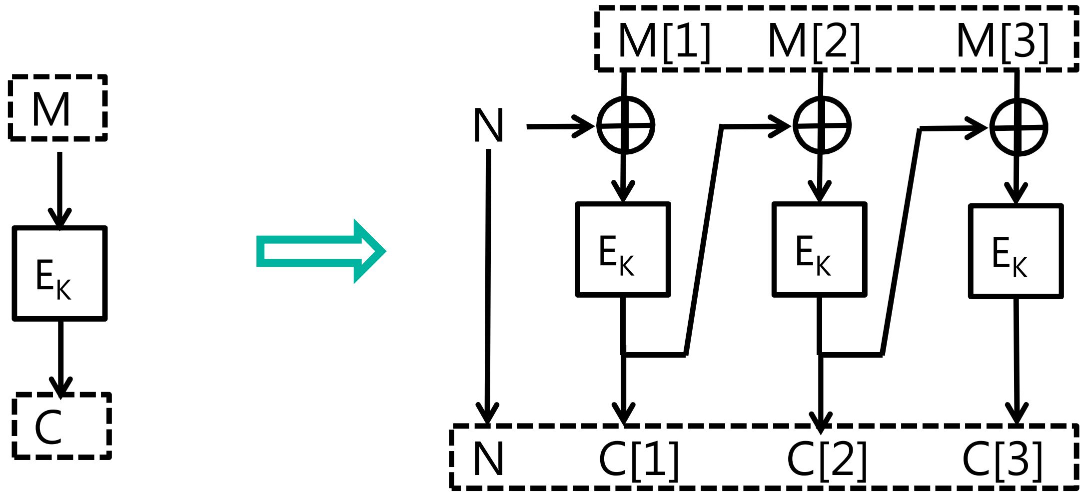
Design approach to efficient blockcipher modes

Kazuhiko Minematsu, NEC Corporation

The Fourth Asian Workshop on Symmetric Key Cryptography,
19-22, December 2014, SETS Chennai, India

Introduction

- Blockcipher mode : turning a blockcipher (BC) into a more usable function
- Ex. CBC encryption mode seen as a conversion of fixed-length encryption into variable-length encryption



Designing modes

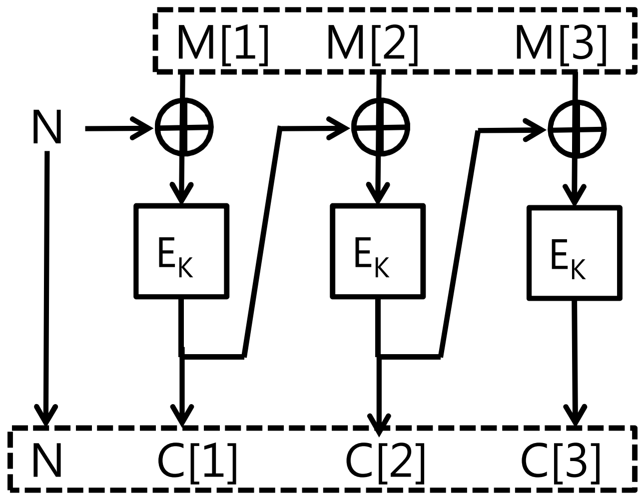
- Designing secure and optimized BC mode is generally a complex task
- This talk will show some useful ideas to reduce this complexity, with applications to authenticated encryption (AE)

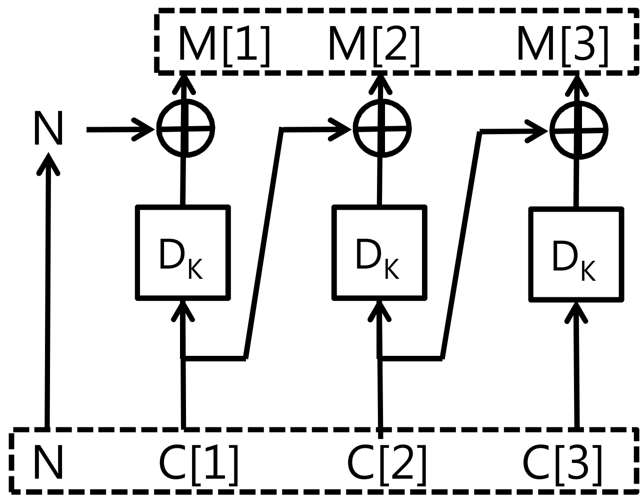
- The first part is about “inverse-free” mode, and a corresponding CAESAR candidate, OTR
- The second part is about “direct tweaking” and a corresponding CAESAR candidate, CLOC and SILC

Removing Blockcipher Inverse

Modes w/ BC inverse

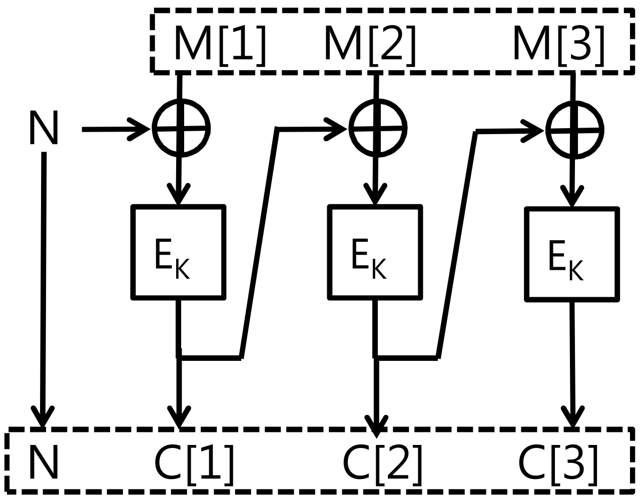
- Some blockcipher modes use blockcipher inverse (decryption)
- Ex. CBC mode needs BC inverse (D_K) for the decryption

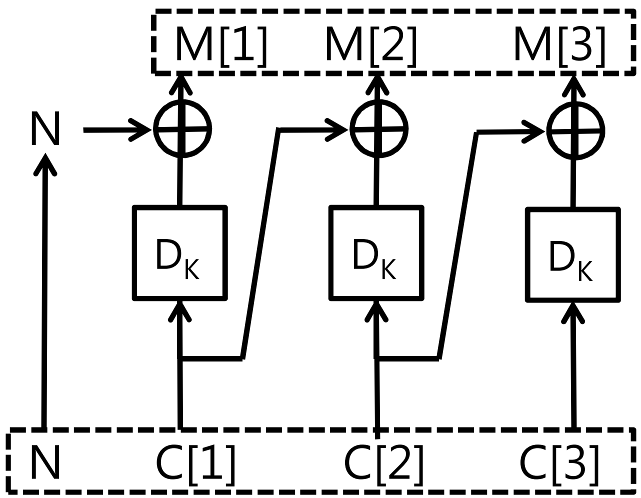




Our task

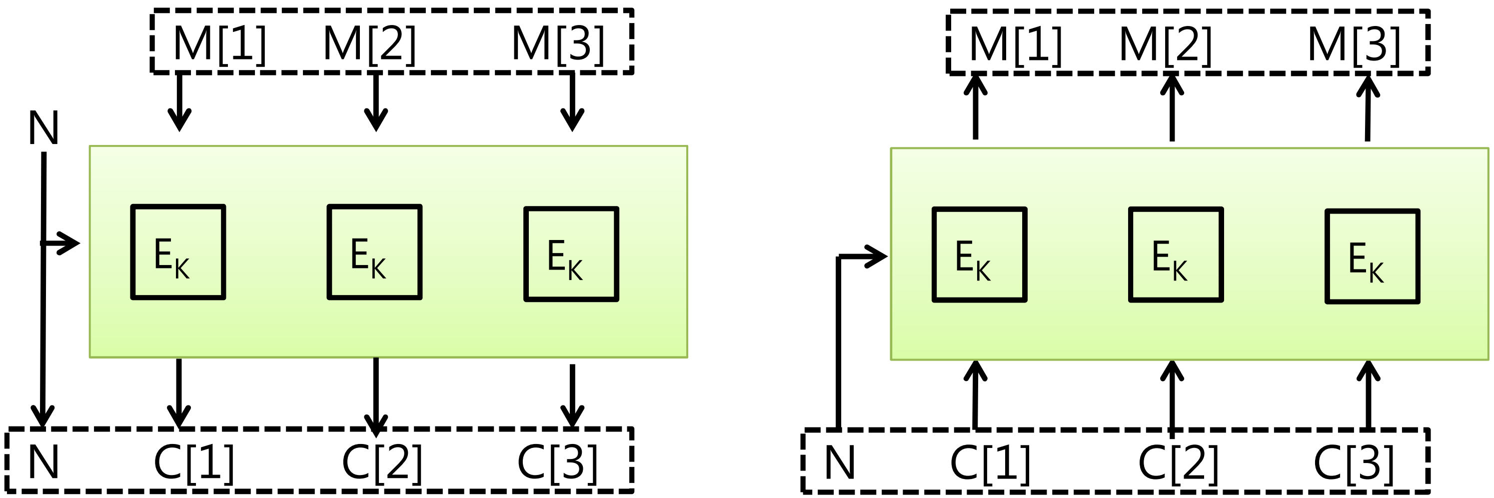
- Given a target mode which needs BC inverse,
- Modify it to inverse-free,
- Keeping features as much as possible
  - I/O format
  - # of primitive calls
  - security properties
  - implementation options (e.g. parallelizability)





Our task

- Given a target mode which needs BC inverse,
- Modify it to inverse-free,
- Keeping features as much as possible
  - I/O format
  - # of primitive calls
  - security properties
  - implementation options (e.g. parallelizability)



Advantages of removing inverse

- We have several reasons for it, taking AES for example
- Size benefit
  - Hardware gate : ~10K additional gates for AES-decryption core
  - Software memory reduction
    - Inverse S-box , inverse T-tables etc.
- Speed benefit
  - For some platforms AES-dec is slower than AES-enc (due to the difference between MixCol and InvMixCol)
  - Ex. Byte-wise AES on 8-bit MCU : ~20 to 50 % slowdown
  - Some SIMD codes on High-end CPU
    - Bitslice or Vector-permutation
    - Not true for AES-NI

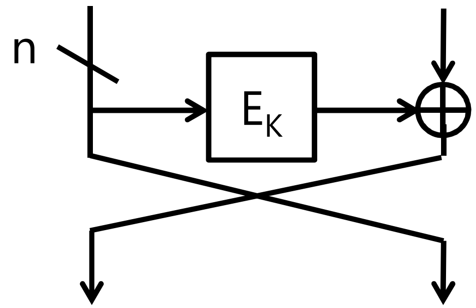
Advantages of removing inverse

- Security benefit
  - For modes w/ BC inverse, BC is (generally) required to be secure against **Chosen-ciphertext attack (CCA)**
    - Strong pseudorandom permutation (SPRP)
  - For inverse-free modes, we need a weaker assumption, **Chosen-plaintext attack (CPA)** security
    - PRP or psedorandom function (PRF)

- Others
  - Enables the use of non-invertible primitives, e.g. HMAC

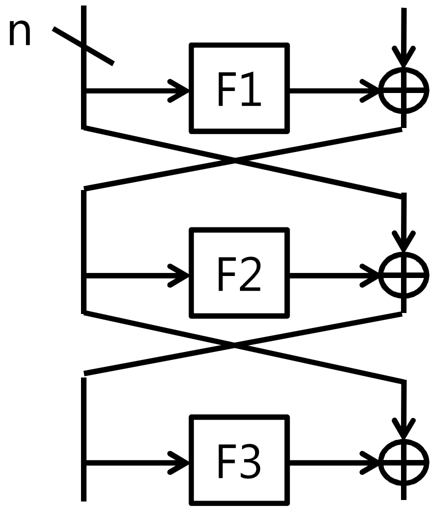
Basic idea

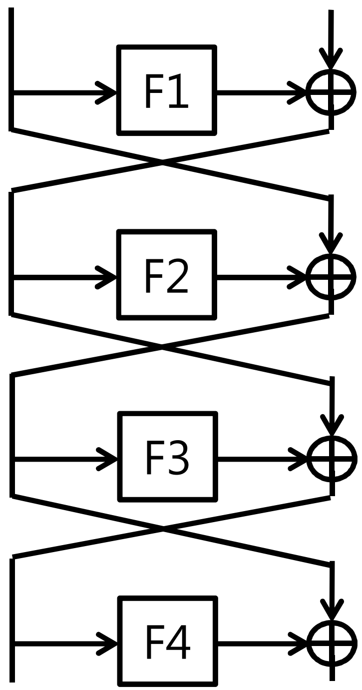
- A classical way to implement cryptographic permutation using cryptographic functions
- Feistel !
- More formally, we implement 2n-bit permutation by iterating a Feistel permutation having n-bit blockcipher as round function
- Also called Luby-Rackoff cipher (LRC)



