Generating a Fixed Number of Masks with Word Permutations and XORs

Tetsu Iwata, Nagoya University

Kazuhiko Minematsu, NEC Corporation

DIAC 2013, Directions in Authenticated Ciphers

August 12, 2013, Chicago, USA

Overview

- Masks are frequently used in designs of blockcipher-based MACs and AEADs
- Some of them use many masks (the number depends on the input length)
  - Examples: PMAC (MAC), OCB (AEAD)
- Others use a fixed number of masks
  - Examples: CMAC (MAC), EAX (AEAD)
- In many cases, multiplications over $GF(2^n)$ are used
  - Gray code, multiplications with a constant over a prime field,…
  - allow an easy and clean security proof
  - efficient

Overview

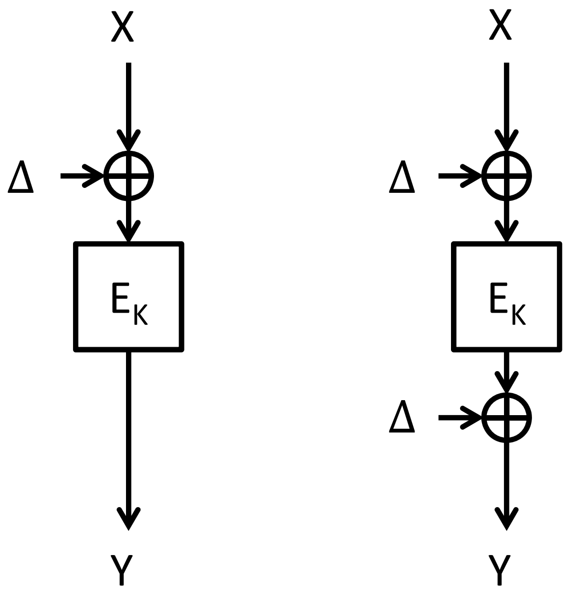
- We show that word permutations and XORs can be used to generate a fixed number of masks
  - can be more efficient depending on the environment
    - similar to a word-oriented LFSR
  - focus on CMAC and EAX
  - can be an option in your design

- [Note] A part of the results will appear in [MiLuIw13]
  - this talk reviews the approach in [MiLuIw13] and presents new concrete examples

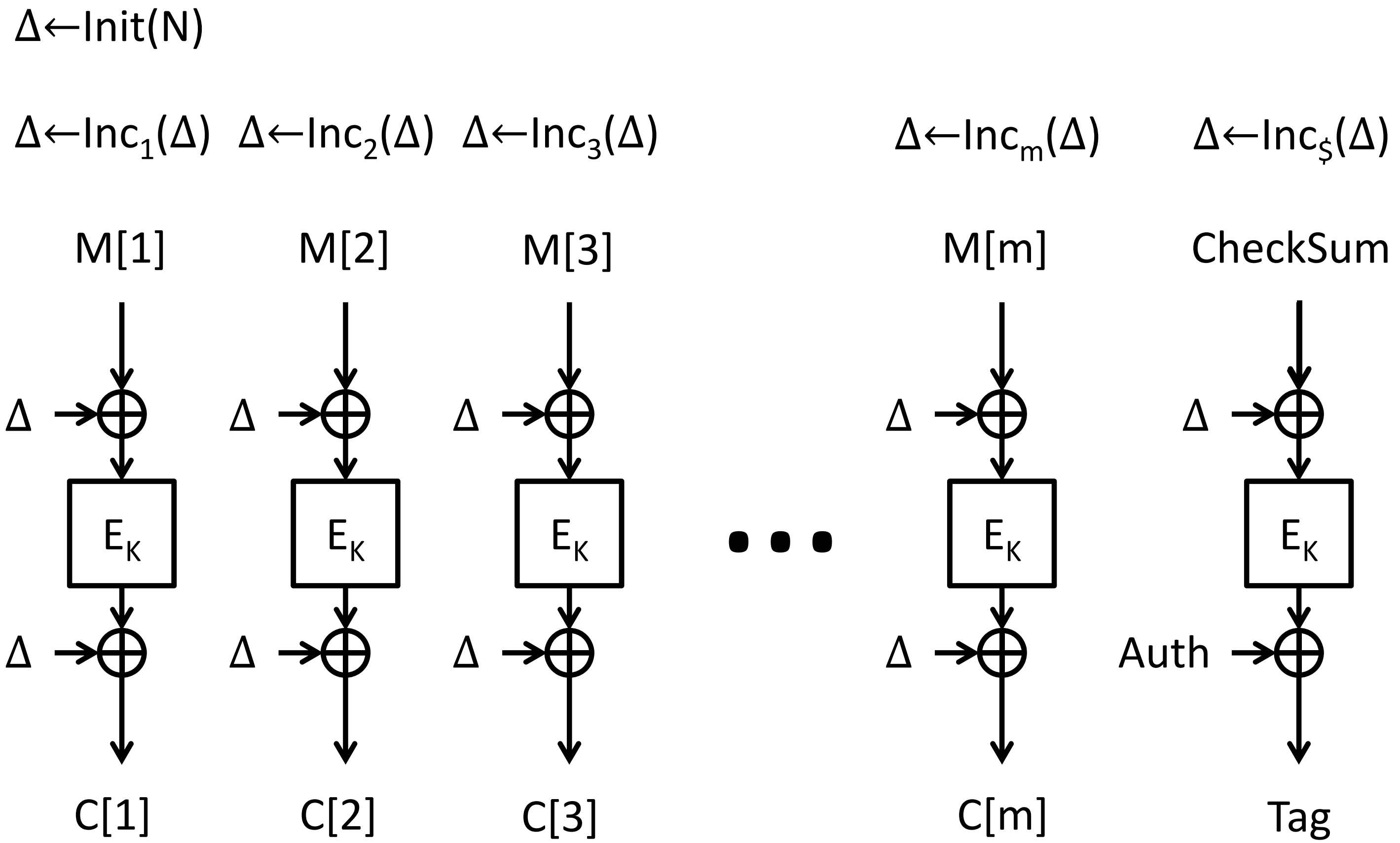
[MiLuIw13] Minematsu, Lucks, Iwata. Improved Authenticity Bound of EAX, and Refinements. ProvSec 2013, to appear.

