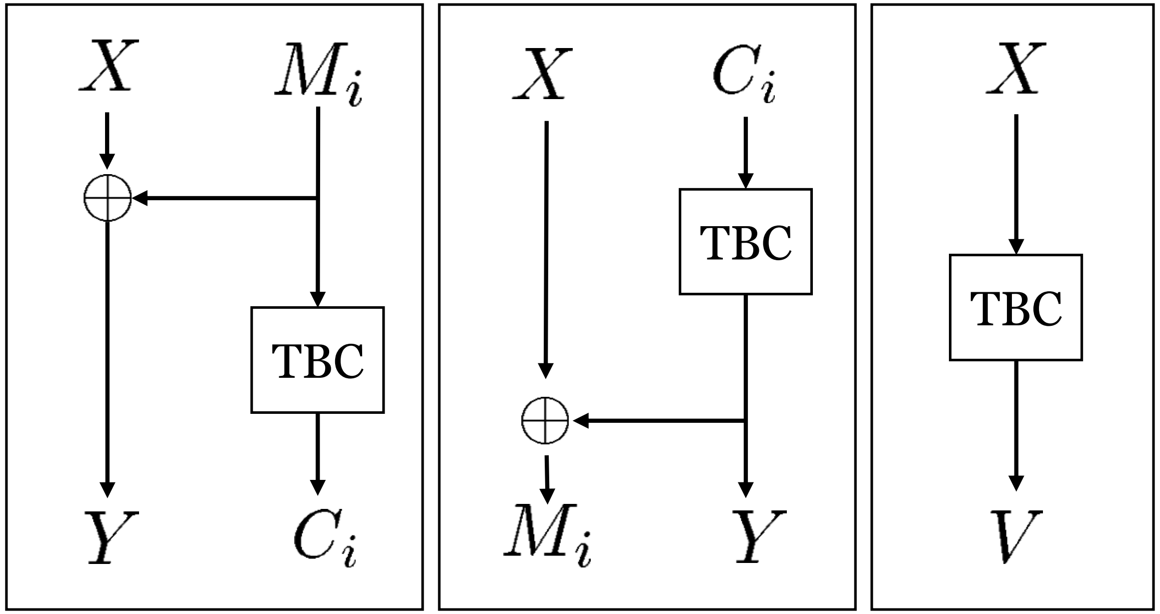


E_K^T: Permutation on $\{0,1\}^n$

Example AE scheme using TBCs: OCB

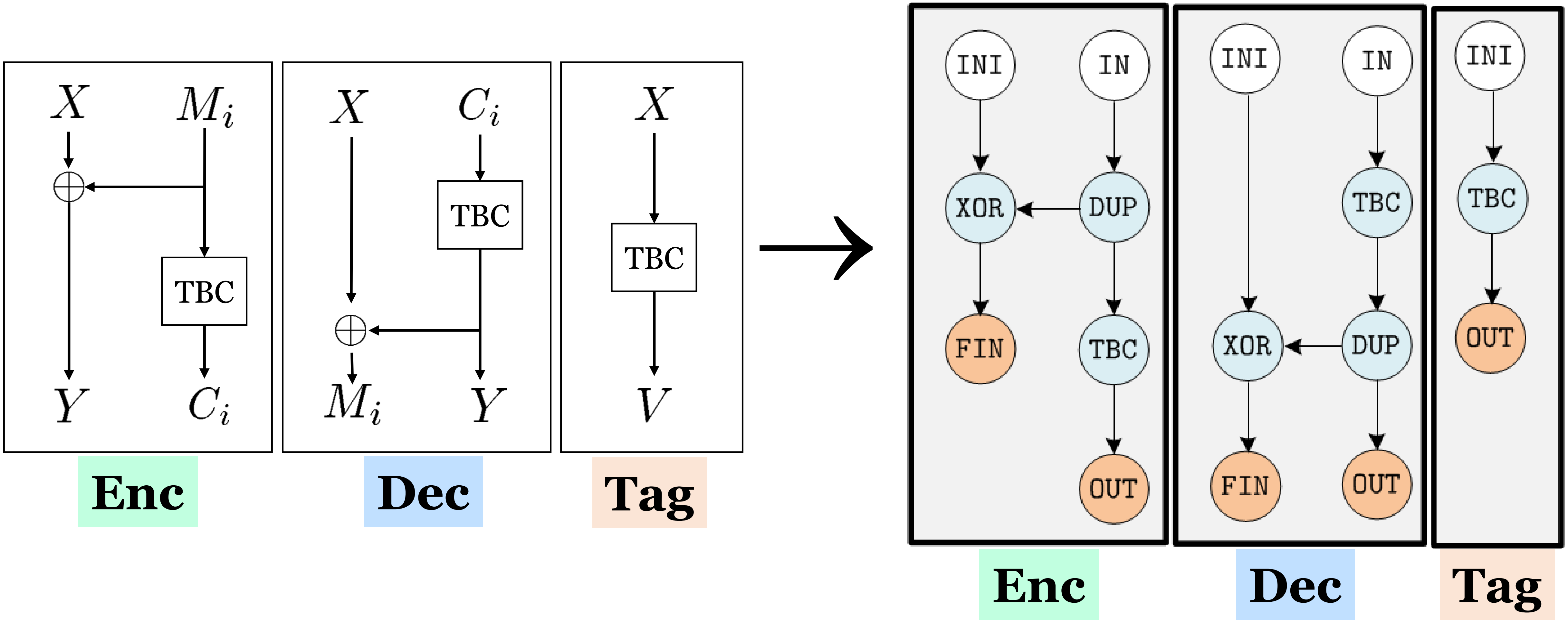
$$\Sigma = M_1 \oplus M_2 \oplus M_3 \oplus M_4$$