Security of LR Cipher

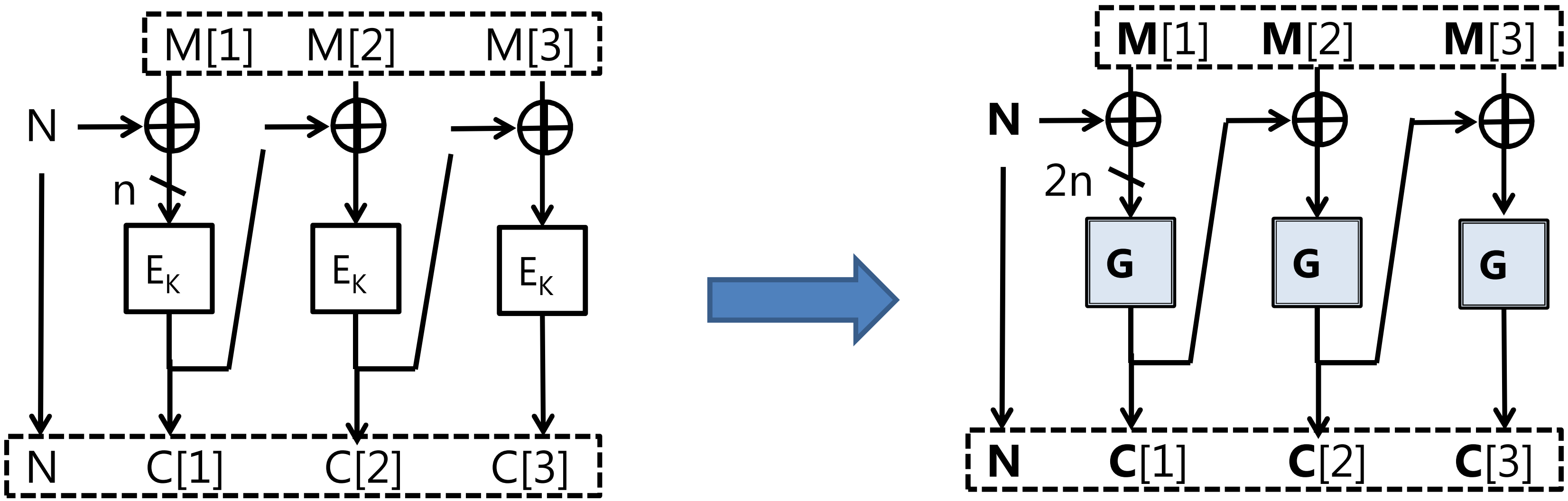
- Brief review of Luby-Rackoff
- Assuming each round function is an independent PRF,
- 3-round LRC is CPA-secure (i.e. a PRP)
- 4-round LRC is CCA-secure (i.e. a SPRP)
- For both cases, distinguishing advantage from 2n-bit random permutation is $O(q^2/2^n)$ for q queries





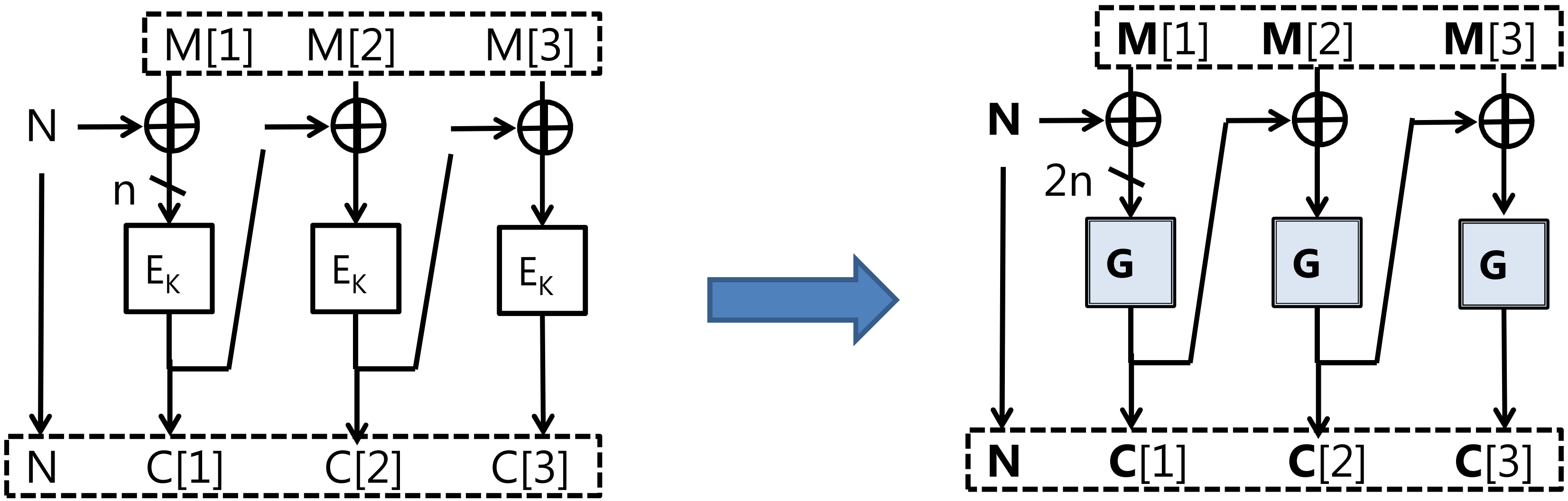
Inverse-removal : Basic Approach

- Find a target mode (say CBC)
- Step 1 . Define a 2-block version of CBC, using a 2n-bit blockcipher **G**



Inverse-removal : Basic Approach

- Step 2. Find the *exact* security condition for **G** to keep the *original* security bounds w.r.t n
  - typically birthday bound, i.e. $O(q^2/2^n)$

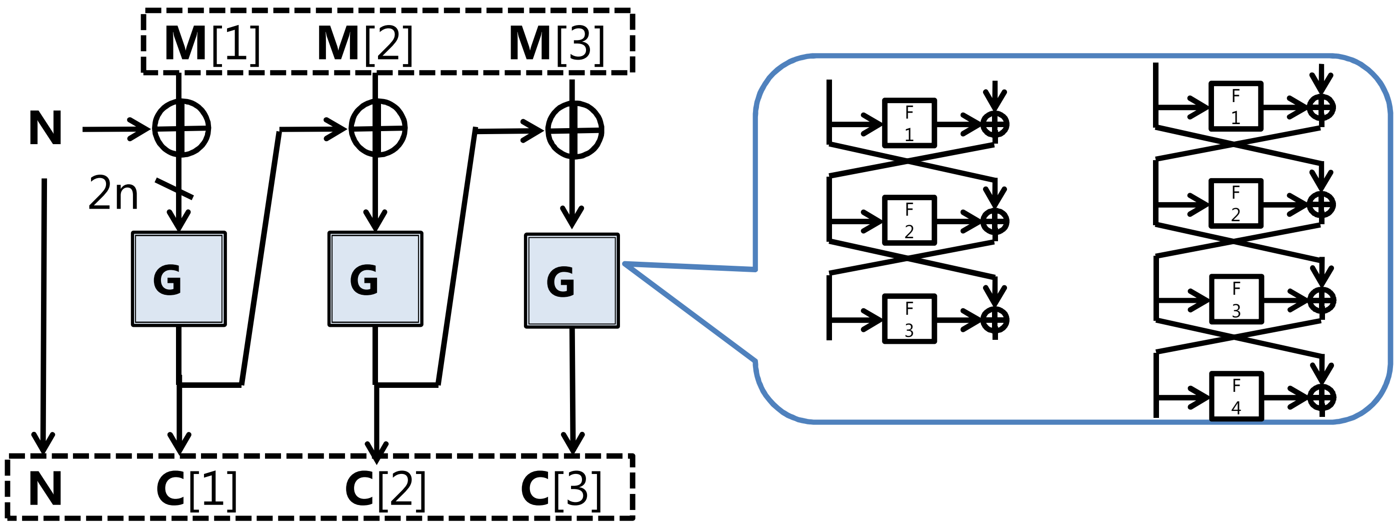


E = PRP

G = …. ?

Inverse-removal : Basic Approach

- Step 3. Instantiate **G** by LRC w/ forward BC function, then find # of rounds meeting the security condition
- 4-round is usually enough[1], but we often find a smaller-round is secure
- May need further modifications…



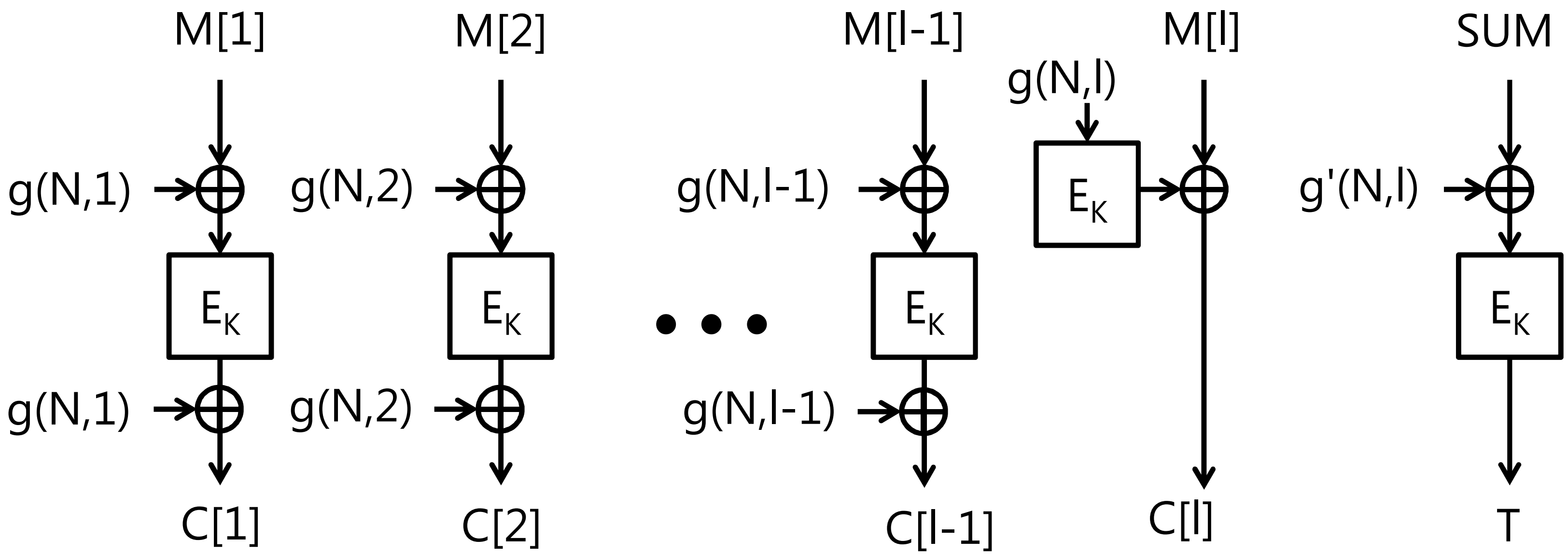
1 As long as the original security is birthday-bound security based on SPRP assumption

Case of Authenticated Encryption

- We focus on authenticated encryption (AE), which provides confidentiality and integrity
- We consider nonce-based AE
  - Each encryption takes unique nonce N
  - Plaintext M is encrypted to Ciphertext C, with Tag T, where |M| = |C|
  - Additionally we may have Associated Data (AD) as information not encrypted but MACed
- The target is OCB mode, which is a seminal nonce-based AE developed by Rogaway (et al.)