Masks

- used to "tweak" the input of a blockcipher
  - often XOR is used
  - depends on the key
  - sometimes they are used for the output as well

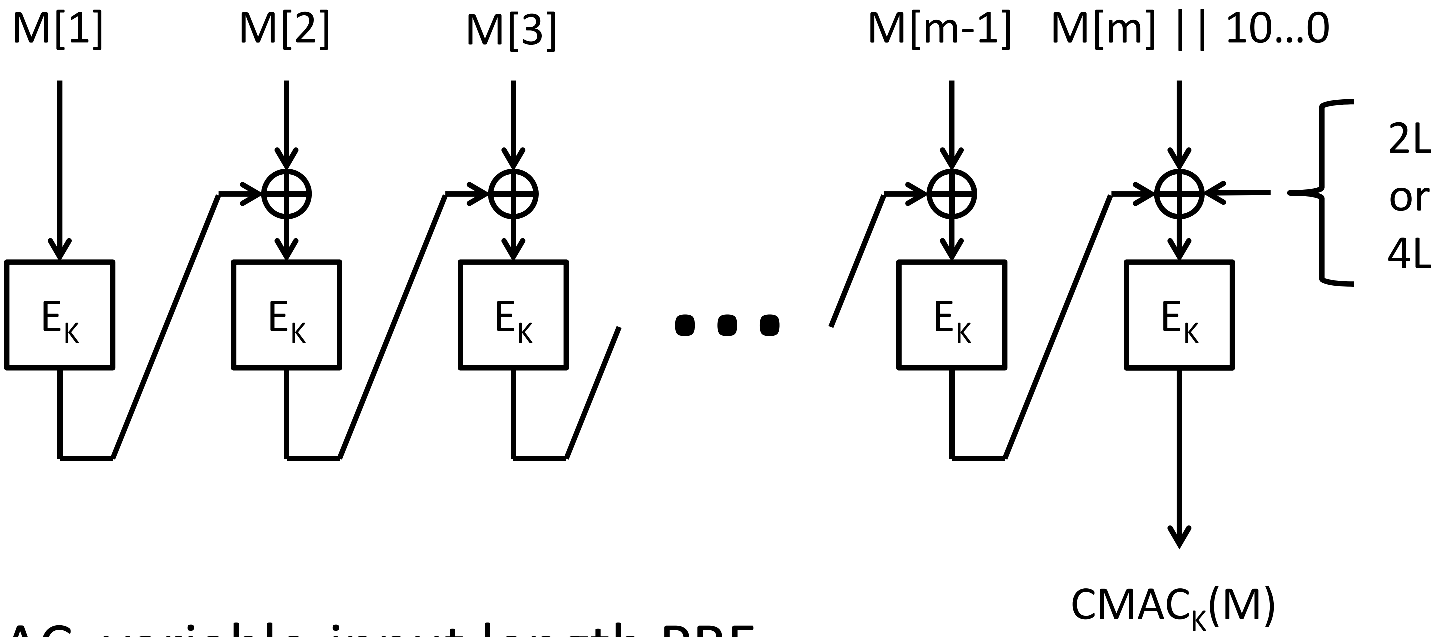


OCB [RoBeBlKr01, Ro04, KrRo11]



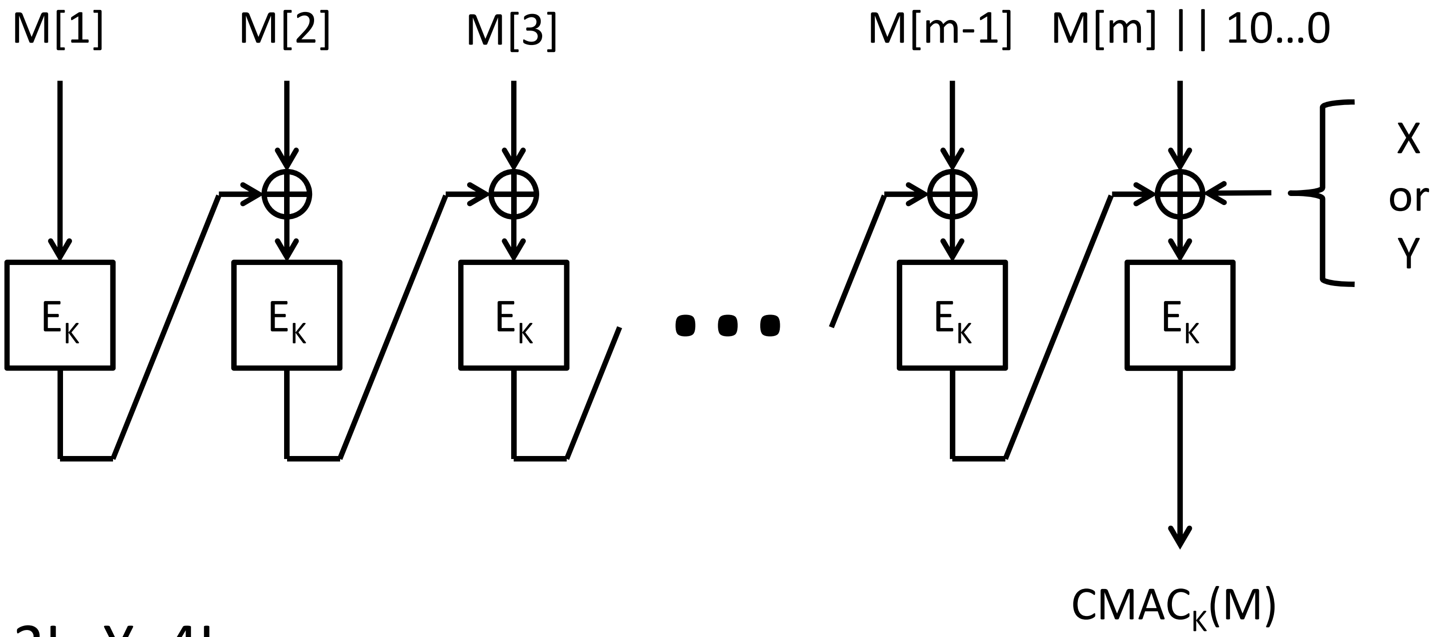
- Gray code, XOR with a pre-computed value
- The number of masks depends on the input length

