CLOC: Authenticated Encryption for Short Input

Tetsu Iwata, Nagoya University

Kazuhiko Minematsu, NEC Corporation

Jian Guo, Nanyang Technological University

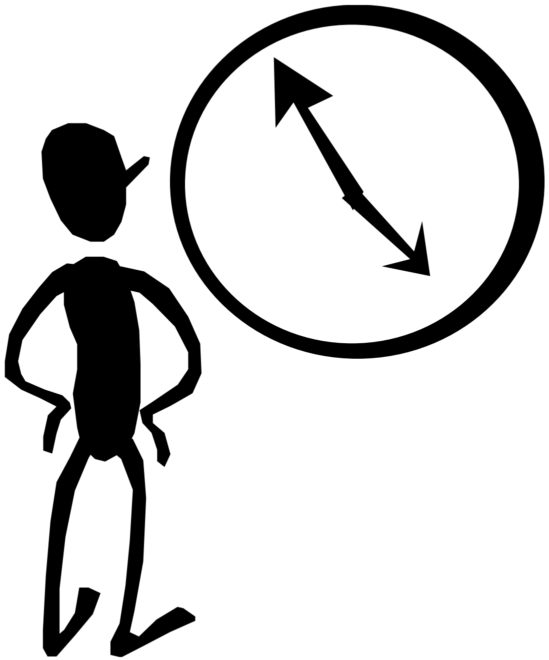
Sumio Morioka, NEC Europe Ltd.

FSE 2014

March 3, 2014, London, UK

Outline

- A new authenticated encryption with associate data scheme (AEAD)
- CLOC: Compact Low-Overhead CFB, pronounced as “clock”



CLOC Design Goal

- Provably secure AEAD that is based on a blockcipher
  - Standard security notions for privacy and authenticity
- To improve previous schemes, CCM, EAX, and EAX-prime
  - the implementation overhead beyond the blockcipher
  - the precomputation complexity
  - the memory requirement

CLOC Design Goal

- Suitable for handling short input data, say 16 bytes, without needing precomputation nor large memory
- Suitable for small microprocessors, where the word size is typically 8 bits or 16 bits, and there are significant restrictions in the size and the number of registers

CCM, EAX, and EAX-Prime

- AEADs based on a blockcipher
- CCM (NIST SP 800-38C)
  - not online
- EAX (ISO/IEC 19772)
  - precomputation costs ($L = E_K(0)$, 2L, 4L, $E_K(1)$, and $E_K(2)$)
  - time and memory
- EAX-prime (ANSI C12.22)
  - efficiently handles short input data with small memory
  - practical attacks
- CLOC removes these limitations
  - remove $L = E_K(0)$ or doubling operations over $GF(2^n)$

Short Input Data

- Performance for short input data matters:
  - Low-power sensor networks
    - Zigbee: at most 127 bytes
  - Bluetooth Low Energy: at most 47 bytes
  - Electronic Product Code (EPC): typically 96 bits
- For long input data, the efficiency of CLOC is the same as CCM, EAX, and EAX-prime
  - 2 blockcipher calls per 1 plaintext block
  - CLOC is for short input data

CLOC Properties

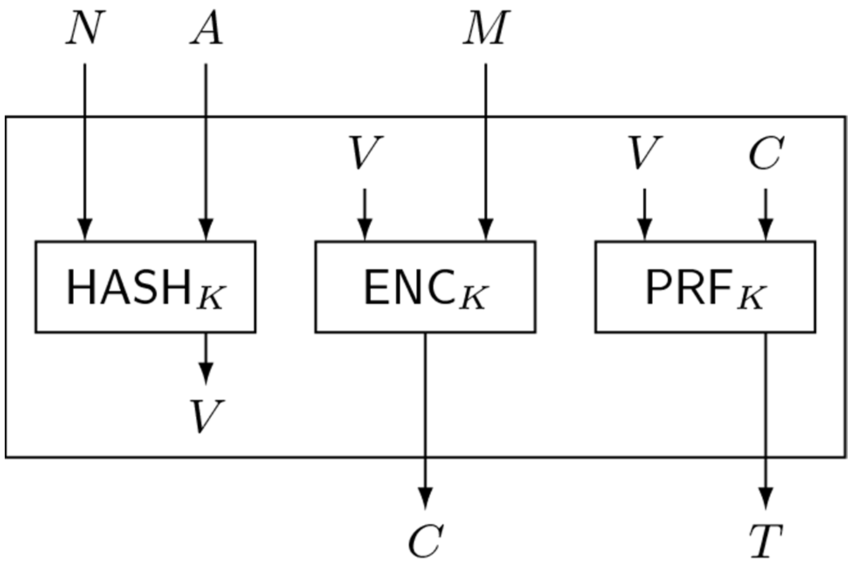
- Nonce-based AEAD
- uses only the encryption of the blockcipher both for encryption and decryption
- When $|A| \geq 1$, it makes $|N|_n + |A|_n + 2|M|_n$ blockcipher calls for a nonce N, associated data A, and a plaintext M
  - where $|X|$ is the length of X in bits and $|X|_n$ is the length in n-bit blocks
  - $1 \leq |N| \leq n-1$, so $|N|_n = 1$
  - No precomputation (blockcipher calls, generation of key dependent tables, . . .) is needed
  - when $|A| = 0$, it needs $|N|_n + 1 + 2|M|_n$ calls

CLOC Properties

- For short input data
  - 1-block nonce, 1-block associated data, and 1-block plaintext
  - CLOC: 4 calls
  - CCM: 5 or 6 calls
  - EAX: 7 calls (where 3 out of 7 can be precomputed)
  - EAX-prime: 5 calls (where 1 out of 5 can be precomputed)
- Static associated data can be handled efficiently
- It works with two state blocks (i.e. 2n bits)
- Sequential

Overview of the Scheme

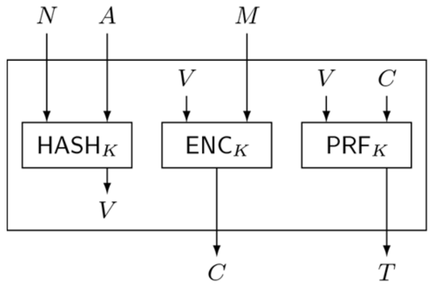
- Encrypt-then-PRF paradigm
- uses a variant of CFB mode in its encryption part and a variant of CBC MAC in the authentication part