OCB (simplified)

- Encryption = ECB w/ mask
- MAC = encryption of plaintext checksum (XORs of plaintext blocks)
- Mask is a function of (nonce, block index), and Key
  - Needs one BC call to produce all masks

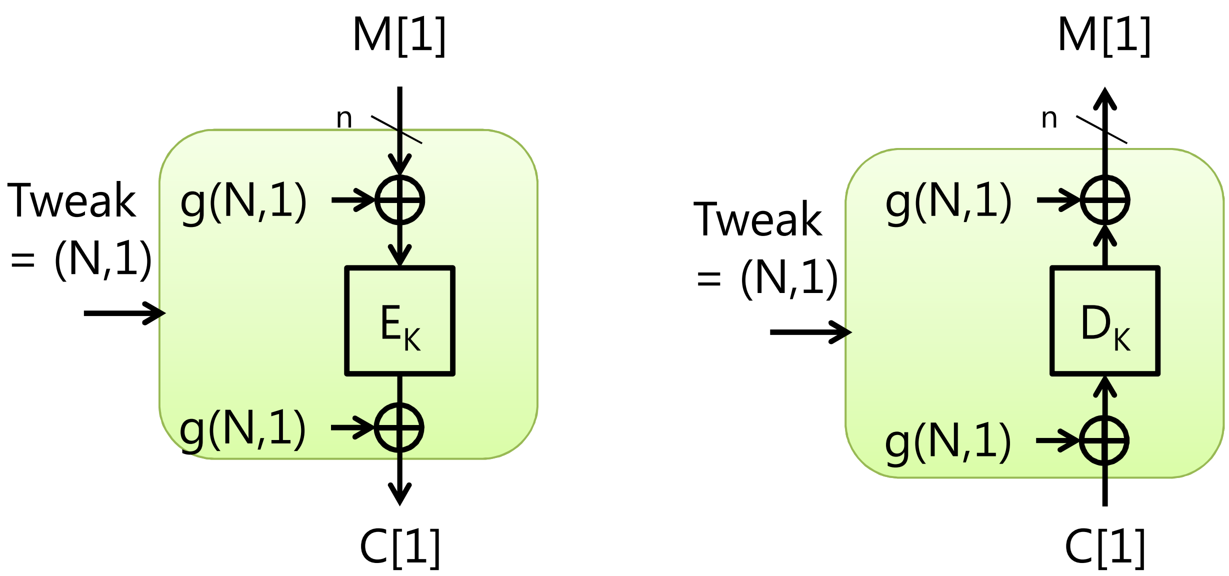


Mask function
$g(N,i) = E_K(N) \times 2^i$ (over $GF(2^n)$) for OCB2

SUM
$= M[1] \oplus M[2] \oplus \ldots \oplus M[l]$

Security of OCB

- Mask-Enc-Mask can be seen as an instance of Tweakable BC (Tweak = (N,i))
- OCB proof requires CCA-security for this TBC
  – (Tweakable SPRP, TSPRP)



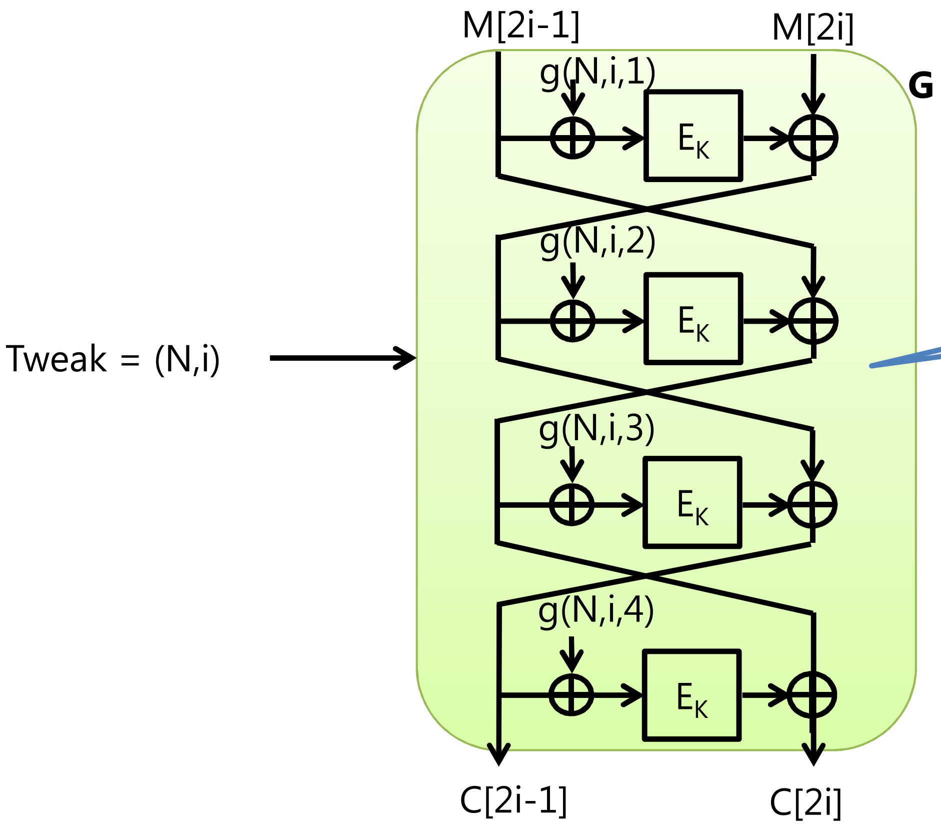
Features of OCB

OCB has a number of strong features

- Rate-1 : 1 BC call for 1 input block
  - Here rate = # of BC calls for 1 input block
- Parallelizable for encryption and decryption
- On-line processing
- Provable security based on the assumption BC = SPRP
  - Security up to birthday bound – advantage $O(\sigma^2/2^n)$ for privacy/authenticity notions, for σ blocks in queries
- But it needs BC inverse for decryption

Removing Inverse from OCB

- Step 1: set OCB for 2n-bit LRC **G**
  - Each round takes a mask g(N,block index, round index)
- **G** itself takes tweak (N, block index)
- If we follow OCB proof, **G** needs to be 2n-bit TSPRP w/ adv. $O(q^2/2^n)$ -> **G** should be 4-round LRC



Trivially works, but rate is 2 !

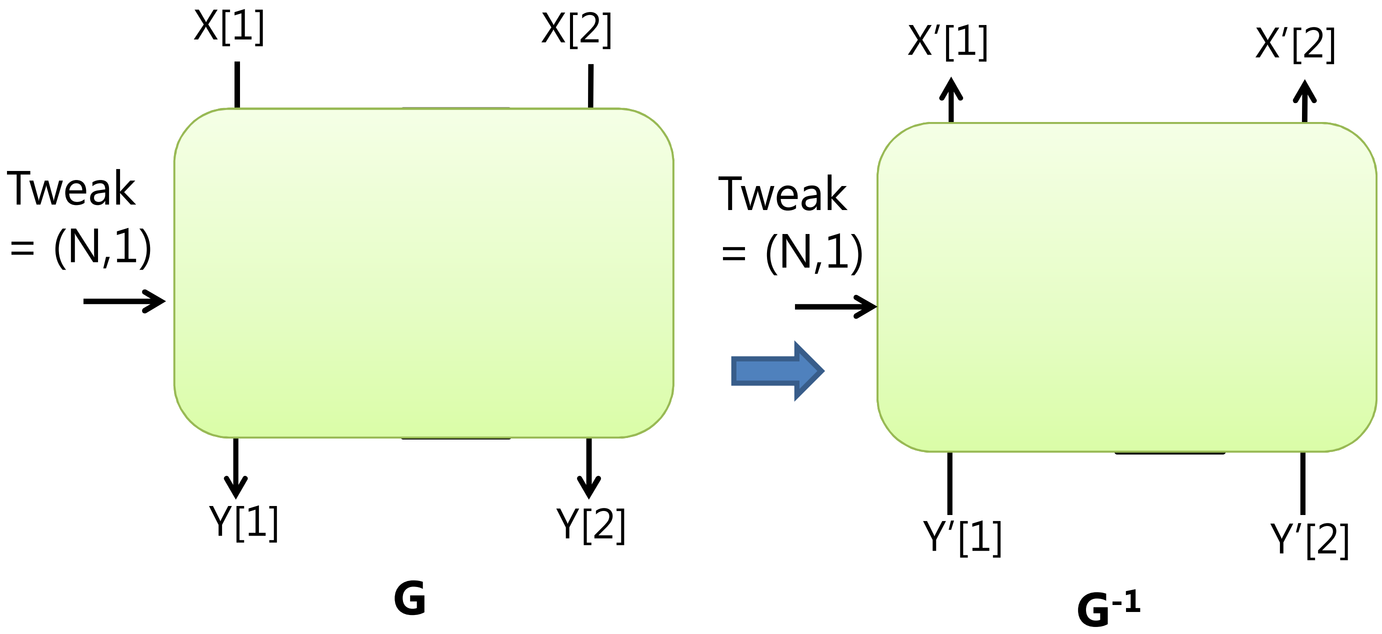
Removing Inverse from OCB

- Step 2: we found the exact condition on **G**, which is as follows
- For each tweak (N,i), (let us set i=1)

1 An encryption query (X[1],X[2]) generates random output (Y[1],Y[2])

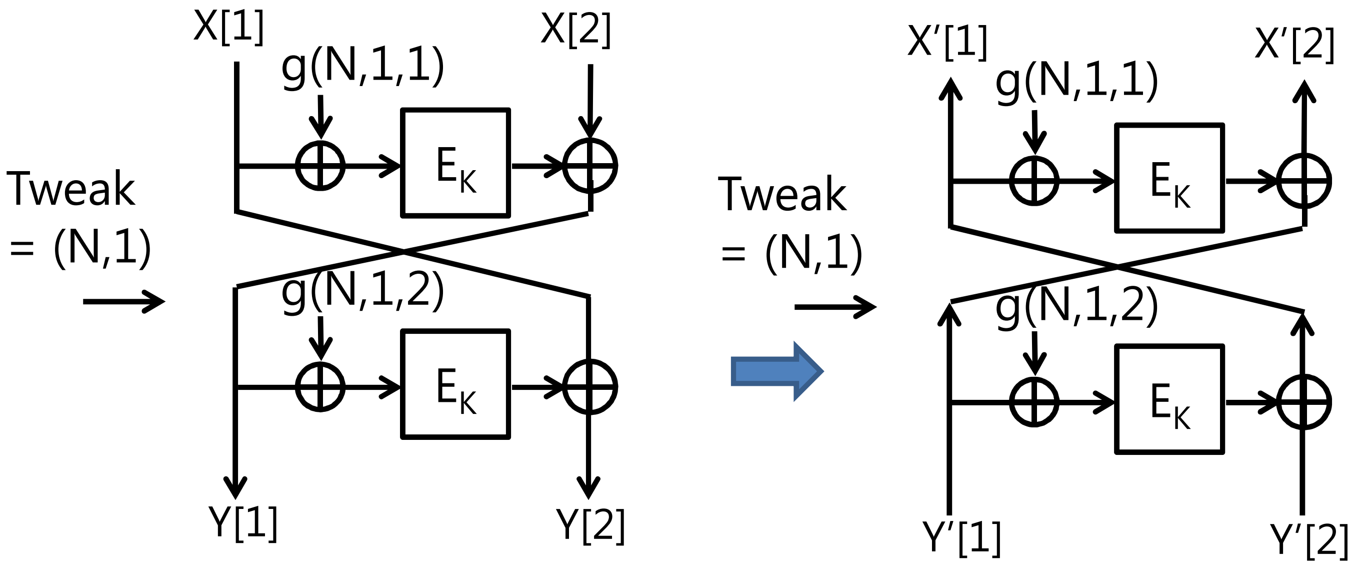
2 Given (X[1],X[2]) and (Y[1],Y[2]), decryption query (Y'[1],Y'[2]) not equal to (Y[1],Y[2]) generates an n-bit unpredictable part in the output (X'[1],X'[2])

- Allowing distinguishing bias of $O(q^2/2^n)$



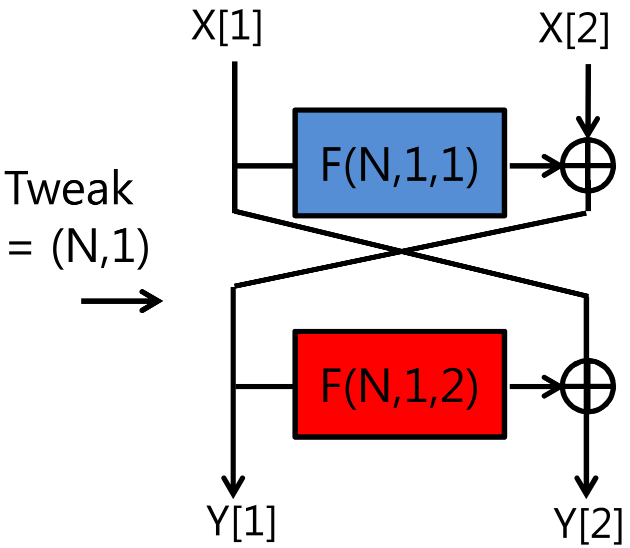
Using 2-round is enough

- Step 3 : find the minimum # of rounds:
- The conditions are about one enc-query and dec-query for one tweak
- And these conditions are satisfied with 2-round LRC. Why?