CMAC [NIST SP 800-38B]



- MAC, variable-input length PRF
- $L=E_K(0^n)$
- 2L: "doubling" of L in $GF(2^n)$
- 4L: 2(2L)

CMAC [NIST SP 800-38B]



- X=2L, Y=4L

Six Conditions on X and Y

- For any n-bit constant c and sufficiently small ε, if L is randomly chosen

$$\begin{cases} \Pr[X = c] \leq \epsilon \\ \Pr[Y = c] \leq \epsilon \\ \Pr[X \oplus Y = c] \leq \epsilon \\ \Pr[X \oplus L = c] \leq \epsilon \\ \Pr[Y \oplus L = c] \leq \epsilon \\ \Pr[X \oplus Y \oplus L = c] \leq \epsilon \end{cases}$$

- These six conditions are sufficient for CMAC being a secure PRF

Six Conditions on X and Y

- with X=2L and Y=4L

$$\begin{cases} \Pr[X = c] \leq \epsilon \\ \Pr[Y = c] \leq \epsilon \\ \Pr[X \oplus Y = c] \leq \epsilon \\ \Pr[X \oplus L = c] \leq \epsilon \\ \Pr[Y \oplus L = c] \leq \epsilon \\ \Pr[X \oplus Y \oplus L = c] \leq \epsilon \end{cases} \Rightarrow \begin{cases} \Pr[2L = c] \leq \epsilon \\ \Pr[4L = c] \leq \epsilon \\ \Pr[6L = c] \leq \epsilon \\ \Pr[3L = c] \leq \epsilon \\ \Pr[5L = c] \leq \epsilon \\ \Pr[7L = c] \leq \epsilon \end{cases}$$

where $\varepsilon=1/2^n$

Breaking L into Words

- block length: n bits
- word length: w bits
- w=n/4 (e.g., (n,w)=(128,32), (64,16))
- $L=(L_1,L_2,L_3,L_4)$
- $L_{[1..4]}=L_1$ xor L_2 xor L_3 xor L_4

$$\begin{cases} X = (L_2, L_3, L_4, L_{[1..4]}) \\ Y = (L_3, L_4, L_{[1..4]}, L_1) \end{cases}$$

Breaking L into Words

- block length: n bits
- word length: w bits
- w=n/4 (e.g., (n,w)=(128,32), (64,16))
- $L=(L_1,L_2,L_3,L_4)$
- $L_{[1..4]}=L_1$ xor L_2 xor L_3 xor L_4

$$\begin{cases} X = (L_2, L_3, L_4, L_{[1..4]}) \\ Y = (L_3, L_4, L_{[1..4]}, L_1) \end{cases}$$

- It works

Breaking L into Words

$$\begin{cases} X = (L_2, L_3, L_4, L_{[1..4]}) \\ Y = (L_3, L_4, L_{[1..4]}, L_1) \end{cases}$$

$$\begin{cases} X = \begin{bmatrix} L_1 \ L_2 \ L_3 \ L_4 \end{bmatrix} \cdot M_X \\ Y = \begin{bmatrix} L_1 \ L_2 \ L_3 \ L_4 \end{bmatrix} \cdot M_Y \end{cases}$$

$$M_X = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \text{ and } M_Y = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

- M_X and M_Y are 4 x 4 matrices over $GF(2^{n/4})$
- full rank

Breaking L into Words

$$
\begin{aligned}
&X \Rightarrow M_X \\
&Y \Rightarrow M_Y \\
&X \oplus Y \Rightarrow M_X \oplus M_Y \\
&X \oplus L \Rightarrow M_X \oplus I \\
&Y \oplus L \Rightarrow M_Y \oplus I \\
&X \oplus Y \oplus L \Rightarrow M_X \oplus M_Y \oplus I
\end{aligned}
$$

the identity matrix

$$
\begin{bmatrix} 0&0&1&0 \\ 1&0&1&1 \\ 1&1&1&1 \\ 0&1&0&1 \end{bmatrix}
\quad
\begin{bmatrix} 1&0&0&1 \\ 1&1&0&1 \\ 0&1&1&1 \\ 0&0&1&0 \end{bmatrix}
\quad
\begin{bmatrix} 1&0&1&1 \\ 0&1&1&0 \\ 1&0&0&0 \\ 0&1&1&1 \end{bmatrix}
\quad
\begin{bmatrix} 1&0&1&0 \\ 1&1&1&1 \\ 1&1&0&1 \\ 0&1&0&0 \end{bmatrix}
$$