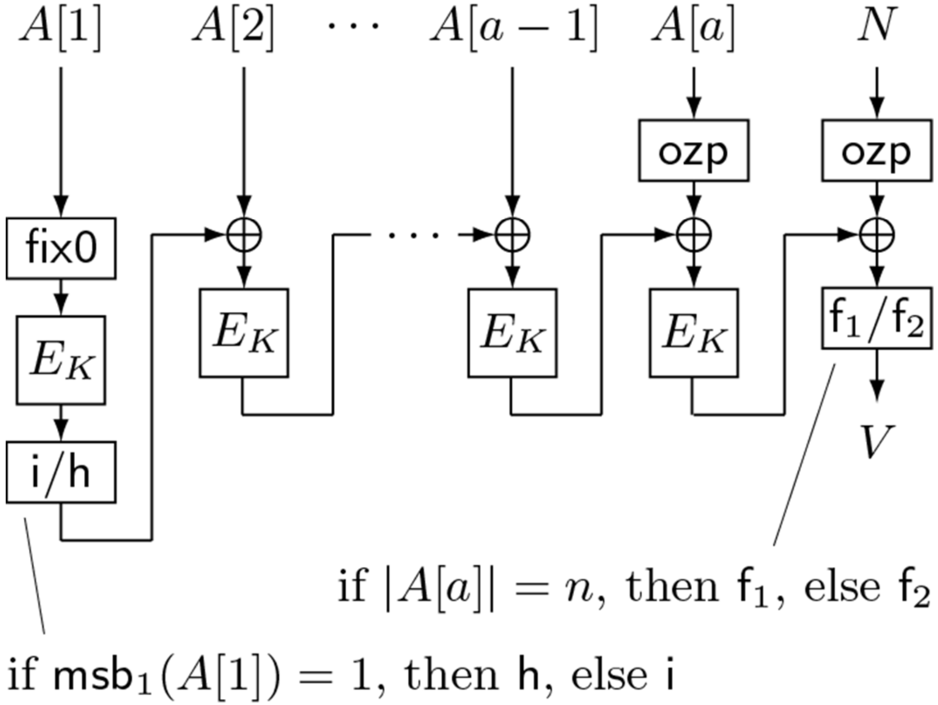
Tools

- The one-zero padding function: ozp
  - ozp(X) = X if |X|=jn for some j > 0, and ozp(X) = X||10…0
- The tweak functions: f_1, f_2, g_1, g_2, and h
  - use them to directly update the state
- The bit fixing functions: fix0 and fix1
  - fix0(X): overwrite msb_1(X) with 0
  - fix1(X): overwrite msb_1(X) with 1
    - fix1(0000) = 1000, fix1(1100) = 1100

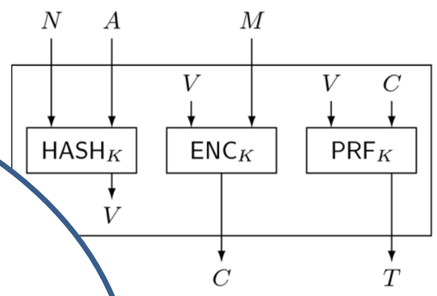
V <- $HASH_K$(A,N)



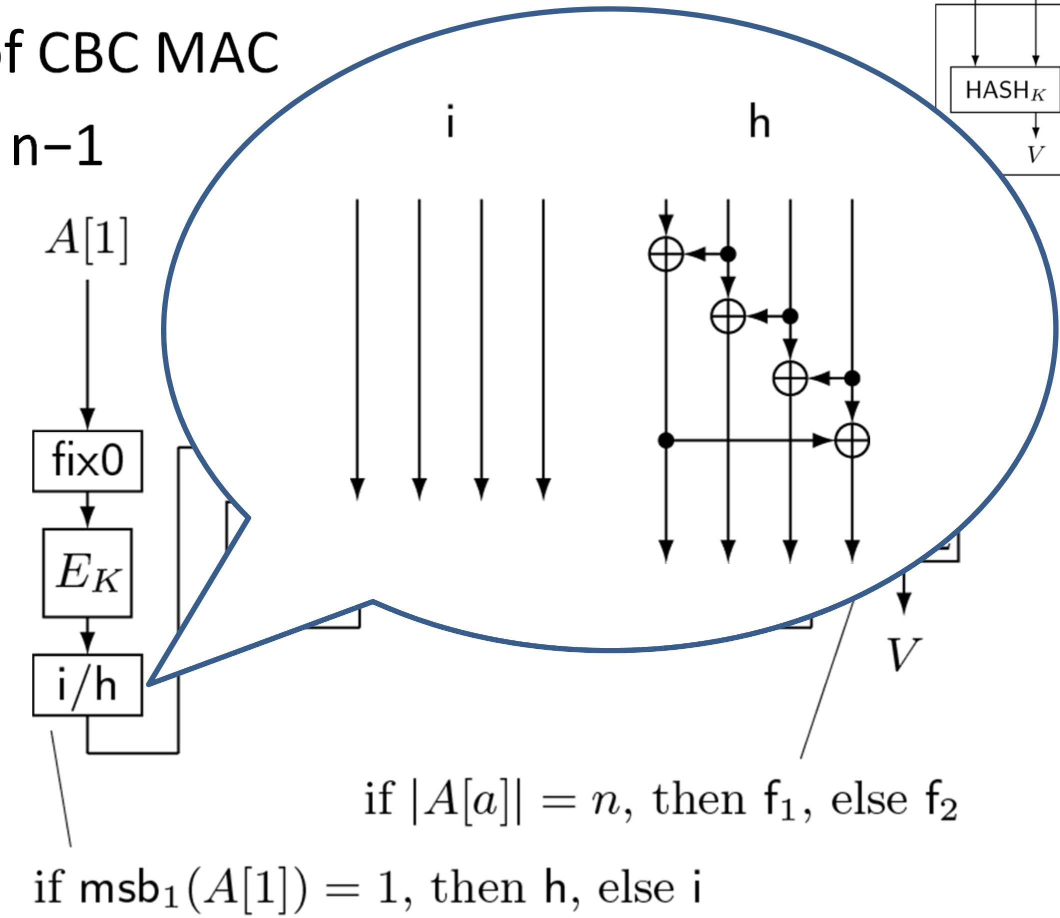
- A variant of CBC MAC
- $1 \leq |N| \leq n-1$