Using 2-round is enough

- Admitting bias $O(q^2/2^n)$, round functions can be seen as independent random functions
- Then, (Y[1],Y[2]) is uniformly random

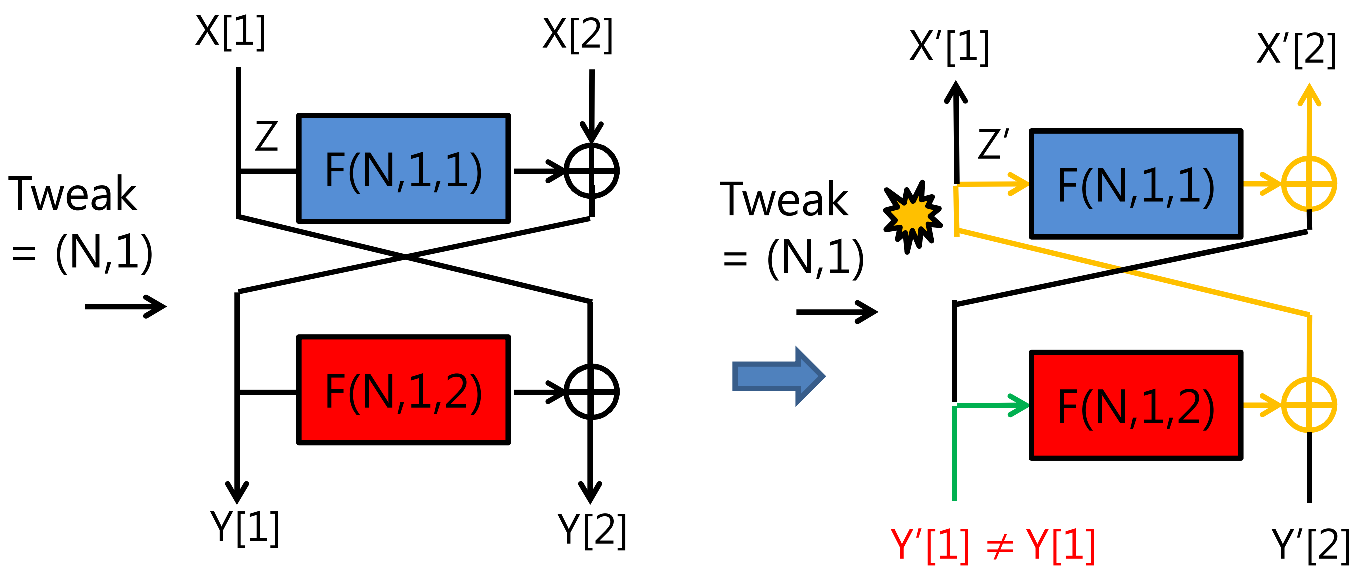


2n-bit randomness for an enc-query

Using 2-round is enough

- Given (X[1],X[2])(Y[1],Y[2]), and dec query (Y'[1],Y'[2]), we have two cases :
- When Y'[1] ≠ Y[1], X'[2] is independent and random
  - Unless Z' collides with Z
  - Z'= Z occurs with prob. $1/2^n$

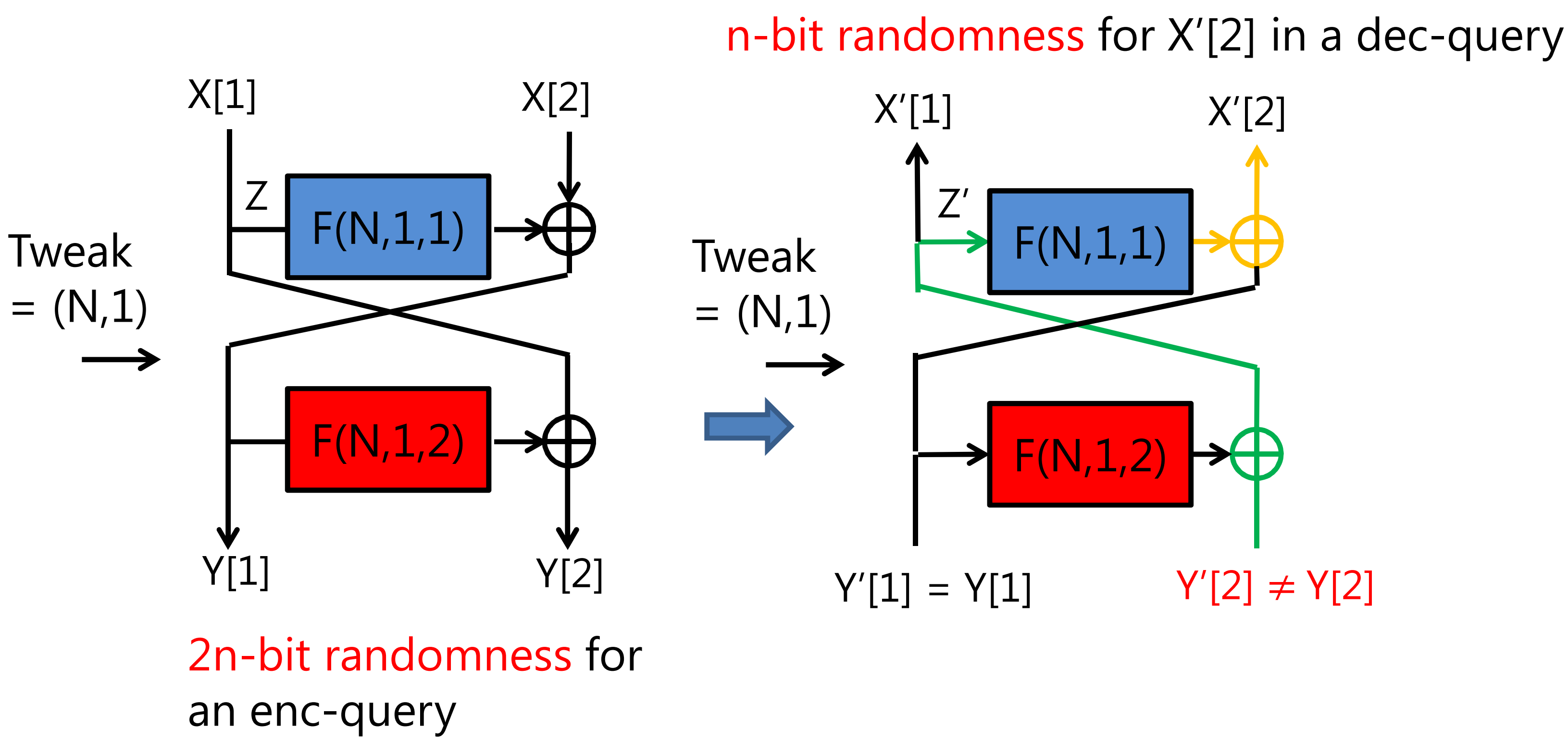
n-bit randomness for X'[2] in a dec-query



2n-bit randomness for an enc-query

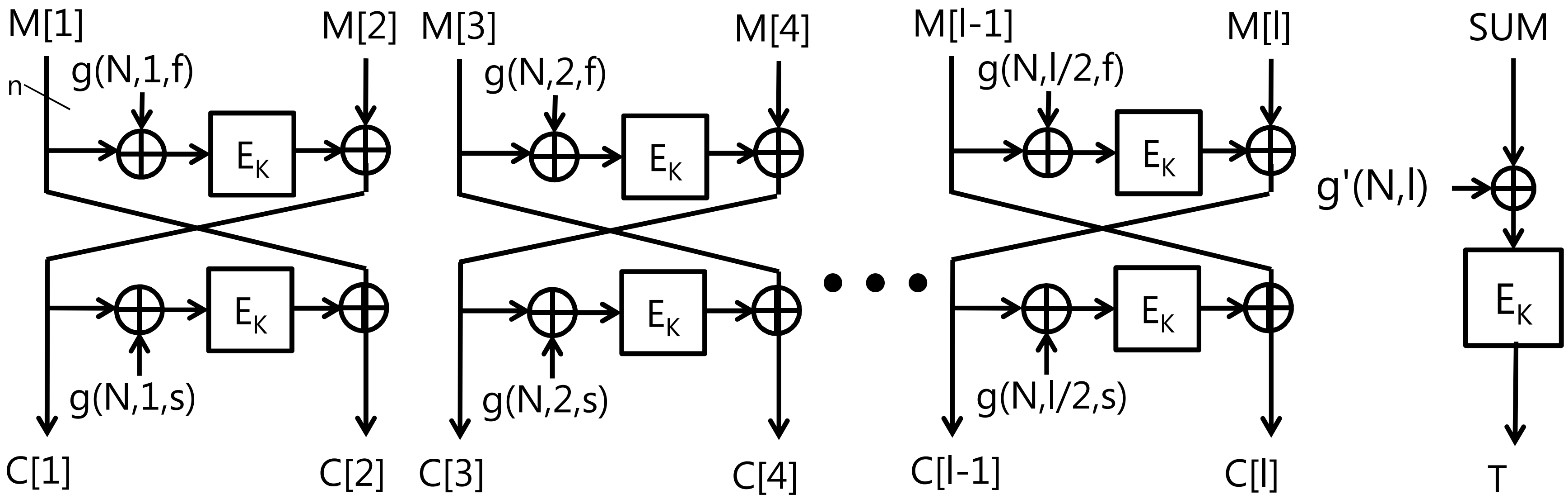
Using 2-round is enough

- When Y′[1] = Y[1] and Y′[2] ≠ Y[2], Z′ is always different from Z and X′[2] is independent and random



OTR : Offset Two-Round (simplified)

- The result : OTR mode presented at Eurocrypt 2014
- (Roughly) Encryption = 2-round LRC,
- MAC = Encryption of plaintext checksum, which is XORs of *even* plaintext block



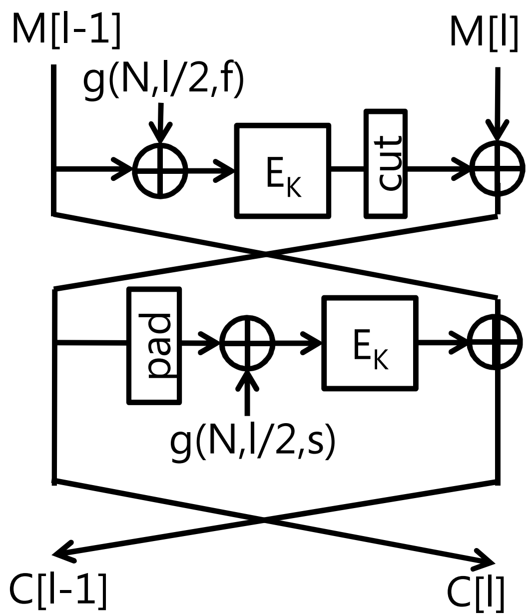
SUM
$= M[2] \oplus M[4] \oplus \dots \oplus M[l]$

Mask function
$g(N,i,j) = E_K(N) \times 2^i$ (xor L if j = ``s'')

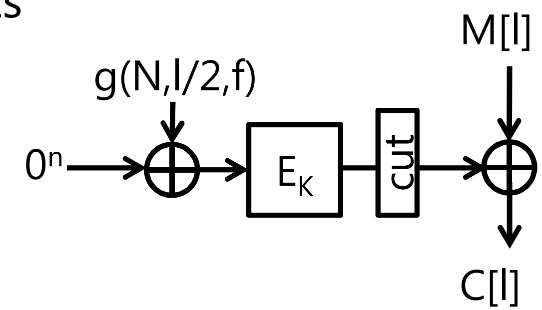
Additional points in design

- Need to handle partial-length messages
  - Padding to 2n bits is no good (expansion!)
- OTR avoids unnecessary ciphertext expansion, with dedicated functions for the last chunk

Last chunk = n+1~2n bits



Last chunk = 1~n bits



Security of OTR

- A brief description of nonce-based AE security notions :
- Privacy : the hardness of distinguishing (C,T) from random sequence, using enc queries (N,M)
- Authenticity : the hardness of producing a forgery (N',C',T'), using enc and dec queries
  - Forgery = given multiple (N,M,C,T) obtained by enc queries, generate a new (N',C',T') which is valid
- The observations so far allow to prove $O(\sigma^2/2^n)$ advantages for both notions, for σ blocks in queries
  - Similar to OCB and many others

Summary of OTR

- Mostly keeping OCB's good properties
  - Rate-1
  - Parallelizable for Enc & Dec
  - On-line (under 2-block partition)
- And inverse-free, provably secure if BC is a PRP or PRF
- CAESAR submission as a mode of AES (AES-OTR)

Table 1. A comparison of AE modes. Calls denotes the number of calls for m-block message and a-block header and one-block nonce, without constants.