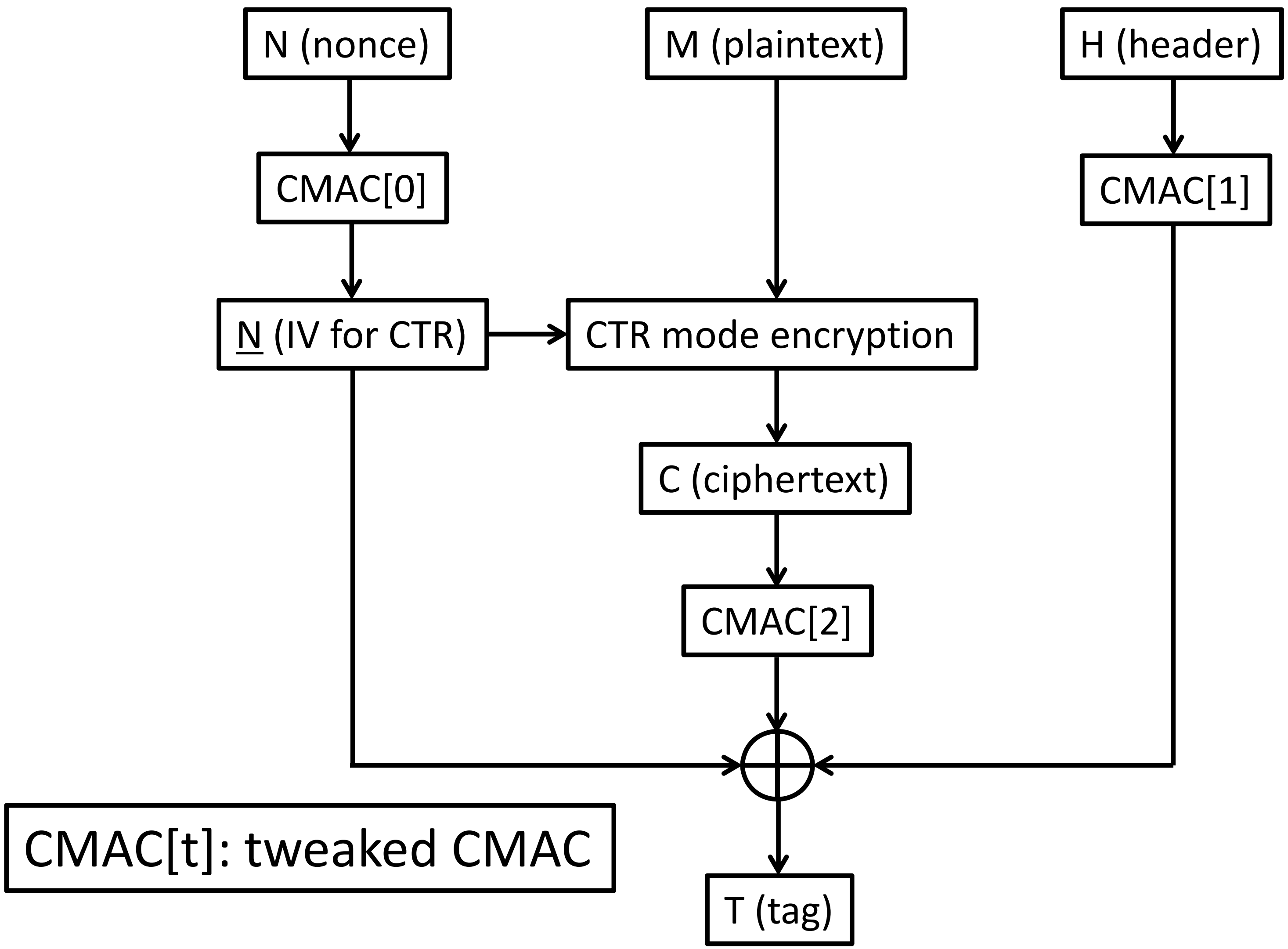
- All six matrices are full rank
- for each condition, one value of L satisfies the equality, $\varepsilon=1/2^n$