V <- $\text{HASH}_K(A,N)$

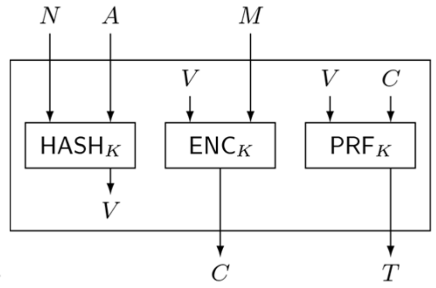


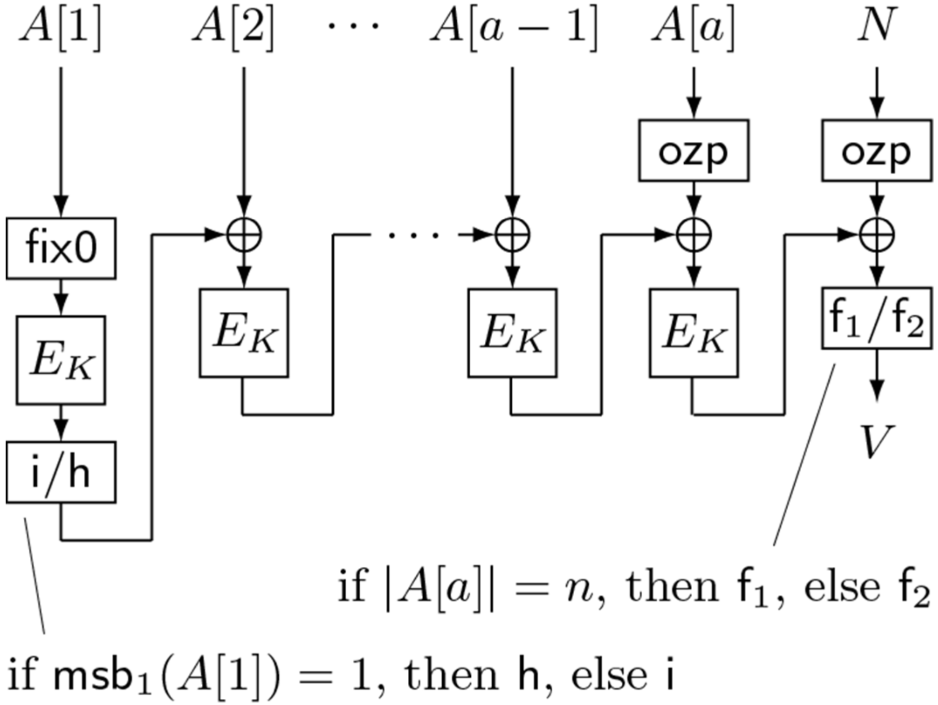
- A variant of CBC MAC
- $1 \leq |N| \leq n-1$



V <- $HASH_K(A,N)$

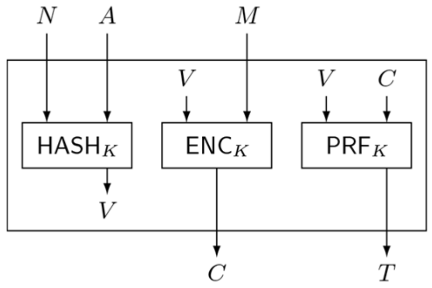
- A variant of CBC MAC
- $1 \leq |N| \leq n-1$

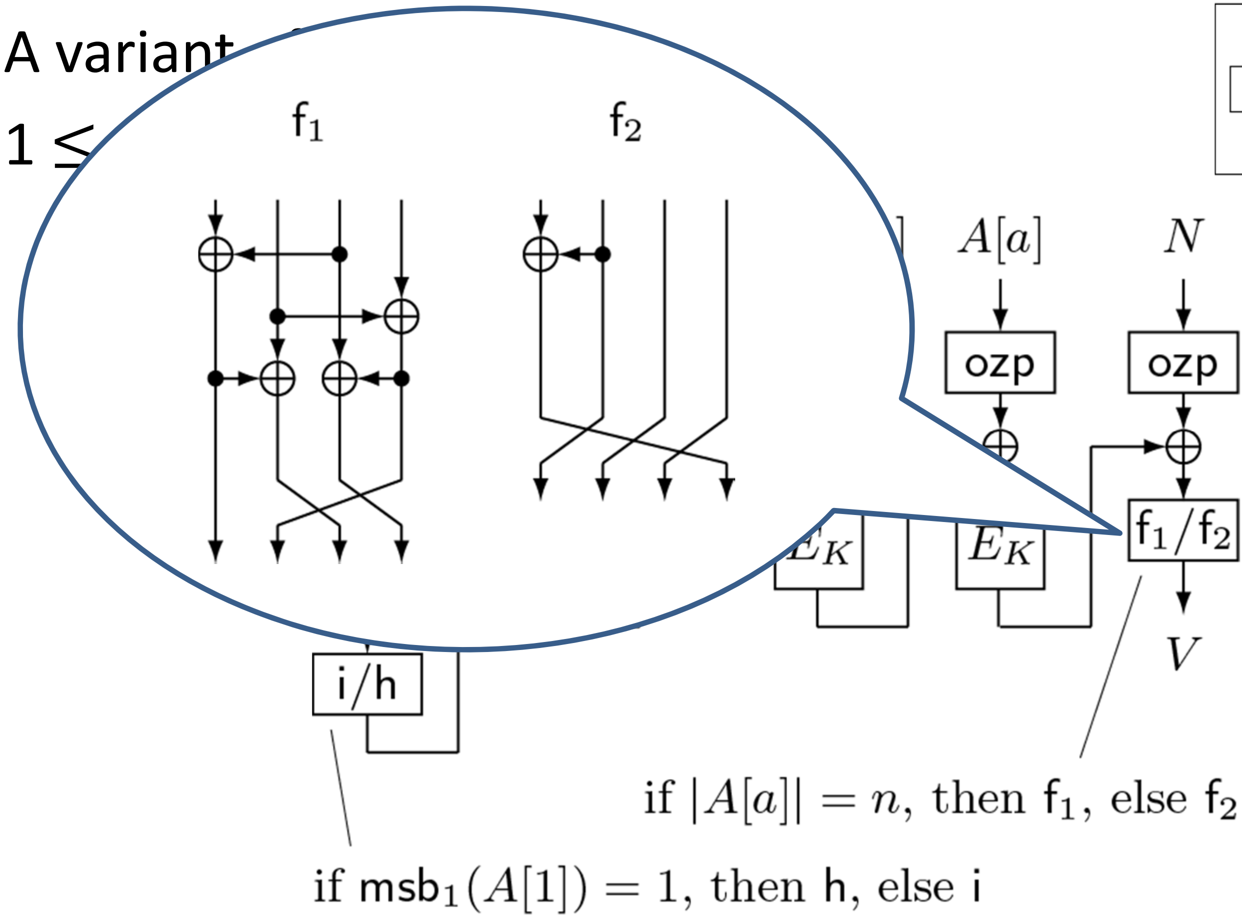




V <- HASH$_K$(A,N)