Enc **Dec** **Tag**

Viewing AE schemes as graphs



Type system for graphs

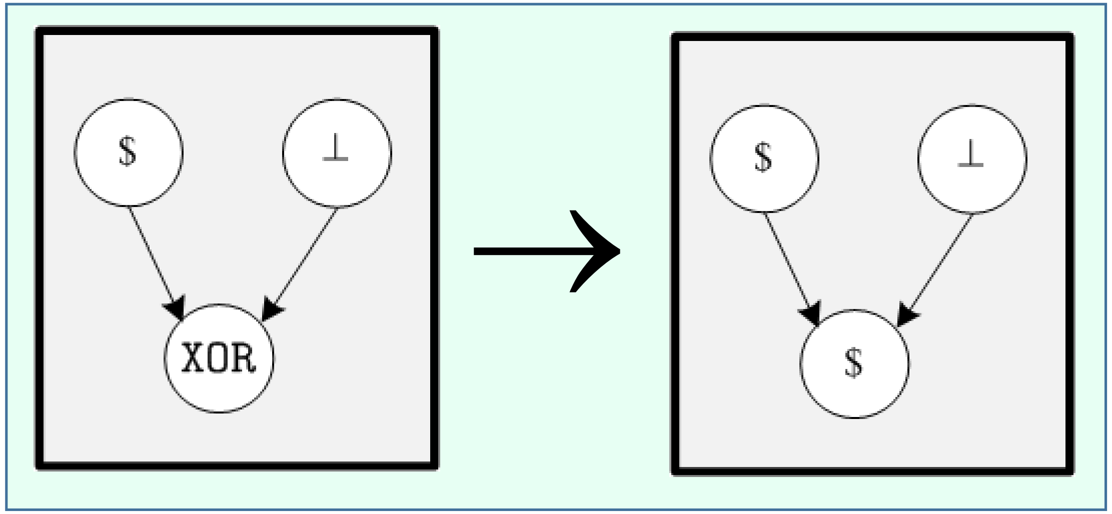
Each **node** assigned ***type*** corresponding to "property" of node output