Breaking L into Words

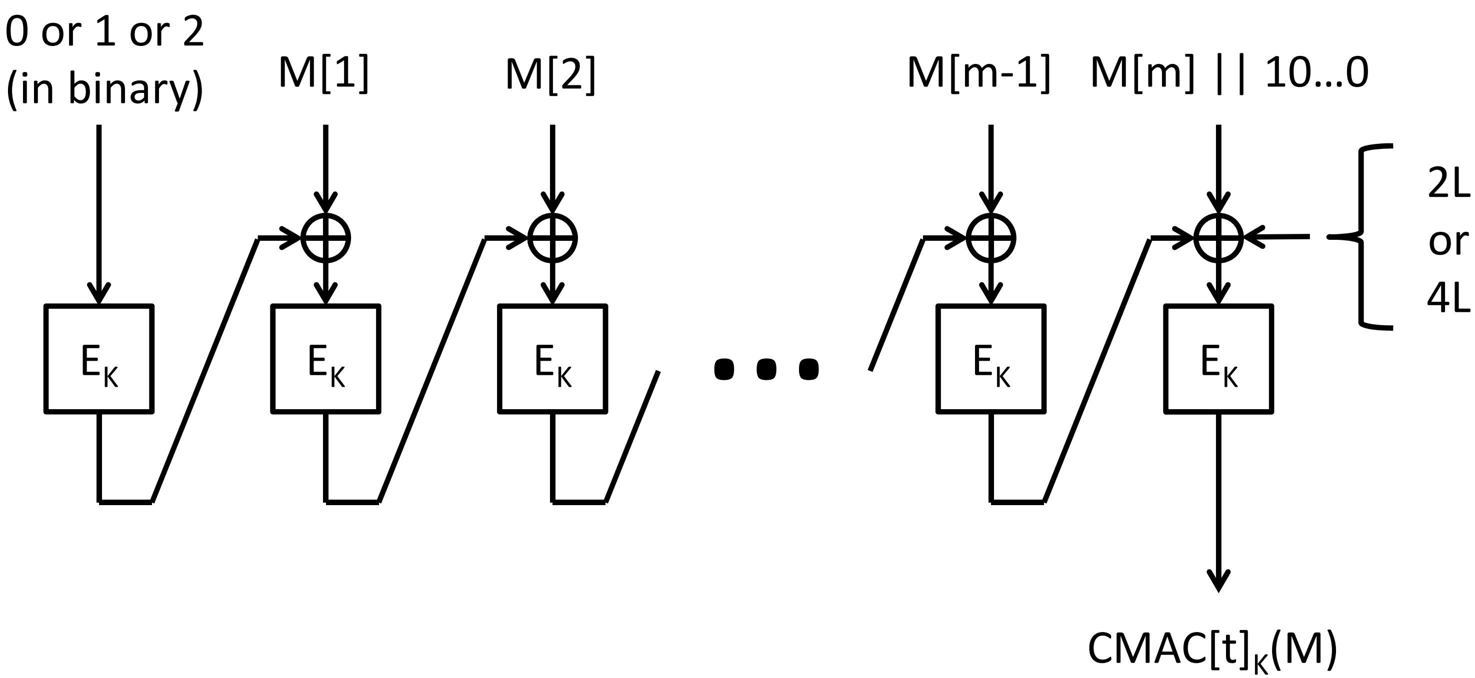
$$\begin{cases} X = (L_2, L_3, L_4, L_{[1..4]}) \\ Y = (L_3, L_4, L_{[1..4]}, L_1) \end{cases}$$

- with (n+n/4)-bit memory
  - store L and $L_{[1..4]}$
  - masks are obtained by a word permutation only
- with n-bit memory
  - store L
  - masks are obtained by a word permutation and three XORs

EAX [BeRoWa04]