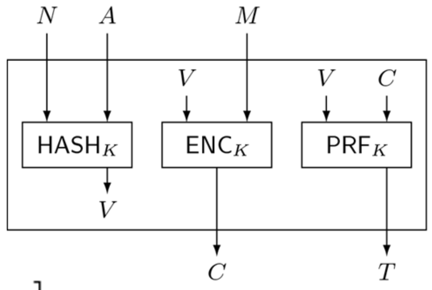
- A variant
- 1 ≤

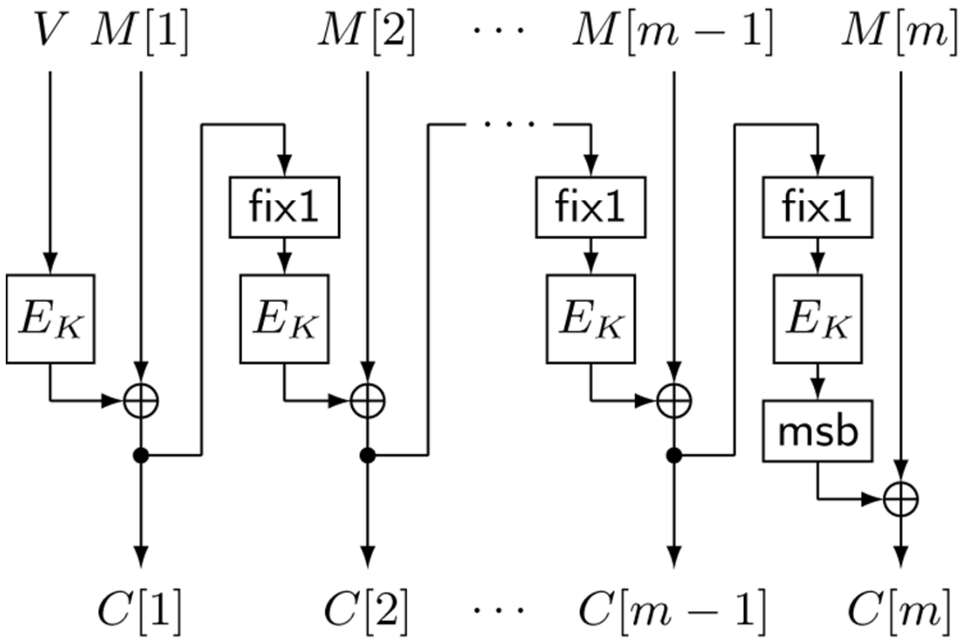




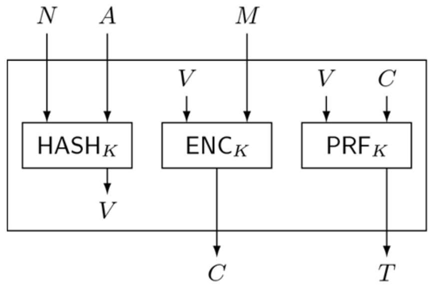
C <- ENC$_K$(V,M)