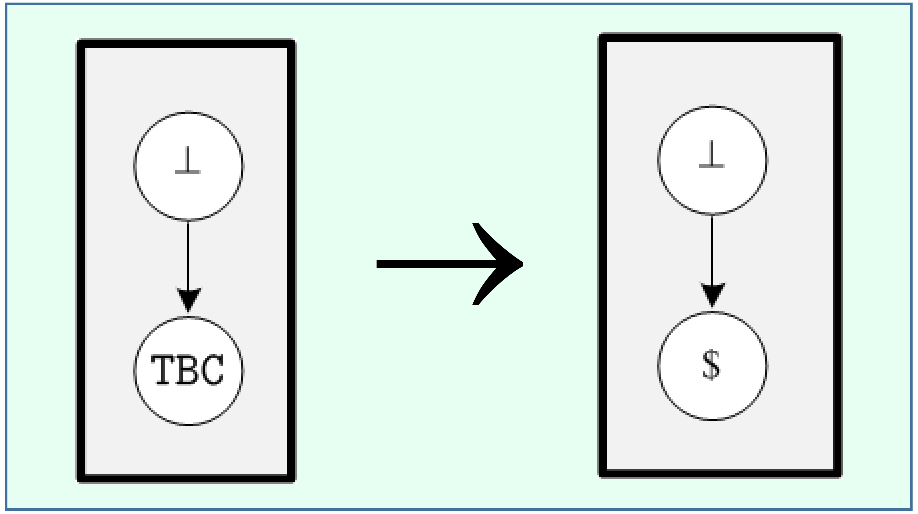
Type = $\{\perp, \$, 0, 1\}$

- $\perp$: "Arbitrary"
- $\$$: "Random"
- 0 and 1: Used in authenticity check (ignore for this talk)

Type system for graphs

Constraints on how nodes can be typed, e.g.:



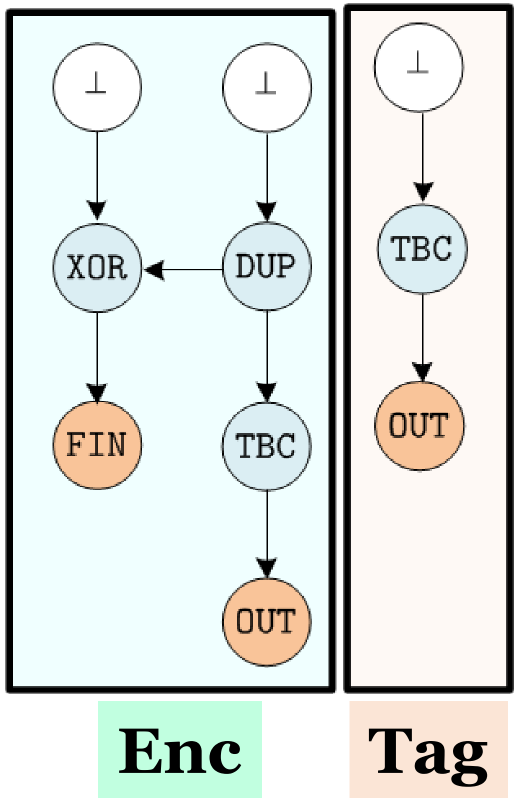


Can show:
Node type \$ $\Rightarrow$ node output is (pseudo)random

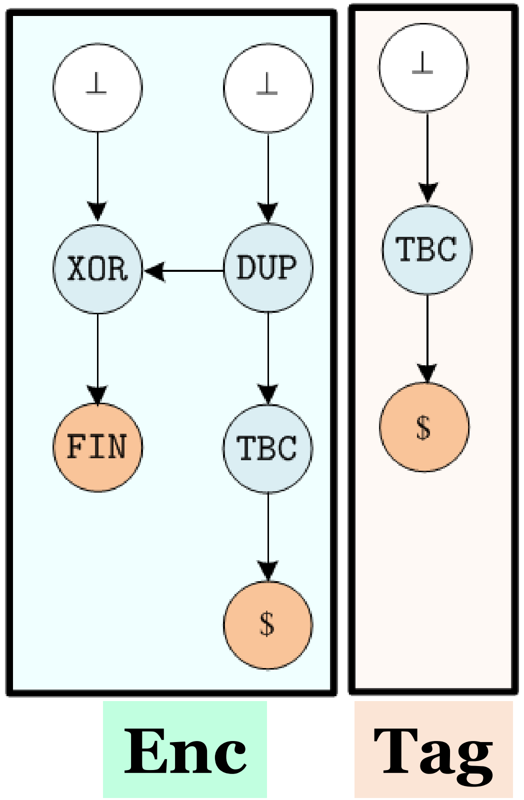
From typed graphs to secure AE schemes

Need to show *two* properties: **privacy** and **authenticity**

Privacy: $\Pr[A^{\mathcal{E}_K(\cdot)} \Rightarrow 1] - \Pr[A^{\$(\cdot)} \Rightarrow 1]$ is small



Goal
$\rightarrow$



From typed graphs to secure AE schemes

Authenticity: $\Pr[A^{\mathcal{E}_K(\cdot)} \text{ forges}]$ is small

valid $C \ (= C_1 \cdots C_\ell V)$

A

C must not be output of some prior query to $\mathcal{E}_K(\cdot)$

Decrypting $C_1 \cdots C_\ell$ produces random tag V'
$\Rightarrow V' \neq V \Rightarrow$ Not valid forgery

Implementation

Implemented *analyzer* and *synthesizer* in OCaml

Analyzer:

(**Enc**, **Tag**) → Derive **Dec** from **Enc** → (**Enc**, **Dec**, **Tag**)

↓

Priv + auth checks

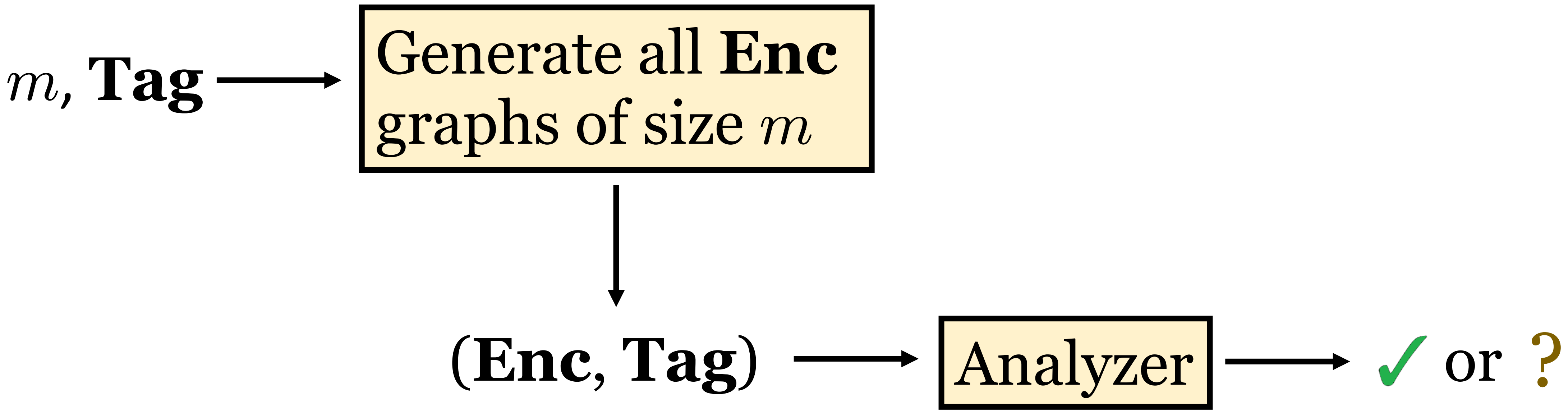
↓

✓ or ?