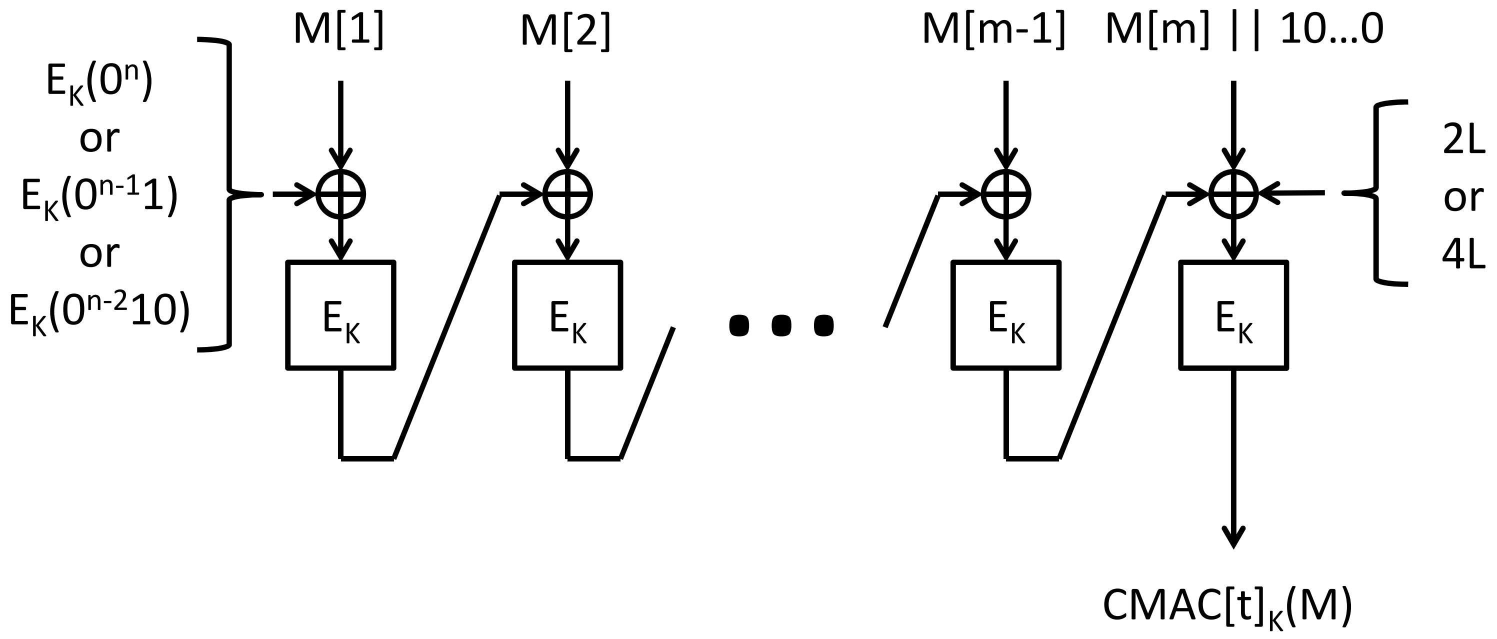


Tweaked CMAC in EAX



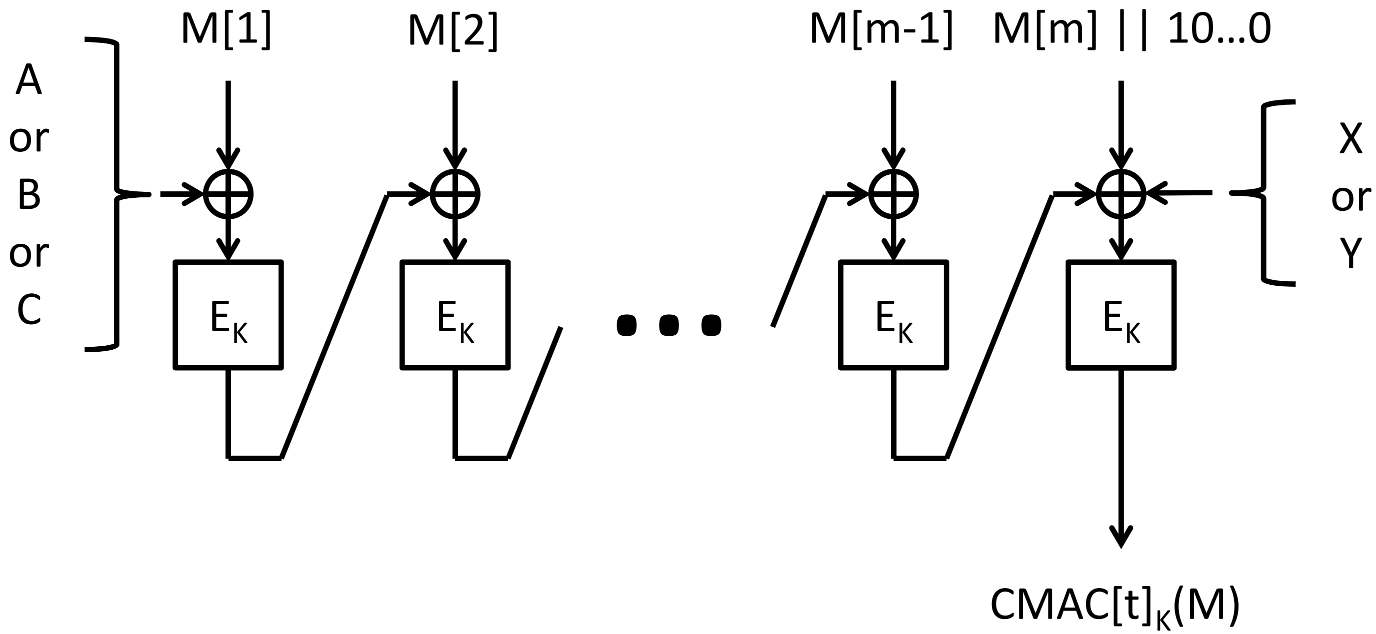
CMAC[0], CMAC[1], CMAC[2]

Tweaked CMAC in EAX



CMAC[0], CMAC[1], CMAC[2]

Tweaked CMAC in EAX



A, B, C, X, and Y Are Masks

- can be pre-computed and stored in memory to optimize the efficiency
  - three blockcipher calls for pre-computation
  - masks are sensitive information (should not be disclosed)
  - memory can be costly
    - resource constrained devices
  - EAX-prime [ANSI C12.22]
    - a slightly modified version of EAX
    - proposed to reduced the pre-computation complexity or memory cost
    - insecure

A, B, C, X, and Y Are Masks

- a fixed number of (five) masks
- desirable to efficiently obtain the five masks from a small amount of memory in any order
  - no need to sequentially generate them
  - unlike word-oriented LFSRs

Twenty Four Conditions [MiLuIw13]

- A, B, C, X, Y are functions of L
- For any n-bit constant c and sufficiently small ε, if L is randomly chosen

$$
\begin{cases}
\Pr[A = c] \leq \epsilon \\
\Pr[B = c] \leq \epsilon \\
\Pr[C = c] \leq \epsilon \\
\Pr[X = c] \leq \epsilon \\
\Pr[Y = c] \leq \epsilon \\
\Pr[A \oplus B = c] \leq \epsilon \\
\Pr[A \oplus C = c] \leq \epsilon \\
\Pr[A \oplus X = c] \leq \epsilon
\end{cases}
\quad
\begin{cases}
\Pr[A \oplus Y = c] \leq \epsilon \\
\Pr[B \oplus C = c] \leq \epsilon \\
\Pr[B \oplus X = c] \leq \epsilon \\
\Pr[B \oplus Y = c] \leq \epsilon \\
\Pr[C \oplus X = c] \leq \epsilon \\
\Pr[C \oplus Y = c] \leq \epsilon \\
\Pr[X \oplus Y = c] \leq \epsilon \\
\Pr[A \oplus B \oplus X = c] \leq \epsilon
\end{cases}
\quad
\begin{cases}
\Pr[A \oplus C \oplus X = c] \leq \epsilon \\
\Pr[B \oplus C \oplus X = c] \leq \epsilon \\
\Pr[A \oplus B \oplus Y = c] \leq \epsilon \\
\Pr[A \oplus C \oplus Y = c] \leq \epsilon \\
\Pr[B \oplus C \oplus Y = c] \leq \epsilon \\
\Pr[A \oplus B \oplus X \oplus Y = c] \leq \epsilon \\
\Pr[A \oplus C \oplus X \oplus Y = c] \leq \epsilon \\
\Pr[B \oplus C \oplus X \oplus Y = c] \leq \epsilon
\end{cases}
$$

- These twenty four conditions are sufficient for EAX being a secure AEAD

Case w=n/4 for EAX (1) [MiLuIw13]

$$\begin{cases} A = (L_1, L_2, L_3, L_4) \\ B = (L_4, L_{[1..4]}, L_1, L_2) \\ C = (L_{[1..4]}, L_1, L_2, L_3) \\ X = (L_2, L_3, L_4, L_{[1..4]}) \\ Y = (L_3, L_4, L_{[1..4]}, L_1) \end{cases}$$