- A variant of CFB mode

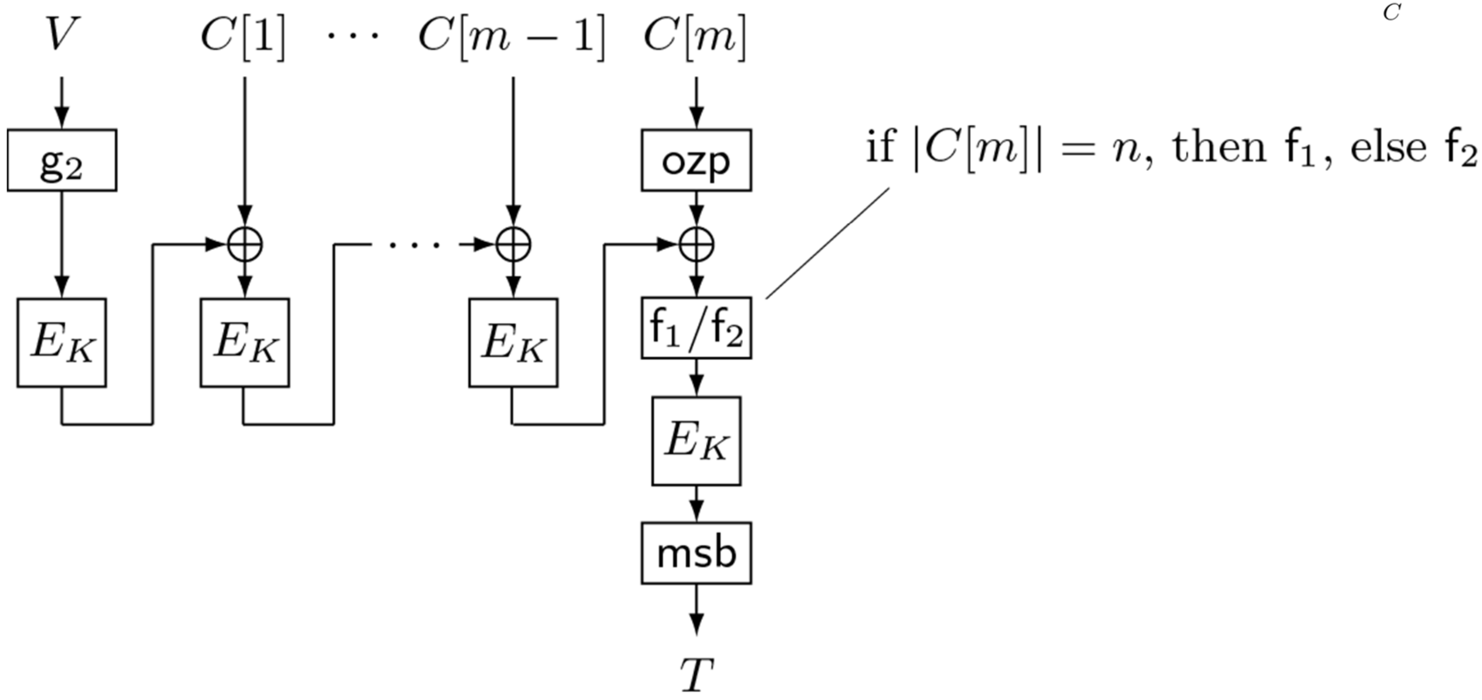




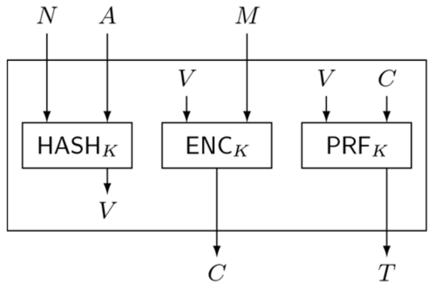
T <- $PRF_K(V,C)$



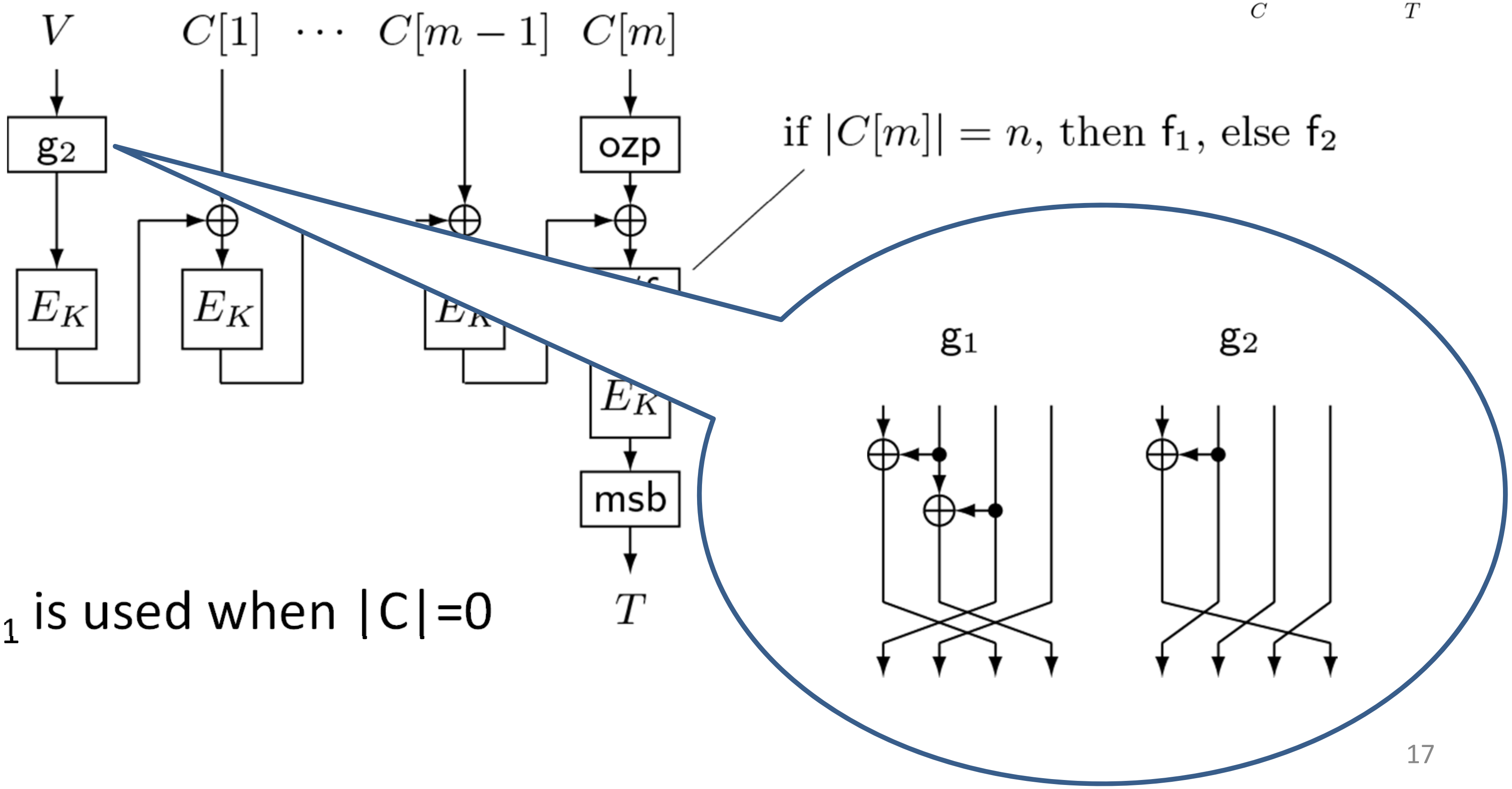
- A variant of CBC MAC



T <- $\text{PRF}_K(V,C)$



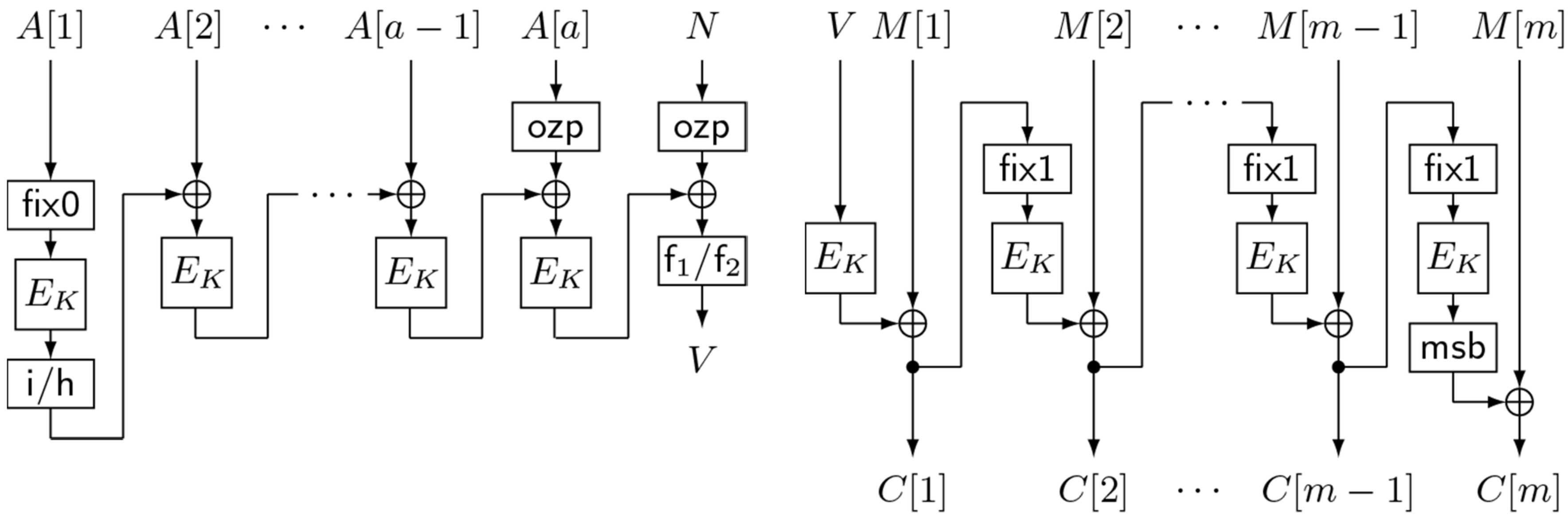
- A variant of CBC MAC



- g_1 is used when $|C|=0$

Rationale

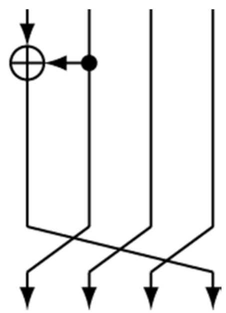
- The bit fixing functions
  - used to logically separate CBC MAC and CFB mode
  - otherwise, attacks are possible



Rationale

- The tweak functions
  - There are 55 differential probability constraints
    - K xor $f_1(K)$, $f_1(K)$ xor $g_1(f_1(h(K)))$, . . .
  - Define a matrix M as

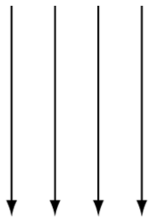
$$\mathbf{M} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$



  - K · M = (K[1], K[2], K[3], K[4]) · M
    = (K[2], K[3], K[4], K[1] xor K[2])