Mode	Calls	On-line	Parallel	Primitive
CCM [3]	$a+2m$	no	no	E
GCM [5]	m [E] and $a+m$ [Mul]	yes	yes	$E, \mathrm{Mul}^\dagger$
EAX [16]	$a+2m$	yes	no	E
OCB [32, 43, 46]	$a+m$	yes	yes	E, E^{-1}
CCFB [35]	$a+cm$ for some $1<c^\ddagger$	yes	no	E
OTR	$a+m$	yes¶	yes¶	E

† $\mathrm{GF}(2^n)$ multiplication
‡ Security degrades as c approaches 1
¶ two-block partition

Comparison of AE modes

OTR implementations w/ AES

- Basic Expectation
  - Almost the same speed as OCB = almost the same speed as enc-only mode
  - with smaller size (sw memory / hw gates)
  - Dec is as fast as Enc
- Suitable to heterogeneous environment

OTR implementations with AES

- On Intel CPU w/ AESNI
  - Bogdanov et al. [BLT14] (Haswell Core i5)
    - Less than 1 cycles/byte (cpb)
    - difference from OCB3 is ~0.15 cpb
  - We obtained similar figures with our own codes (0.88 cpb at Haswell Core i7)

OTR implementations with AES

- On 8-bit Atmel AVR (ATmega 128)
  - Assembly AES from open source (AVRAES), runs at 156 cpb for enc, 196 cpb for dec
  - Mode is written in **assembly**
  - ~240 cpb for 256 input bytes, for both Enc/Dec
  - ~2100 ROM bytes, ~180 RAM bytes
- For reference, OCB on Atmega 128 [IMGM14]
  - AVRAES + mode written in **C**
  - 315 cpb for Enc, 354 for Dec (~256 input bytes)
  - ~5000 ROM, ~970 RAM bytes

OTR implementations with AES

- Hardware : working on FPGA
- Third-party implementation for any platform is always welcome!

Possible Further Applications

- OTR was a quite successful application, but there may be some other application areas ;
- Large-block cipher mode ?
  - CMC and EME (Rate-2, using inverse)
  - Recent AEZ v3 (a CAESAR candidate) by Hoang et al. did the work for EME, results in a rate-2.5 scheme

Possible Further Applications

- OTR was a quite successful application, but there may be some other application areas ;
- Large-block cipher mode ?
  - CMC and EME (Rate-2, using inverse)
  - Recent AEZ v3 (a CAESAR candidate) by Hoang et al. did the work for EME, results in a rate-2.5 scheme
- On-line (authenticated) encryption ?
  - TC1/2/3 by Rogaway and Zhang
  - CAESAR submissions (COPA, ELmD, POET)
    - COBRA : inverse-free but turned out to be wrong (withdrawn due to the attack by Nandi)
- Questions :
  - Achievable rate
  - Appropriate security notions (for 2n-bit block ?)
    - Answers can depend on the target functionality

Direct tweaking and Decomposition

Motivation

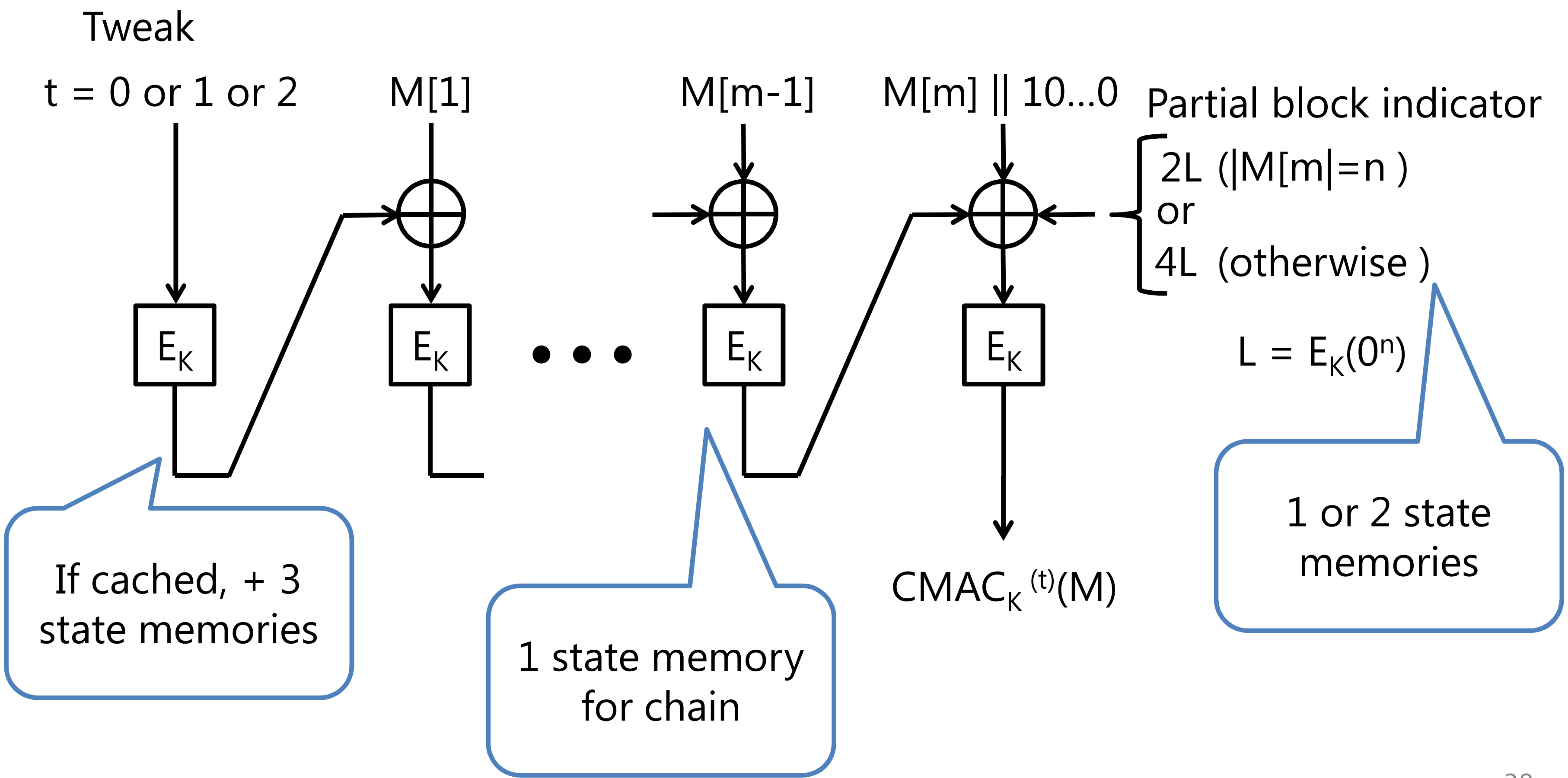
- Modes generally need its own memories outside BC we use
  - OCB/OTR's mask, CBC-MAC chain value, etc.
- How we can reduce these memories?
  - Not by implementation, not by changing the blockcipher – mode refinements
  - Possibly keeping the efficiency
- Beneficial to constrained devices
  - Often comes with several side effects (reduced pre-computation etc.)

A bad example

- EAX [Bellare-Rogaway-Wagner] : a rate-2 AE mode
  - Enc-then-auth style
  - Provable security
- EAX-prime : ANSI standard for Smart Grid (C12.22)
  - Derived from EAX, but requires fewer state memories than EAX, which would be good for constrained devices
- Both use different variants of CMAC (tweaked CMAC)
- and the difference is significant in security

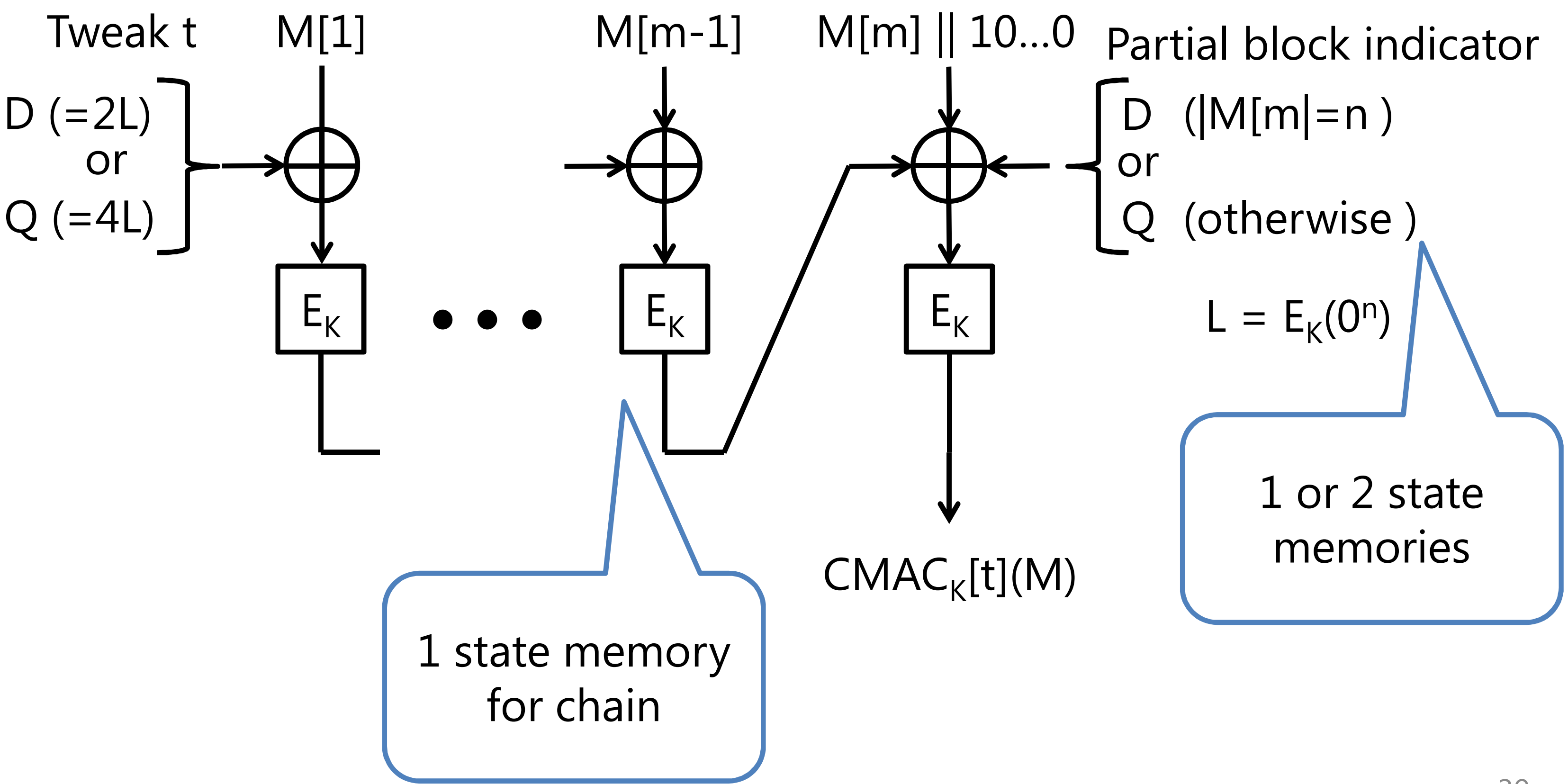
Tweaked CMAC in EAX

- 3 variants with $\mathrm{CMAC}^{(\mathrm{tweak})} = \mathrm{CMAC}(\mathrm{tweak} \,||\, X)$, tweak = 0,1,2 (in n bits)
  - E_K(tweak) can be cached as initial mask
  - 4 ~ 6 state memory blocks