- the first four elements of rotations of $(L_1,L_2,L_3,L_4,L_{[1..4]})$
  - $L=(L_1,L_2,L_3,L_4)$, $L_{[1..4]}=L_1$ xor L_2 xor L_3 xor L_4
- All twenty four matrices are full rank

Case w=n/4 for EAX (1) [MiLuIw13]

$$\begin{cases} A = (L_1, L_2, L_3, L_4) \\ B = (L_4, L_{[1..4]}, L_1, L_2) \\ C = (L_{[1..4]}, L_1, L_2, L_3) \\ X = (L_2, L_3, L_4, L_{[1..4]}) \\ Y = (L_3, L_4, L_{[1..4]}, L_1) \end{cases}$$

- with (n+n/4)-bit memory
  - store $L=E_K(0^n)$ and $L_{[1..4]}$
  - masks are obtained by a word permutation only
- with n-bit memory
  - store L
  - masks are obtained by a word permutation and three XORs

Case w=n/4 for EAX (2) [MiLuIw13]

$$
\begin{cases}
A = (L_1, L_2, L_3, L_4) \\
B = (L_2, L_{[1,2]}, L_4, L_{[3,4]}) \\
C = (L_{[1,2]}, L_1, L_{[3,4]}, L_3) \\
X = (L_3, L_{[3,4]}, L_2, L_1) \\
Y = (L_4, L_3, L_{[1,2]}, L_2)
\end{cases}
$$

- $L_{[a,b]}=L_a$ xor L_b
- All twenty four matrices are full rank
- Searched for (limited) space, picked one that “looks good”
  - small memory to implement, small number of XORs
- X and Y can be used for CMAC as well

Case w=n/4 for EAX (2) [MiLuIw13]

$$\begin{cases} A = (L_1, L_2, L_3, L_4) \\ B = (L_2, L_{[1,2]}, L_4, L_{[3,4]}) \\ C = (L_{[1,2]}, L_1, L_{[3,4]}, L_3) \\ X = (L_3, L_{[3,4]}, L_2, L_1) \\ Y = (L_4, L_3, L_{[1,2]}, L_2) \end{cases}$$

- with (n+2 x n/4)-bit memory
  - store L and $L_{[1,2]}$ and $L_{[3,4]}$
  - masks are obtained by a word permutation only
- with n-bit memory
  - store L
  - masks are obtained by a word permutation and two XORs

So Far, w=n/4

- w=n/4
  - (n,w)=(128,32), (64,16)
- w=n/8
  - (n,w)=(128,16), (64,8)
- w=n/16
  - (n,w)=(128,8)

Case w=n/8 for EAX (1)

$$\begin{cases} A = (L_1, \ldots, L_8) \\ B = (L_4, L_{[1..4]}, L_1, L_2, L_8, L_{[5..8]}, L_5, L_6) \\ C = (L_{[1..4]}, L_1, L_2, L_3, L_{[5..8]}, L_5, L_6, L_7) \\ X = (L_2, L_3, L_4, L_{[1..4]}, L_6, L_7, L_8, L_{[5..8]}) \\ Y = (L_3, L_4, L_{[1..4]}, L_1, L_7, L_8, L_{[5..8]}, L_5) \end{cases}$$

- applied the previous method (of using $L_{[1..4]}$=L_1 xor L_2 xor L_3 xor L_4) to (L_1,L_2,L_3,L_4) and (L_5,L_6,L_7,L_8) independently
- All twenty four matrices are full rank
- X and Y can be used for CMAC

Case w=n/8 for EAX (1)

$$\begin{cases} A = (L_1, \ldots, L_8) \\ B = (L_4, L_{[1..4]}, L_1, L_2, L_8, L_{[5..8]}, L_5, L_6) \\ C = (L_{[1..4]}, L_1, L_2, L_3, L_{[5..8]}, L_5, L_6, L_7) \\ X = (L_2, L_3, L_4, L_{[1..4]}, L_6, L_7, L_8, L_{[5..8]}) \\ Y = (L_3, L_4, L_{[1..4]}, L_1, L_7, L_8, L_{[5..8]}, L_5) \end{cases}$$

- applied the previous method (of using $L_{[1..4]}=L_1$ xor L_2 xor L_3 xor L_4) to (L_1,L_2,L_3,L_4) and (L_5,L_6,L_7,L_8) independently
- All twenty four matrices are full rank
- X and Y can be used for CMAC

Case w=n/8 for EAX (1)

$$\begin{cases} A = (L_1, \ldots, L_8) \\ B = (L_4, L_{[1..4]}, L_1, L_2, L_8, L_{[5..8]}, L_5, L_6) \\ C = (L_{[1..4]}, L_1, L_2, L_3, L_{[5..8]}, L_5, L_6, L_7) \\ X = (L_2, L_3, L_4, L_{[1..4]}, L_6, L_7, L_8, L_{[5..8]}) \\ Y = (L_3, L_4, L_{[1..4]}, L_1, L_7, L_8, L_{[5..8]}, L_5) \end{cases}$$

- with (n+2 x n/8)-bit memory
  - store L and $L_{[1..4]}$ and $L_{[5..8]}$
  - masks are obtained by a word permutation only
- with n-bit memory
  - store L
  - masks are obtained by a word permutation and six XORs

Case w=n/8 for EAX (1)

$$\begin{cases} A = (L_1, \ldots, L_8) \\ B = (L_4, L_{[1..4]}, L_1, L_2, L_8, L_{[5..8]}, L_5, L_6) \\ C = (L_{[1..4]}, L_1, L_2, L_3, L_{[5..8]}, L_5, L_6, L_7) \\ X = (L_2, L_3, L_4, L_{[1..4]}, L_6, L_7, L_8, L_{[5..8]}) \\ Y = (L_3, L_4, L_{[1..4]}, L_1, L_7, L_8, L_{[5..8]}, L_5) \end{cases}$$