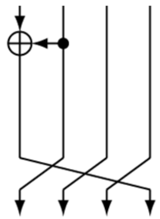
$\mathbf{M}^0$

$$\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{pmatrix}$$



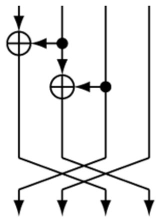
$\mathbf{M}^1$

$$\begin{pmatrix}0&0&0&1\\1&0&0&1\\0&1&0&0\\0&0&1&0\end{pmatrix}$$



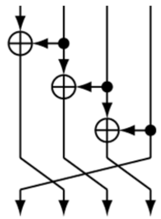
$\mathbf{M}^2$

$$\begin{pmatrix}0&0&1&0\\0&0&1&1\\1&0&0&1\\0&1&0&0\end{pmatrix}$$



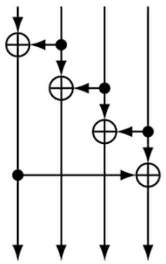
$\mathbf{M}^3$

$$\begin{pmatrix}0&1&0&0\\0&1&1&0\\0&0&1&1\\1&0&0&1\end{pmatrix}$$



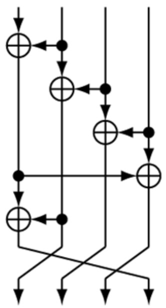
$\mathbf{M}^4$

$$\begin{pmatrix}1&0&0&1\\1&1&0&1\\0&1&1&0\\0&0&1&1\end{pmatrix}$$



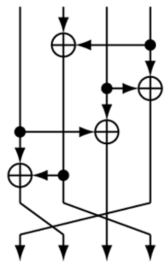
$\mathbf{M}^5$

$$\begin{pmatrix}0&0&1&1\\1&0&1&0\\1&1&0&1\\0&1&1&0\end{pmatrix}$$



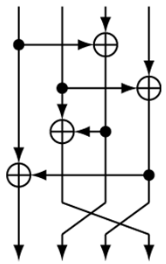
$\mathbf{M}^6$

$$\begin{pmatrix}0&1&1&0\\0&1&0&1\\1&0&1&0\\1&1&0&1\end{pmatrix}$$



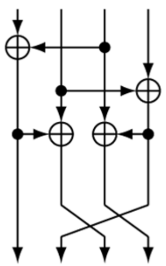
$\mathbf{M}^7$

$$\begin{pmatrix}1&1&0&1\\1&0&1&1\\0&1&0&1\\1&0&1&0\end{pmatrix}$$



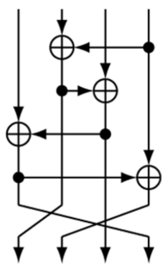
$\mathbf{M}^8$

$$\begin{pmatrix}1&0&1&0\\0&1&1&1\\1&0&1&1\\0&1&0&1\end{pmatrix}$$



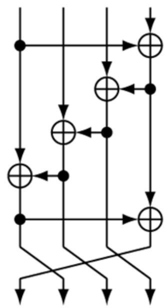
$\mathbf{M}^9$

$$\begin{pmatrix}0&1&0&1\\1&1&1&1\\0&1&1&1\\1&0&1&1\end{pmatrix}$$



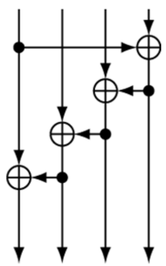
$\mathbf{M}^{10}$

$$\begin{pmatrix}1&0&1&1\\1&1&1&0\\1&1&1&1\\0&1&1&1\end{pmatrix}$$



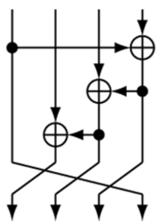
$\mathbf{M}^{11}$

$$\begin{pmatrix}0&1&1&1\\1&1&0&0\\1&1&1&0\\1&1&1&1\end{pmatrix}$$



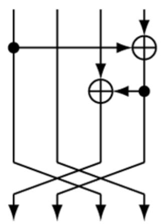
$\mathbf{M}^{12}$

$$\begin{pmatrix}1&1&1&1\\1&0&0&0\\1&1&0&0\\1&1&1&0\end{pmatrix}$$



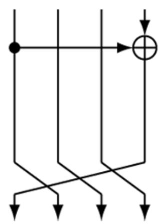
$\mathbf{M}^{13}$

$$\begin{pmatrix}1&1&1&0\\0&0&0&1\\1&0&0&0\\1&1&0&0\end{pmatrix}$$



$\mathbf{M}^{14}$

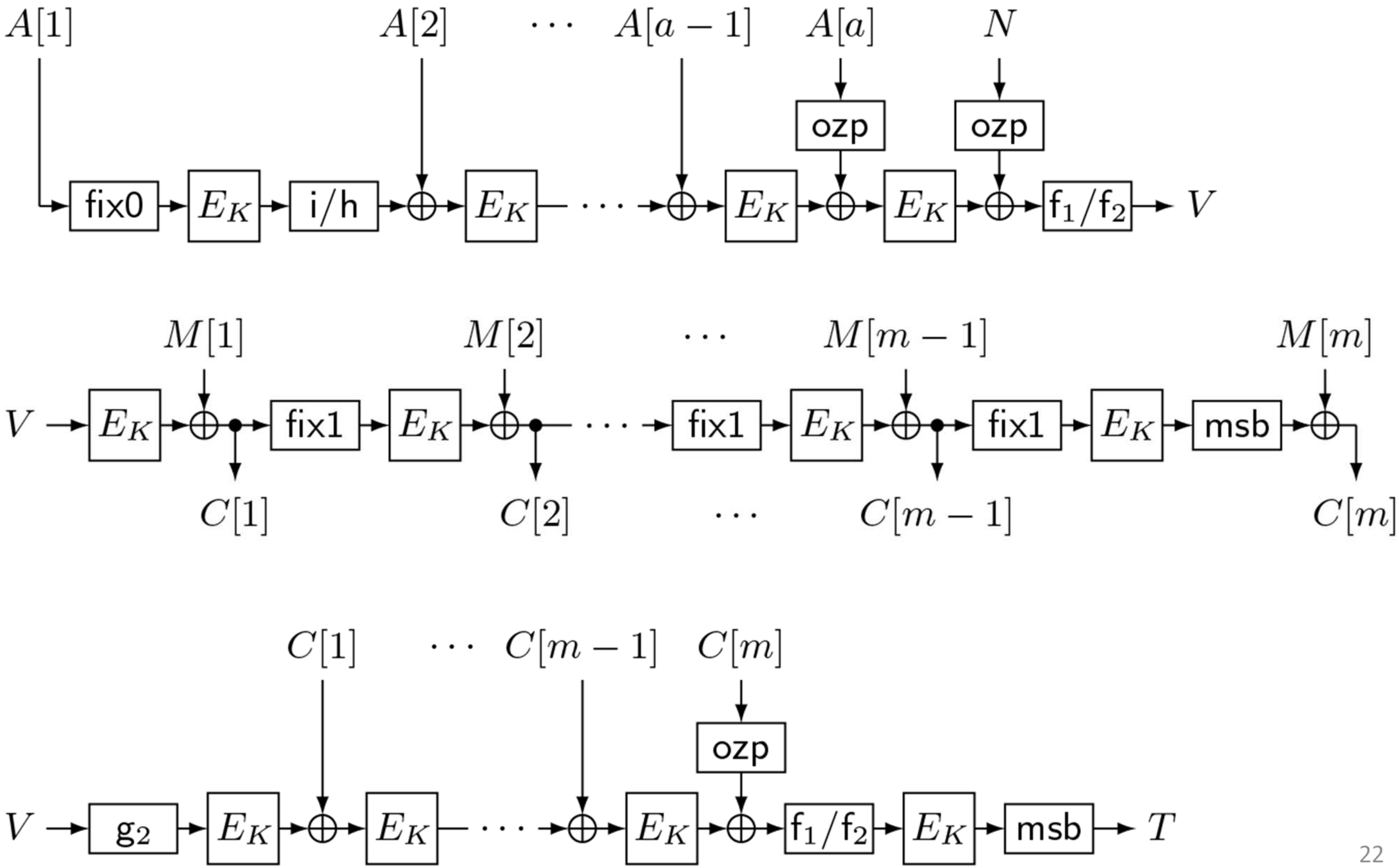
$$\begin{pmatrix}1&1&0&0\\0&0&1&0\\0&0&0&1\\1&0&0&0\end{pmatrix}$$