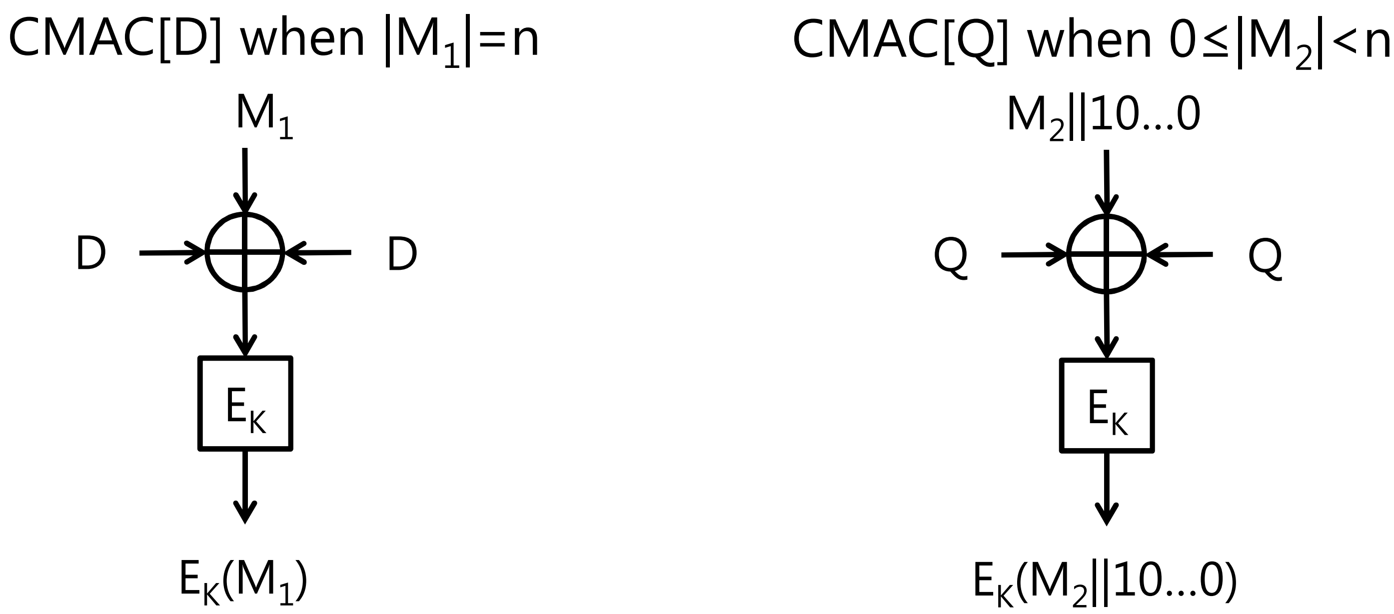
Tweaked CMAC in EAX-Prime

- 2 variants with CMAC[D] and CMAC[Q]
  (tweak = D, Q)
- Initial mask set = last mask set ({D,Q})
- Reduced state memories : 2 ~ 3 blocks



Insecure Separation

- CMAC[D] and CMAC[Q] fail to provide (independent) PRFs
- In case $|M| \le n$;

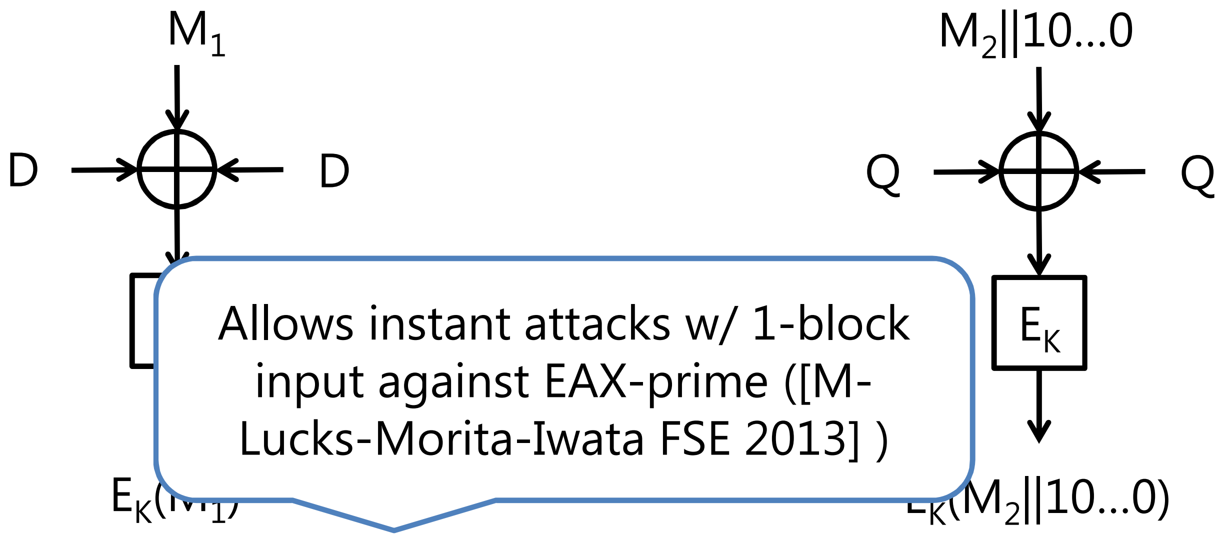


Making $M_1 = M_2||10\ldots0$ yields the same outputs -> unlikely for two independent PRFs

Insecure Separation

- CMAC[D] and CMAC[Q] fail to provide (independent) PRFs
- In case $|M| \leq n$;

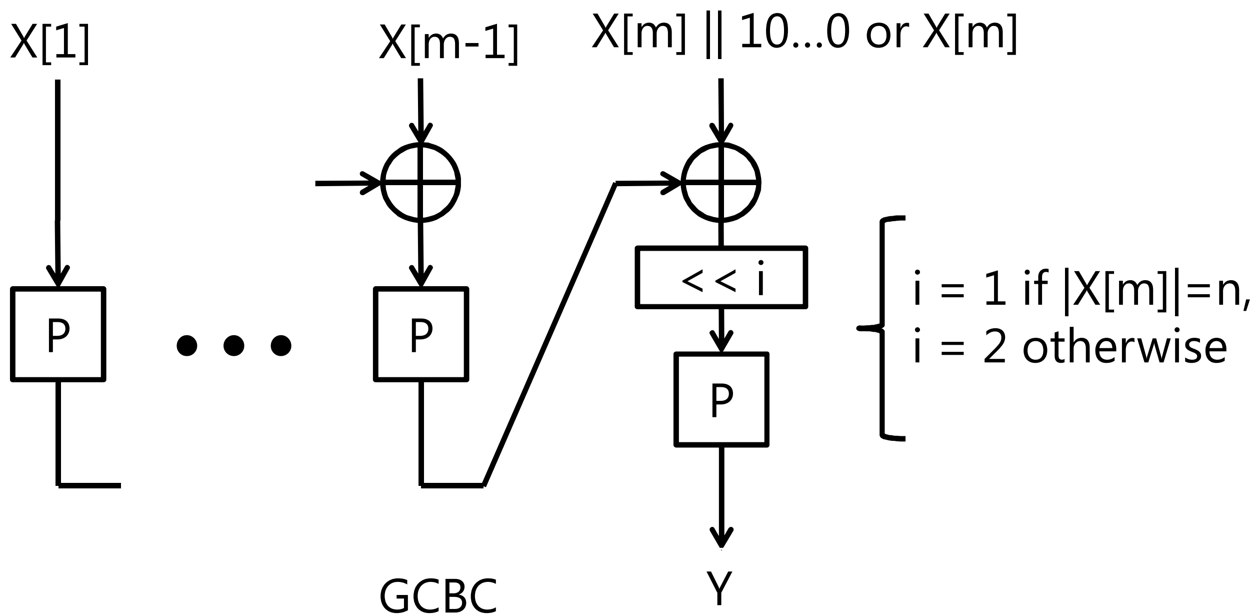
CMAC[D] when $|M_1|=n$ CMAC[Q] when $0 \leq |M_2| < n$



Making $M_1 = M_2||10\ldots0$ yields the same outputs -> unlikely for two independent PRFs

A good example

- How to avoid 2L / 4L masking in CMAC, w/o another BC call ?
- GCBC [Nandi] did the job
- Instead of masking, GCBC introduces in-state modification,
- which we call tweak function or *direct tweaking*

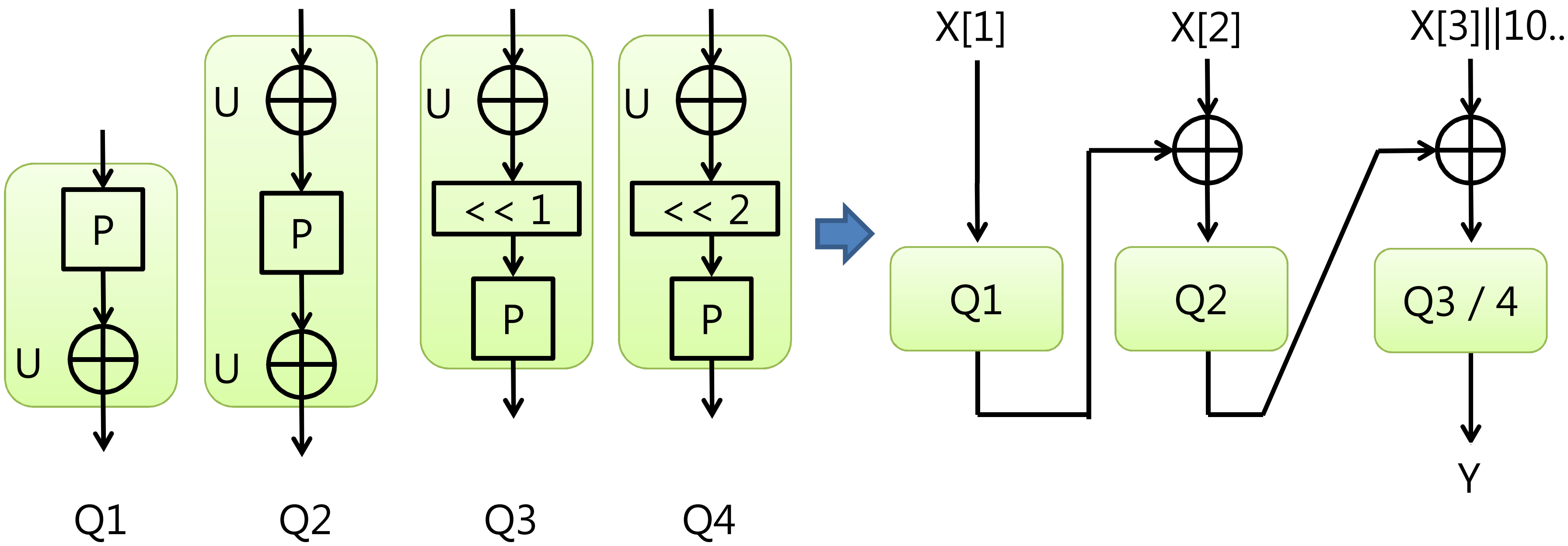


GCBC

(slightly different from the original, and for 1-block message the operation is different)