- can be used for the cases w=n/4j for any $j \geq 1$
  - break L into $(L_1, L_2, \ldots, L_{4j})$
  - apply to (L_1, L_2, L_3, L_4), (L_5, L_6, L_7, L_8), …, $(L_{4j-3}, L_{4j-2}, L_{4j-1}, L_{4j})$ independently

Case w=n/8 for EAX (2)

$$\begin{cases} A = (L_1, \ldots, L_8) \\ B = (L_2, L_{[1,2]}, L_4, L_{[3,4]}, L_6, L_{[5,6]}, L_8, L_{[7,8]}) \\ C = (L_{[1,2]}, L_1, L_{[3,4]}, L_3, L_{[5,6]}, L_5, L_{[7,8]}, L_7) \\ X = (L_3, L_{[3,4]}, L_2, L_1, L_7, L_{[7,8]}, L_6, L_5) \\ Y = (L_4, L_3, L_{[1,2]}, L_2, L_8, L_7, L_{[5,6]}, L_6) \end{cases}$$

- applied the previous method (of using $L_{[a,b]}=L_a$ xor L_b) to (L_1,L_2,L_3,L_4) and (L_5,L_6,L_7,L_8) independently
- All twenty four matrices are full rank
- X and Y can be used for CMAC

Case w=n/8 for EAX (2)

$$\begin{cases} A = (L_1, \dots, L_8) \\ B = (L_2, L_{[1,2]}, L_4, L_{[3,4]}, L_6, L_{[5,6]}, L_8, L_{[7,8]}) \\ C = (L_{[1,2]}, L_1, L_{[3,4]}, L_3, L_{[5,6]}, L_5, L_{[7,8]}, L_7) \\ X = (L_3, L_{[3,4]}, L_2, L_1, L_7, L_{[7,8]}, L_6, L_5) \\ Y = (L_4, L_3, L_{[1,2]}, L_2, L_8, L_7, L_{[5,6]}, L_6) \end{cases}$$

- with (n+4 x n/8)-bit memory
  - store L and $L_{[1,2]}$ and $L_{[3,4]}$ and $L_{[5,6]}$ and $L_{[7,8]}$
  - masks are obtained by a word permutation only
- with n-bit memory
  - store L
  - masks are obtained by a word permutation and four XORs

Case w=n/8 for EAX

- Interestingly, taking the first eight elements of the rotations of $(L_1,\ldots,L_8,L_{[1..8]})$ does not work

$$\begin{cases} A = (L_1, \ldots, L_8) \\ B = (L_4, \ldots, L_8, L_{[1..8]}, L_1, L_2) \\ C = (L_5, \ldots, L_8, L_{[1..8]}, L_1, L_2, L_3) \\ X = (L_2, \ldots, L_8, L_{[1..8]}) \\ Y = (L_3, \ldots, L_8, L_{[1..8]}, L_1) \end{cases}$$

- X and Y do not work for CMAC

Case w=n/16 for EAX (1)

- Taking the first sixteen elements of the rotations of $(L_1,\ldots,L_{16},L_{[1..16]})$ works

$$\begin{cases} A = (L_1, \ldots, L_{16}) \\ B = (L_4, \ldots, L_{16}, L_{[1..16]}, L_1, L_2) \\ C = (L_5, \ldots, L_{16}, L_{[1..16]}, L_1, L_2, L_3) \\ X = (L_2, \ldots, L_{16}, L_{[1..16]}) \\ Y = (L_3, \ldots, L_{16}, L_{[1..16]}, L_1) \end{cases}$$

- a word permutation only with (n+n/16)-bit memory
  - store L and $L_{[1..16]}$
- with n-bit memory, 15 XORs are needed (if we store L)
- X and Y work for CMAC

Case w=n/16 for EAX (2)

- Construction that “looks good” (from searching limited space)

$$\begin{cases} A = (L_1, \ldots, L_{16}) \\ B = (L_4, \ldots, L_{16}, L_{[1,2]}, L_{[2,3]}, L_{[3,4]}) \\ C = (L_5, \ldots, L_{16}, L_{[1,2]}, L_{[2,3]}, L_{[3,4]}, L_{[4,5]}) \\ X = (L_2, \ldots, L_{16}, L_{[1,2]}) \\ Y = (L_3, \ldots, L_{16}, L_{[1,2]}, L_{[2,3]}) \end{cases}$$

- a word permutation only if (n+4 x n/16)-bit memory
  - store L and $L_{[1,2]}$ and $L_{[2,3]}$ and $L_{[3,4]}$ and $L_{[4,5]}$
- with n-bit memory
  - store L
  - masks are obtained by a word permutation and four XORs

Summary of Mask Generation for EAX

- w=n/4

	Perm. only if	with n-bit memory	ref.
(1)	n + n/4	permutation + three XORs	[MiLuIw13]
(2)	n + 2 x n/4	permutation + two XORs	[MiLuIw13]

- w=n/8

	Perm. only if	with n-bit memory	
(1)	n + 2 x n/8	permutation + six XORs	
(2)	n + 4 x n/8	permutation + four XORs	

- w=n/16

	Perm. only if	with n-bit memory	
(1)	n + n/16	permutation + 15 XORs	
(2)	n + 4 x n/16	permutation + four XORs	

Summary

- Considered a problem of generating a fixed number of masks used in CMAC and EAX
- Demonstrated that the approach can be used to reduce the pre-computation complexity or memory cost with various word lengths
- Optimality of the examples in this talk is open, but generating examples is not hard (just to see if the matrices are full rank)
  - how we can obtain good constructions is open
- can be an option in your design
  - formalizing the sufficient conditions may not be easy