Rationale

- The tweak functions
  - associate $(i_1, i_2, i_3, i_4, i_5) \in \{1, . . . , 14\}^5$ with (f_1, f_2, g_1, g_2, h)
  - f_1: M^{i1}, f_2: M^{i2}, g_1: M^{i3}, g_2: M^{i4}, h: M^{i5}
- Tested all $(i_1, i_2, i_3, i_4, i_5) \in \{1, . . . , 14\}^5$
  - e.g., K xor $f_1(K)$: the rank of I xor M^{i1} is full (I is the identity matrix)
  - 14^5 -> 864 candidates
- Defined a cost function to choose the best exponentiations
  - roughly measures the computational cost of (f_1, f_2, g_1, g_2, h)
  - $(i_1, i_2, i_3, i_4, i_5) = (8, 1, 2, 1, 4)$

Works with Two State Blocks



Security

- Privacy:
  - Indistinguishability of ciphertexts from random bits against nonce-respecting adversaries in a chosen plaintext attack setting

- $\mathbf{Adv}^{\mathrm{priv}}_{\mathrm{CLOC}[E,\ell_N,\tau]}(\mathcal{A}) \stackrel{\mathrm{def}}{=} \Pr\left[\mathcal{A}^{\mathrm{CLOC}\text{-}\mathcal{E}_K(\cdot,\cdot,\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\$(\cdot,\cdot,\cdot)} \Rightarrow 1\right]$

- $\mathbf{Adv}^{\mathrm{priv}}_{\mathrm{CLOC}[\mathrm{Perm}(n),\ell_N,\tau]}(\mathcal{A}) \leq \dfrac{5\sigma_{\mathrm{priv}}^2}{2^n}$, where $\sigma_{\mathrm{priv}} = q + \sigma_A + 2\sigma_M$

Security

- Authenticity:
  - Unforgeability against nonce-reusing adversaries in a chosen ciphertext attack setting
  - A strong adversary

- $\mathbf{Adv}^{\text{auth}}_{\text{CLOC}[E,\ell_N,\tau]}(\mathcal{A}) \stackrel{\text{def}}{=} \Pr\left[\mathcal{A}^{\text{CLOC-}\mathcal{E}_K(\cdot,\cdot,\cdot),\text{CLOC-}\mathcal{D}_K(\cdot,\cdot,\cdot,\cdot)} \text{ forges}\right]$

- $\mathbf{Adv}^{\text{auth}}_{\text{CLOC}[\text{Perm}(n),\ell_N,\tau]}(\mathcal{A}) \leq \frac{5\sigma_{\text{auth}}^2}{2^n} + \frac{q'}{2^\tau}$

  where $\sigma_{\text{auth}} = q + \sigma_A + 2\sigma_M + q' + \sigma_{A'} + \sigma_{C'}$

Software Implementation

- Embedded software
- Atmel AVR ATmega128
  - 8-bit microprocessor
  - AES from [AVR-Crypto-Lib] written in assembler
    - 156.7 cpb for encryption, 196.8 cpb for decryption
  - CLOC, EAX, and OCB3
    - modes are written in C
    - OCB3 code from [OCB News and Code] w/ modification
      - doubling operations are on-line, large precomputation may not be suitable to handle short input data for microprocessors
  - compiled with Atmel Studio 6

Software Implementation

	ROM (bytes)	RAM (bytes)	Init (cycles)	Speed (cycles/byte)					
				Data 16	32	64	96	128	256
CLOC	2980	362	1999	750.1	549.0	448.4	414.9	398.2	373.0
EAX	2772	402	12996	913.6	632.5	490.8	443.6	419.9	384.5
OCB-E	5010	971	4956	1217.5	736.1	495.5	412.2	375.1	314.9
OCB-D	5010	971	4956	1252.2	773.4	534.0	451.2	414.3	354.4

- 1-block AD, no static AD computation
- cycle counting is obtained by the simulation of Atmel Studio 6
- RAM is measured with a public tool [EZSTACK]
- In CLOC, the RAM usage is low and Init is fast, and it is fast for short input data, up to around 128 bytes

Software Implementation

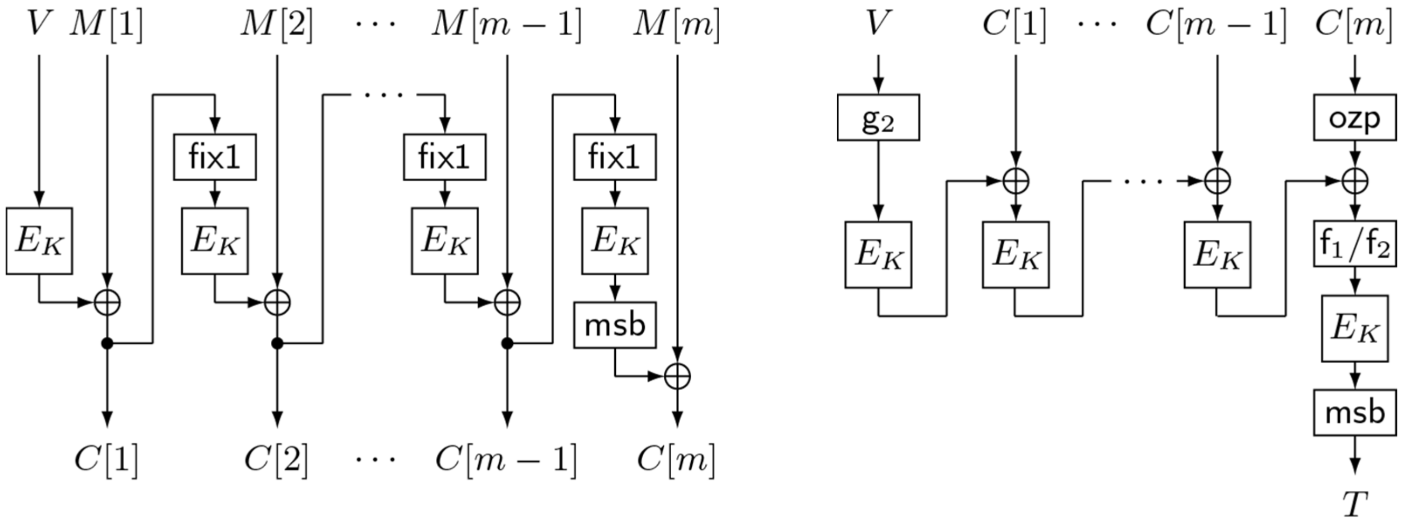
	ROM (bytes)	RAM (bytes)	Init (cycles)	Speed (cycles/byte)					
				Data 16	32	64	96	128	256
CLOC	2980	362	1999	750.1	549.0	448.4	414.9	398.2	373.0
EAX	2772	402	12996	913.6	632.5	490.8	443.6	419.9	384.5
OCB-E	5010	971	4956	1217.5	736.1	495.5	412.2	375.1	314.9
OCB-D	5010	971	4956	1252.2	773.4	534.0	451.2	414.3	354.4

