OCB : Parallelizable Authenticated Encryption

PMAC : Parallelizable Message Authentication Code

Phillip Rogaway

UC Davis (USA) and
CMU (Thailand)

`rogaway@cs.ucdavis.edu`
`www.cs.ucdavis.edu`

with assistance from Mihir Bellare (UCSD)
and John Black (UNR)

NIST Modes of Operation Workshop - 20 October 2000

What I'm doing

OCB - Refining a parallelizable scheme recently suggested by [Jutla] for authenticated encryption (privacy+authenticity)

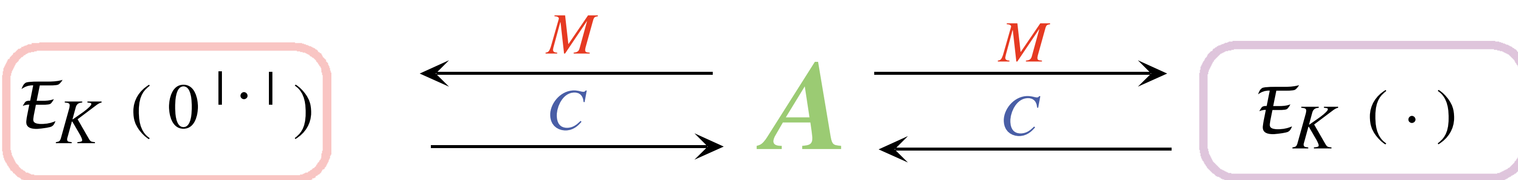
PMAC - Improving on [Bellare, Guerin, Rogaway], [Bernstein], [Gligor, Donescu] for a parallelizable MAC.

OCB (Offset CodeBook) Mode

Security Goals

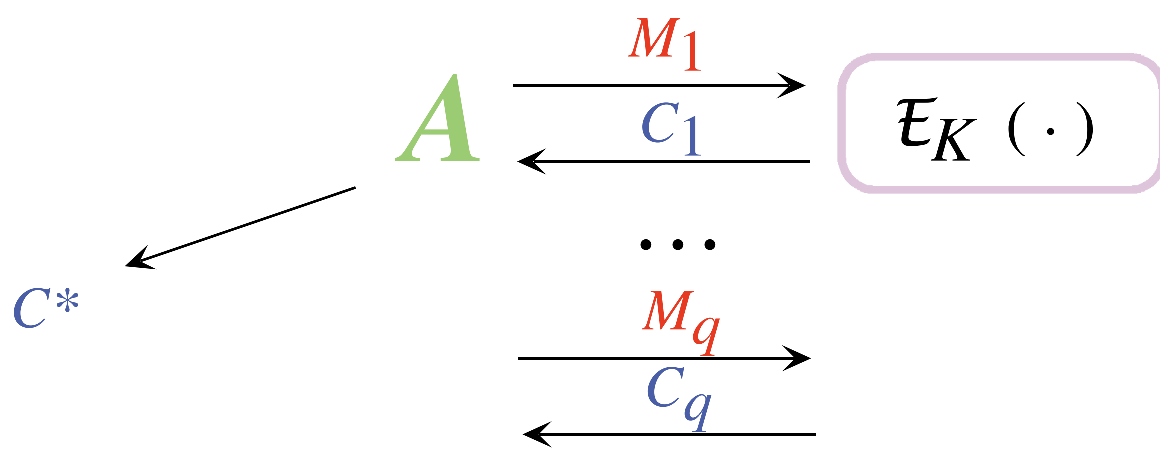
(1) The adversary can't understand anything about plaintexts

Formalized as *IND - CPA* [GM, BDJR]



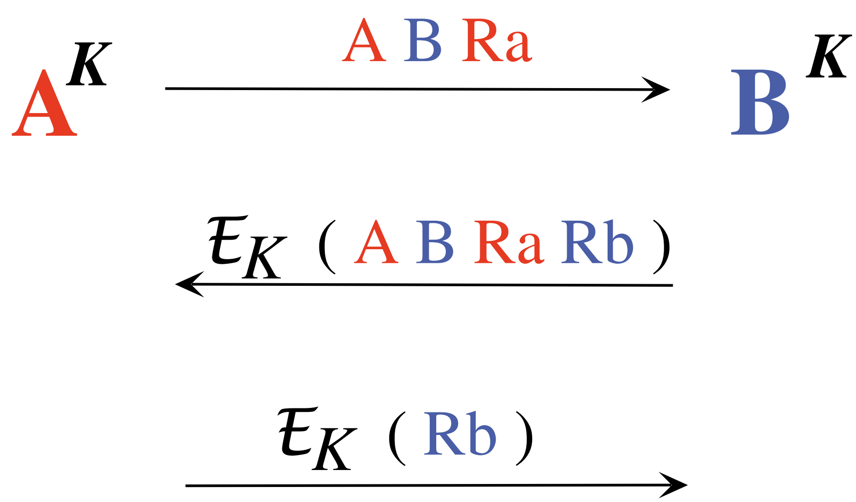
(2) The adversary can't produce valid ciphertexts