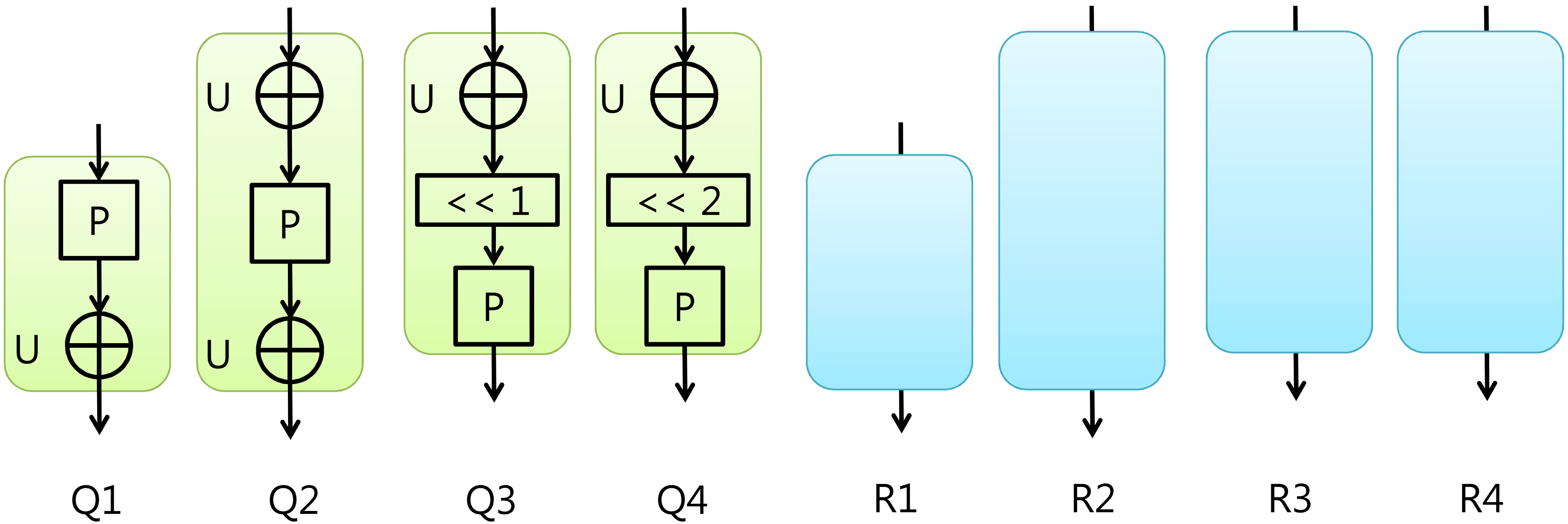
Security of GCBC

- How we prove security of GCBC?
- Use decomposition via dummy mask
  - Initially employed by Iwata-Kurosawa for proof of CMAC
- We define 4 n-bit functions using a random dummy mask U
- GCBC can be simulated by these 4 functions
- GCBC is easily analyzed if 4 functions were independent PRFs



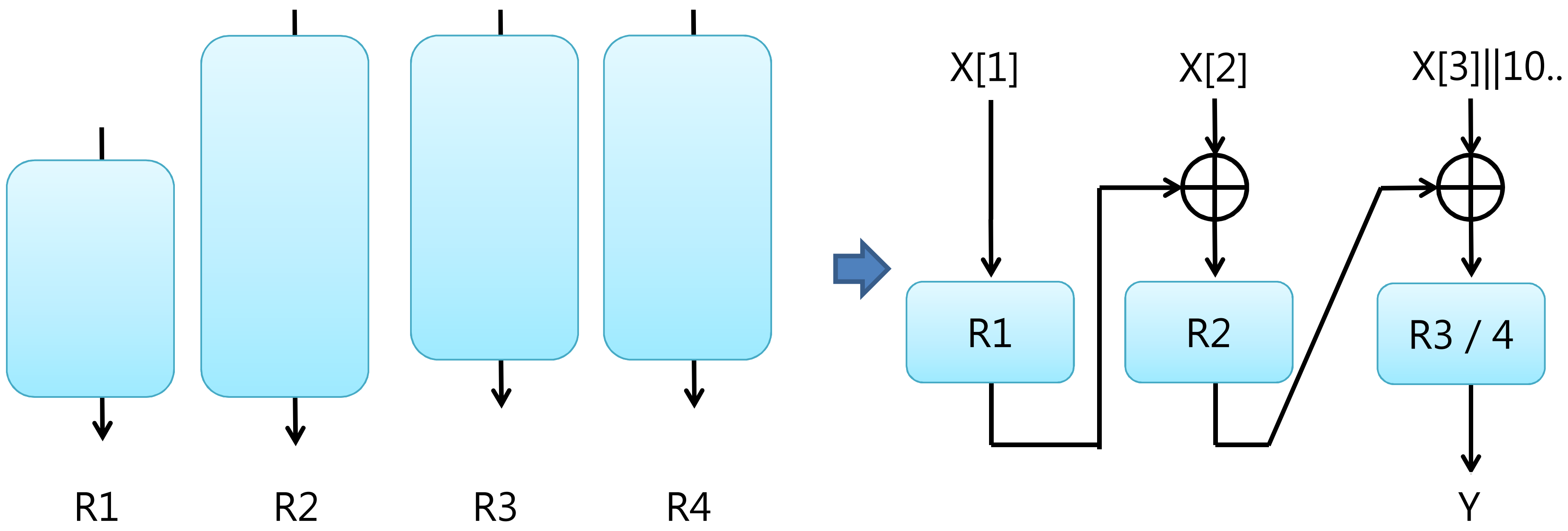
GCBC analysis

- We prove 4 functions are (comp-independent) PRFs
- Step 1. find *input differential probability constraints*
  - e.g. max_c Pr[U xor (U<<1)=c] for Q2 and Q3
  - ${}_4C_2$ = 6 constraints
- Step 2. prove all constraints have a small upper bound
  - secure from the theory of tweakable blockcipher [Liskov-Rivest-Wagner]



GCBC analysis (Contd.)

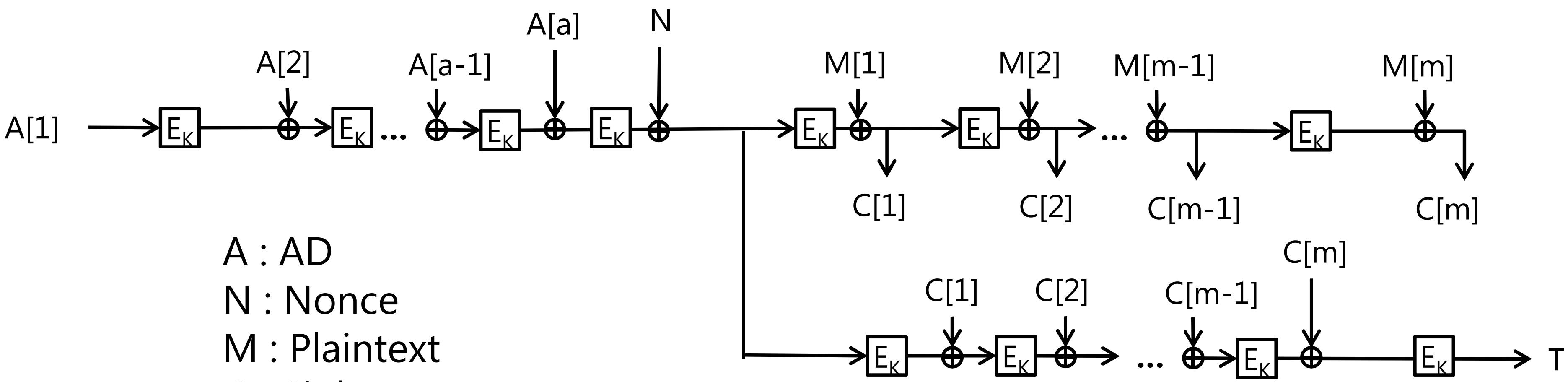
- Step 3. Proving CBC-MAC-like function using 4 PRFs



The case of Authenticated Encryption

Initial design

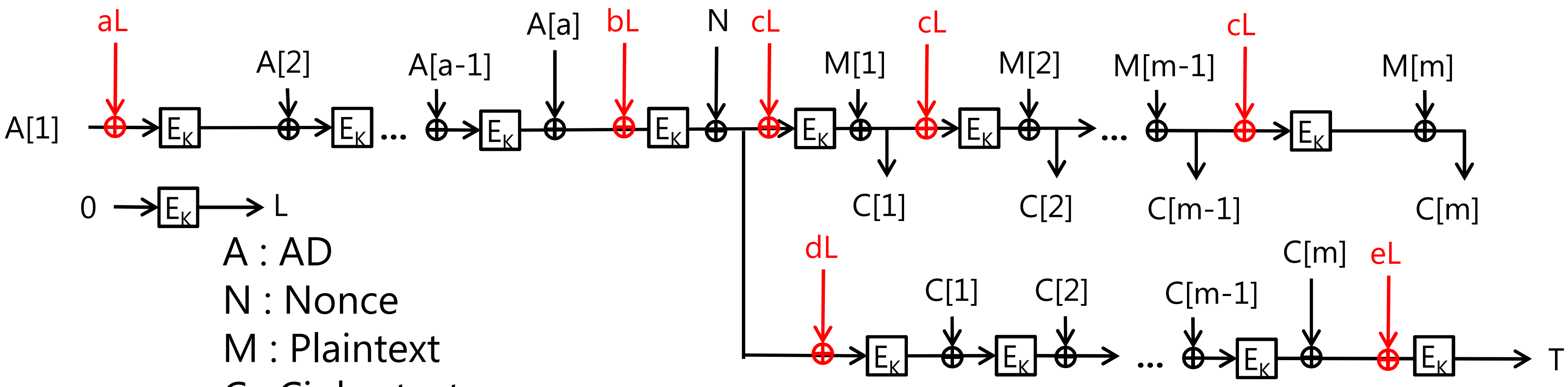
- We start with a generic composition
  - Enc-then-MAC
  - MAC = CBC-MAC-like
  - Enc = CTR or OFB or CFB : We chose CFB for its small memory
  - One-key : insecure at this stage



A : AD
N : Nonce
M : Plaintext
C : Ciphertext
T : Tag

Initial design

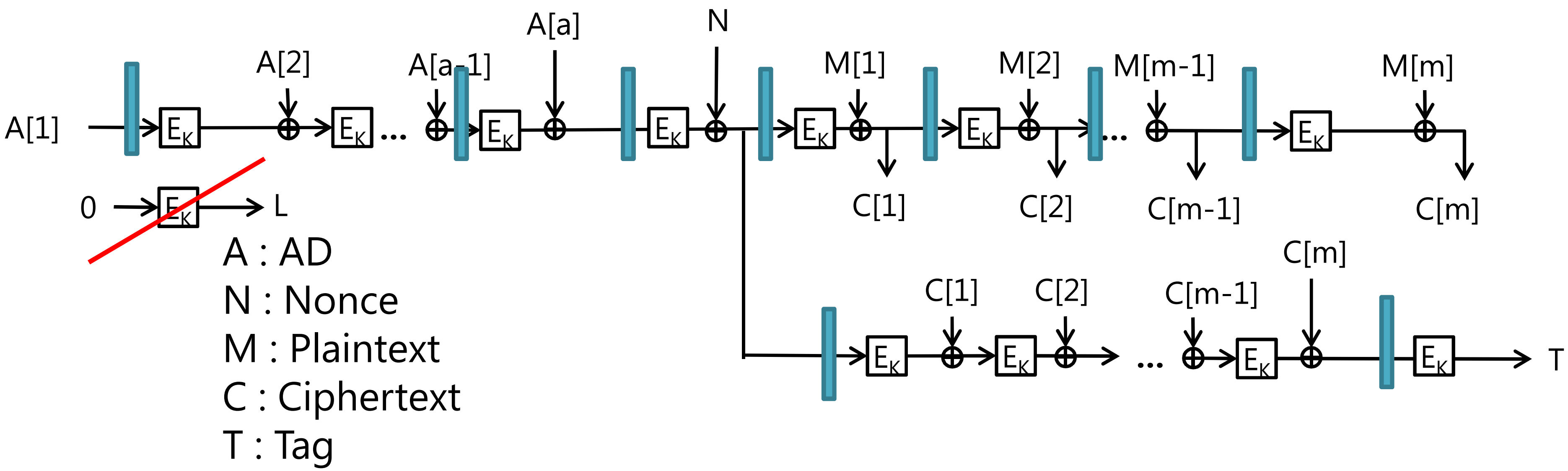
- CCM, EAX, and EAX-prime use input masking based on E(const)
- While we want our AE to work without masking
  - Small memory and fast for short input w/o precomputation (or, key-agility)
  - Suitable to constrained devices, short-packet communication