Implementation

Implemented *analyzer* and *synthesizer* in OCaml

Synthesizer:



Synthesis results

Ran synthesizer for **Enc** graphs of size 12-16…

Size	# Secure	# Optimal	# Parallel	Time
12	13	13	5	47 sec
13	142	0	0	4.3 min
14	582	171	5	24.2 min
15	2826	40	6	2.8 hours
16	3090	66	1	3 hours*
Total	**6653**	**290**	**17**	

- **Optimal**: 1 TBC per message block
- **Parallel**: TBC calls can be parallelized

* = Stopped analysis after given time

Synthesis results

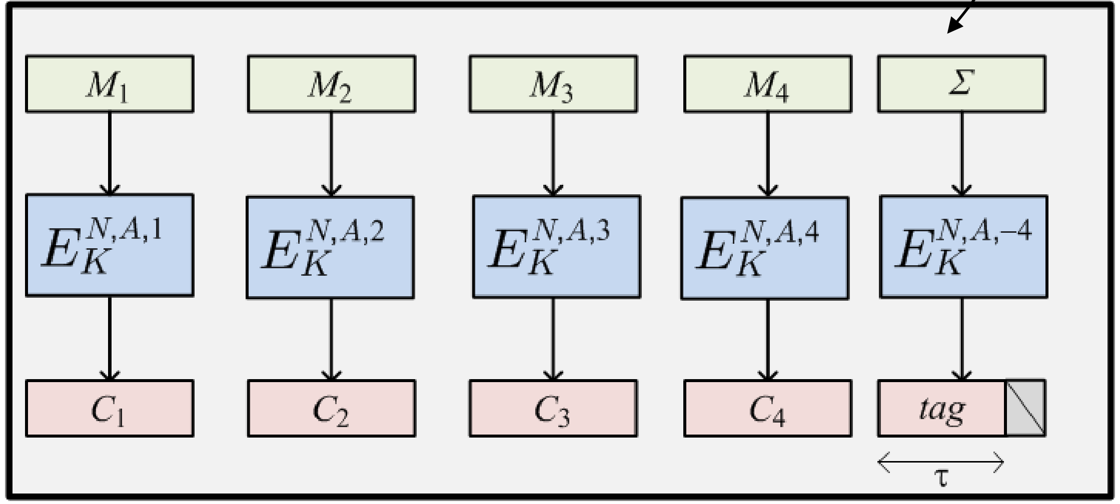
Ran synthesizer for **Enc** graphs of size 12-16...

Size	# Secure	# Optimal	# Parallel	Time
12	13	13	5	47 sec
13	142	0	0	4.3 min
14	582	171	5	24.2 min
15	2826	40	6	2.8 hours
16	3090	66	1	3 hours*
Total	**6653**	**290**	**17**	