updated from the pre-proceedings

Software Implementation

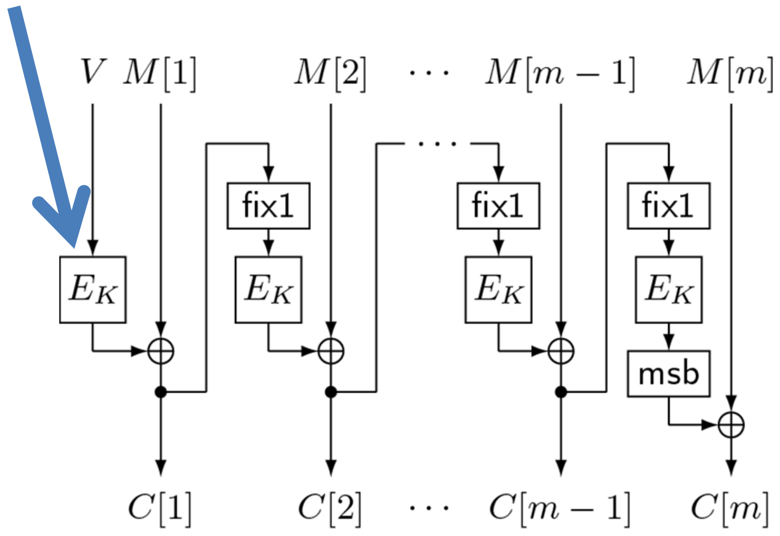
- General purpose CPU
- Intel processor, Core i5-3427U 1.80GHz (Ivy Bridge family)
- AES-128, AES-NI
- CLOC: about 4.9 cpb for long input data (more than 2^{20} blocks)
- AES calls in CFB mode and CBC MAC (in tag generation) can be done in parallel

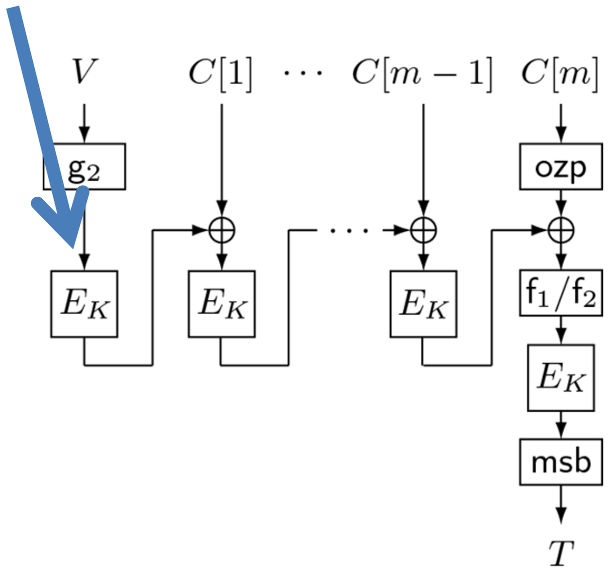
Software Implementation



- For long input data, CLOC is close to the speed of serial encryption only mode (CBC mode)
- CLOC: about 4.9 cpb
  - serial AES-128 encryption: about 4.3 cpb

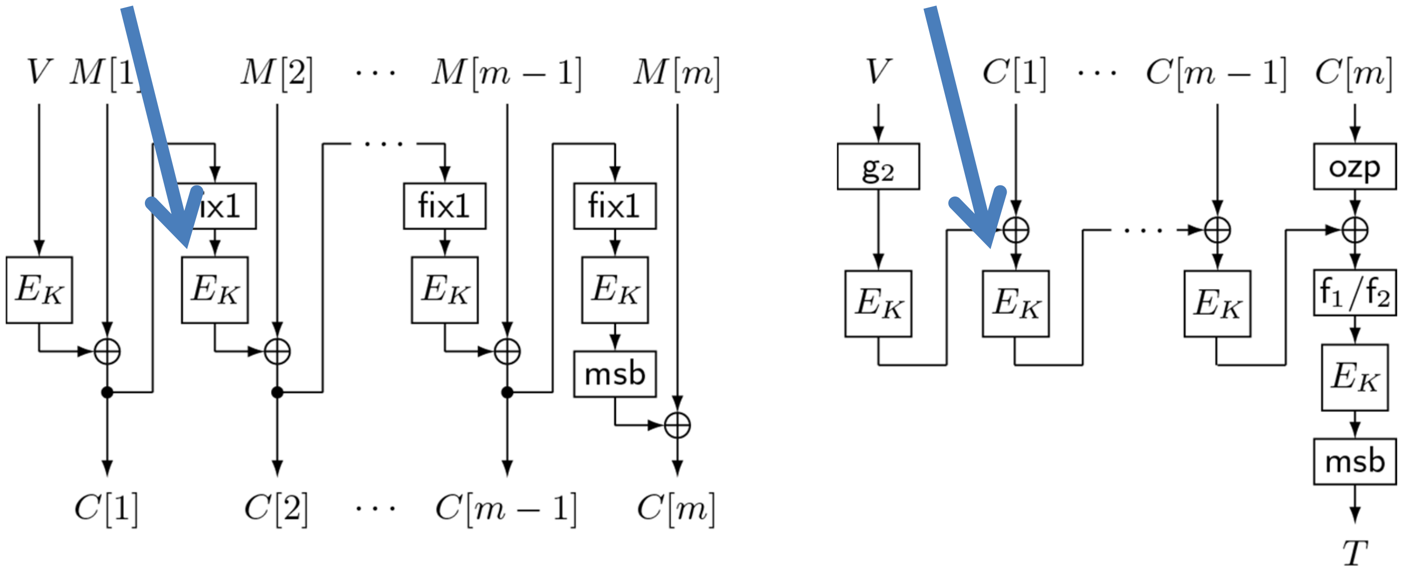
Software Implementation





- For long input data, CLOC is close to the speed of serial encryption only mode (CBC mode)
- CLOC: about 4.9 cpb
  - serial AES-128 encryption: about 4.3 cpb

Software Implementation



- For long input data, CLOC is close to the speed of serial encryption only mode (CBC mode)
- CLOC: about 4.9 cpb
  - serial AES-128 encryption: about 4.3 cpb

Hardware Implementation

- Not the main focus
- Altera FPGA, Cyclone IV GX (EP4CGX110DF31C7)
  - w/ AES-128, composite field S-box implementation, round-based architecture
- Size is measured in terms of LEs (logic elements)
- one block of associated data and 8 blocks of plaintexts

	Size (LE)	Max. Freq. (MHz)	Throughput (Mbit/sec)
CLOC	5628	82.1	400.7
EAX	6453	61.3	342.2
AES Enc	3175	98.7	971.7

- Slightly smaller and faster than EAX

Conclusions

- Designed CLOC and analyzed the security and the efficiency
- CLOC is designed to efficiently handle short input data and suitable for use in small microprocessors
  - it works without heavy precompuation nor large memory