A : AD
N : Nonce
M : Plaintext
C : Ciphertext
T : Tag

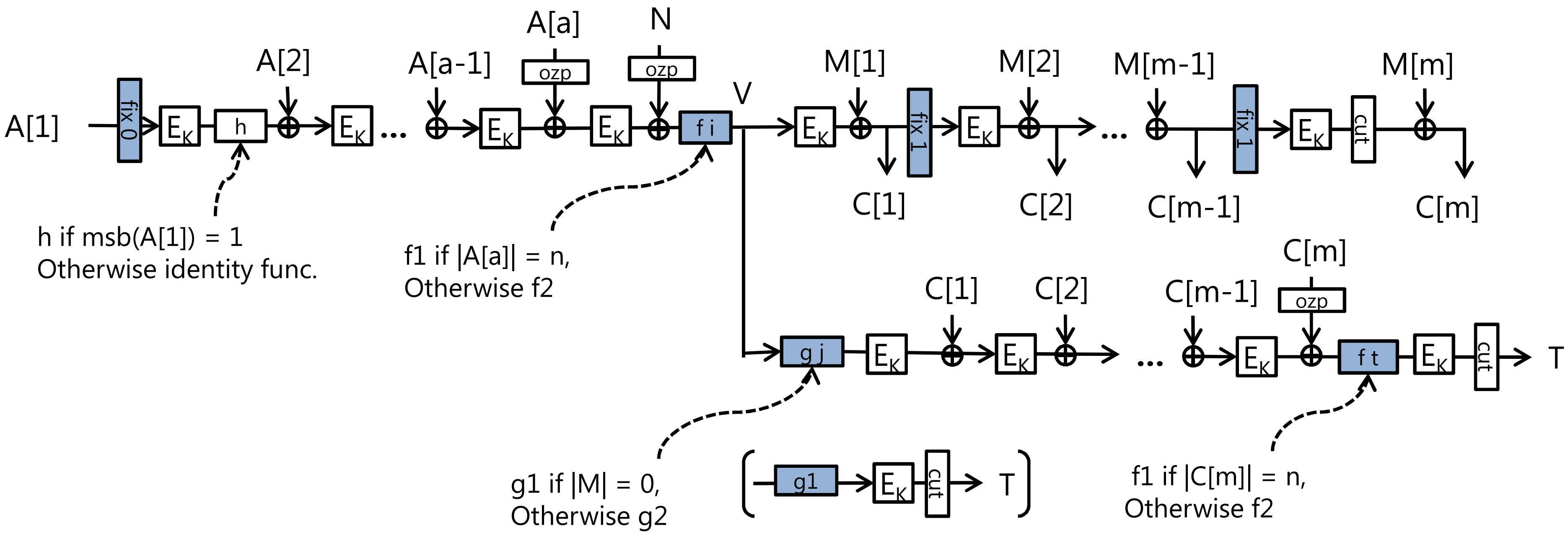
Initial design

- We want to make it secure with tweak functions
- How should we modify plain CBC-MAC + CFB?
- How many tweak functions needed, where to insert?



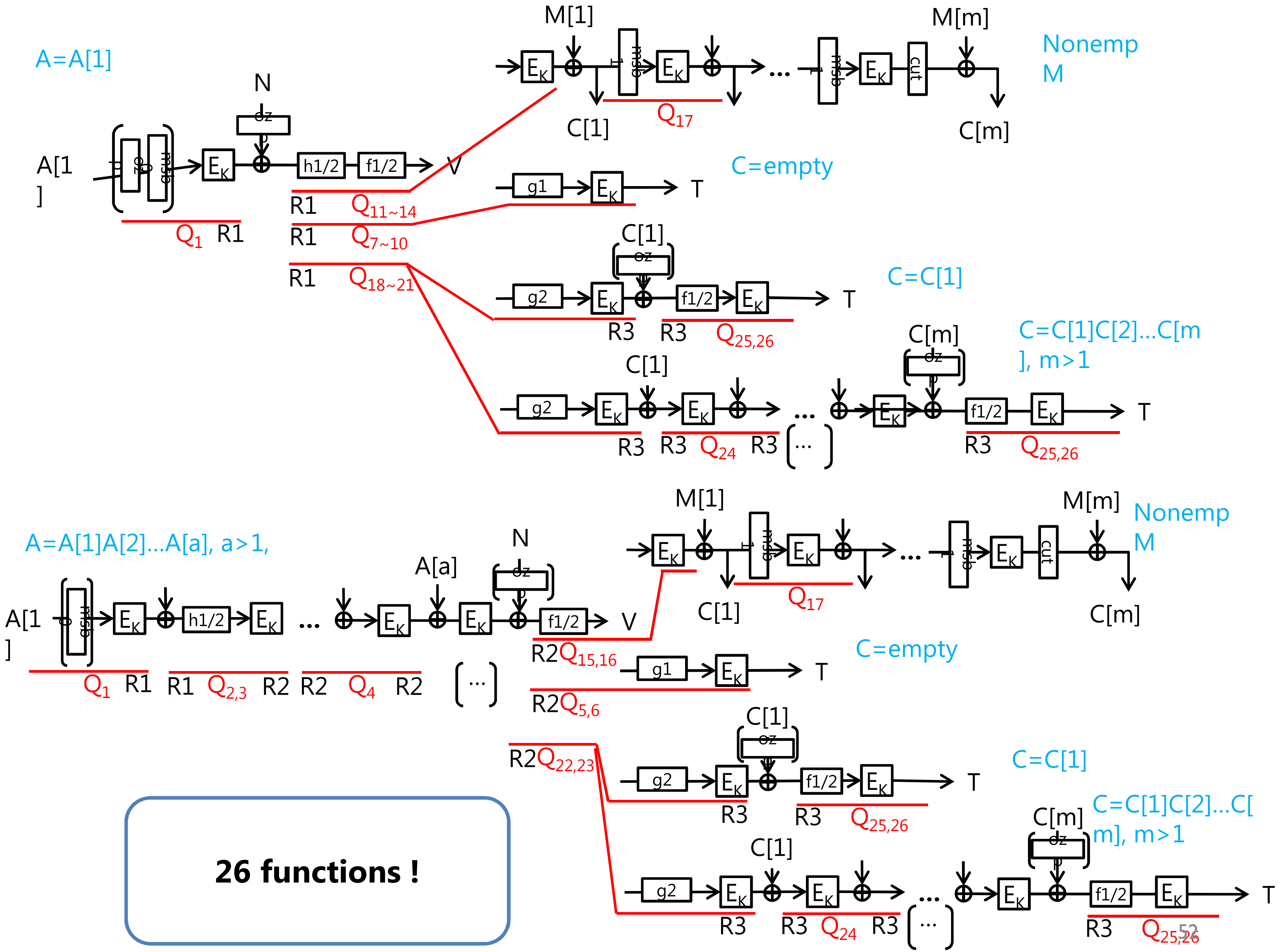
Concrete design = CLOC

- Investigated a large number of possibilities
- We found a solution using 5 tweak functions + 2 msb-fixing functions
  – h, f1, f2, g1, g2, and fix0, fix1
- The result is CLOC (presented at FSE 2014 and submitted to CAESAR) [Iwata-M-Guo-Morioka]



Decomposition of CLOC

- How we prove the security of CLOC?
- Decomposition needs to consider various cases on the lengths of Nonce, AD, and plaintext/ciphertext
- The analysis is considerably more complex than the case of MAC, as follows



26 functions !

Conditions for the tweak functions

- If these 26 functions were independent, proving security is not difficult
- We have 26 functions -> ${}_{26}C_2$ =325 differential provability constraints to make CLOC secure !
- Removing equivalent ones, there remains 55 constraints
- Ideally all should be satisfied w/ prob = $1/2^n$
- How we make ?

$i \oplus f_1$	$i \oplus f_2h$	$f_1 \oplus f_2h$	$h \oplus g_2f_1$	$g_2f_1 \oplus g_1f_2h$
$i \oplus g_1f_1$	$i \oplus h$	$f_2 \oplus g_1f_1$	$h \oplus f_2$	$g_2f_1 \oplus f_2h$
$i \oplus g_1f_1h$	$i \oplus g_1$	$f_2 \oplus g_1f_1h$	$h \oplus g_1$[illegible]	$g_1f_2 \oplus g_2f_1$
$i \oplus g_2f_1$	$i \oplus g_2$	$f_2 \oplus g_2f_1$	$h \oplus g_2f_2$	[illegible] $\oplus g_2f_1h$
$i \oplus g_2f_1h$	$f_1 \oplus g_1f_1h$	$f_2 \oplus g_2f_1h$	$g_1f_1 \oplus f_1h$	$g_1f_2 \oplus$ [illegible]
$i \oplus f_1h$	$f_1 \oplus g_2f_1h$	$f_2 \oplus f_1h$	$g_1f_1 \oplus g_2f_1h$	$g_1f_2 \oplus g_2f_2h$
$i \oplus f_2$	$f_1 \oplus f_2$	$f_2 \oplus g_1f_2h$	$g_1f_1 \oplus g_2f_2$	$g_1f_2 \oplus f_2h$
$i \oplus g_1f_2$	$f_1 \oplus g_1f_2$	$f_2 \oplus g_2f_2h$	$g_1f_1 \oplus g_2f_2h$	$g_2f_2 \oplus g_1f_1h$
$i \oplus g_1f_2h$	$f_1 \oplus g_1f_2h$	$g_1 \oplus g_2$	$g_1f_1 \oplus f_2h$	$g_2f_2 \oplus f_1h$
$i \oplus g_2f_2$	$f_1 \oplus g_2f_2$	$h \oplus f_1$	$g_2f_1 \oplus g_1f_1h$	$g_2f_2 \oplus g_1f_2h$
$i \oplus g_2f_2h$	$f_1 \oplus g_2f_2h$	$h \oplus g_1f_1$	$g_2f_1 \oplus f_1h$	$g_2f_2 \oplus f_2h$

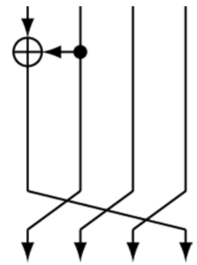
Fig. 9. Differential probability constraints of f_1, f_2, g_1, g_2, and h

e.g. max_c Pr_U[f1(U) xor f2(h(U)) = c]

Building the tweak functions

- For efficiency reason we require the tweak functions to be
  - computed by word permutation and XOR, with 4 words
  - -> each function is a 4x4 matrix over GF(2^n/4)
  - -> differential pr = $1/2^n$ iff corresponding sum of matrices is full rank (4)
- Define a generator 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])
  - Assign M^i to a tweak function
  - $M^{15}=M^0$ = identity so we have 14^5 space for search
  - Each M^i (except i=5 and 10) can be implemented using at most 4 word XORs and a block permutation

Search

- We associate $(i_1, i_2, i_3, i_4, i_5) \in \{1, \ldots, 14\}^5$ 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 $(i_1, i_2, i_3, i_4, i_5) \in \{1, \ldots, 14\}^5$ with 55 constraints, using computer
  - matrix rank computations
- 864 combinations proved to be secure
- Define a cost function to choose the best combination (# of XORs etc.)
  - The chosen one is $(i_1, i_2, i_3, i_4, i_5) = (8, 1, 2, 1, 4)$
  - This specifies CLOC

Performance of CLOC-AES

- Primary focus : embedded software
- Atmel AVR ATmega128
  - 8-bit microprocessor
  - Using AVRAES
    - 156.7 cpb for encryption, 196.8 cpb for decryption
  - Compare CLOC with EAX and OCB3
    - All modes are written in C
    - OCB3 is taken from OCB website, w/ some modifications for optimized performance on AVR

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
- In CLOC, the RAM usage is low and Init is fast, and it is fast for short input data, up to around 128 bytes

Conclusions

- Two design ideas to make blockcipher modes efficient
- Inverse-removal : removing BC inverse w/o increasing BC calls
  - substituting BC/BC^{-1} with 2-round Feistel
  - Result is OTR : inverse-free, rate-1, parallel AE
- Direct tweaking : reducing the memory amount, removing precomputation
  - Result is CLOC : a low-overhead AE, fast for short input
  - CLOC focuses on (embedded) software
  - We also designed SILC as a variant of CLOC for (constraind) hardware
- Would be applicable to other application areas …

Thank you !!