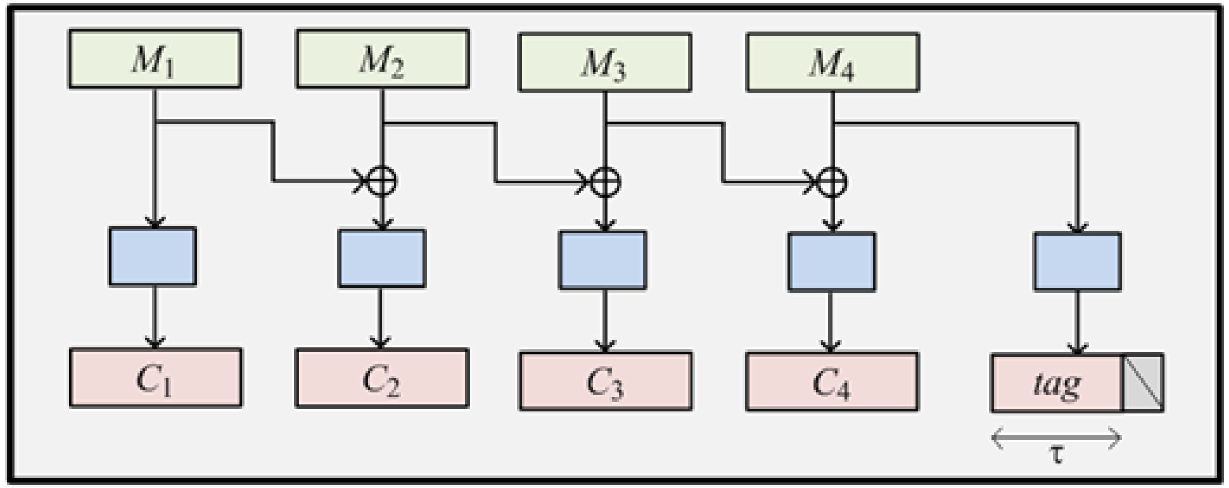
OCB is the only previously known AE scheme of size 12

Example synthesized schemes

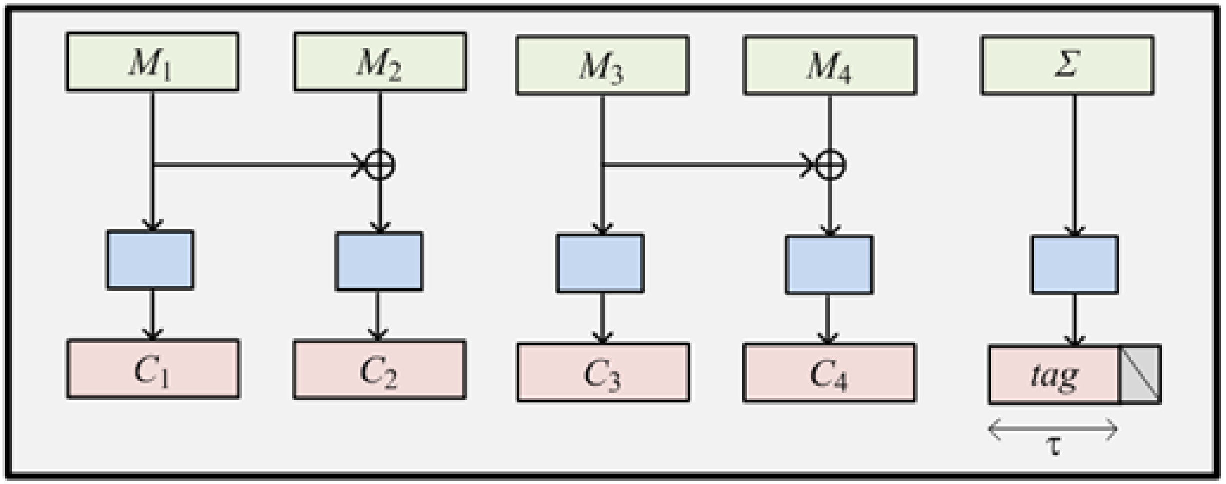
$$\Sigma = M_1 \oplus M_2 \oplus M_3 \oplus M_4$$



OCB



Scheme 1



Scheme 2

Example synthesized schemes: performance

Scheme	Encryption (cpb)	Decryption (cpb)
OCB	0.71	0.76
1	0.72	0.75
2	0.71	0.76

Synthesized novel schemes on par with OCB

Conclusion

- Developed system for automatically *analyzing* and *synthesizing* AE schemes
- Able to synthesize schemes *as efficient and parallelizable as OCB*
- More results (e.g., automated attack generation) in paper

Full Version: https://eprint.iacr.org/2015/624
Code: https://www.github.com/amaloz/ae-generator