Formalized as *Integrity of Ciphertexts* [KY, BR, BN]



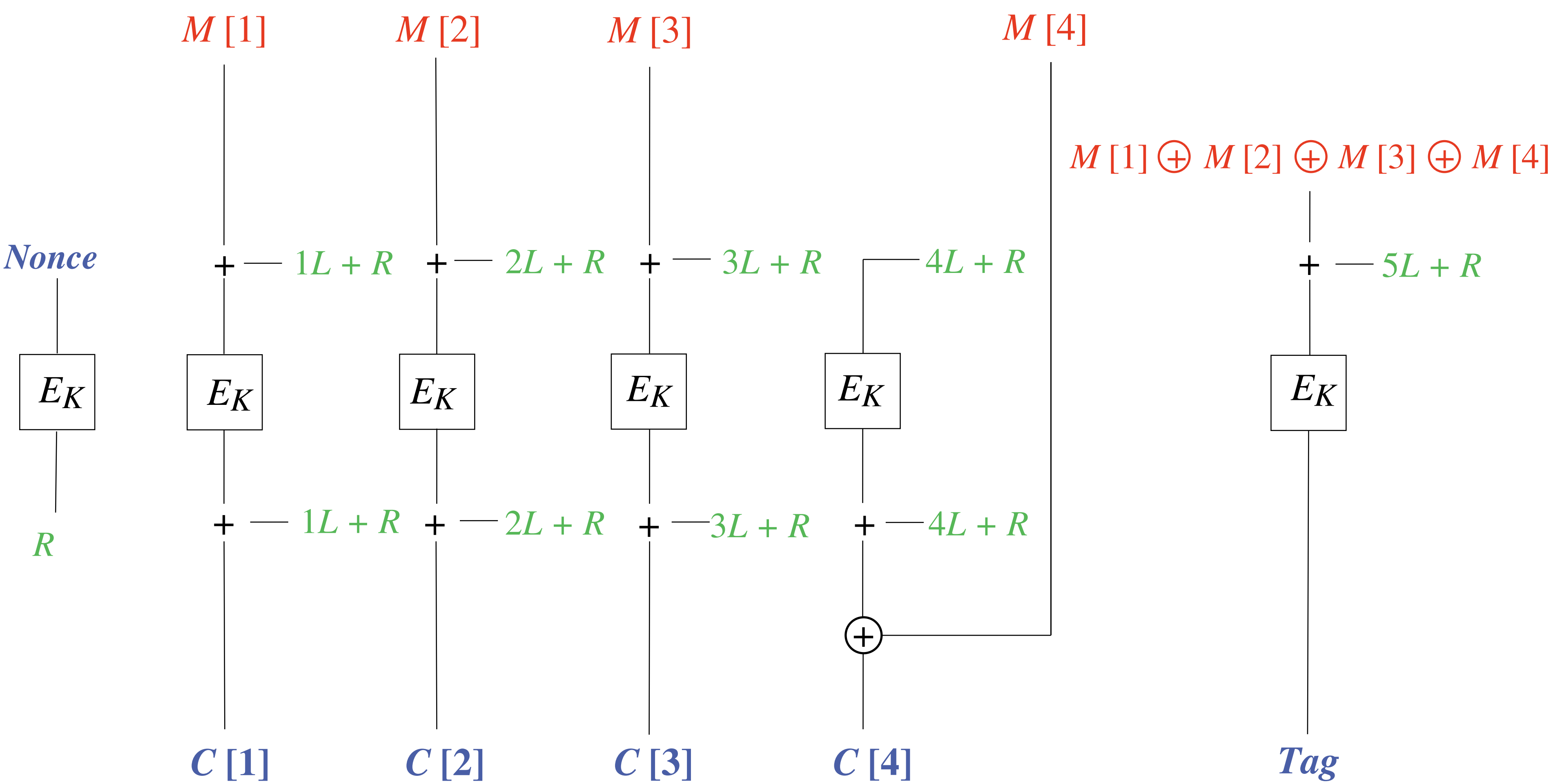
Why is Integrity-of-Ciphertexts important?

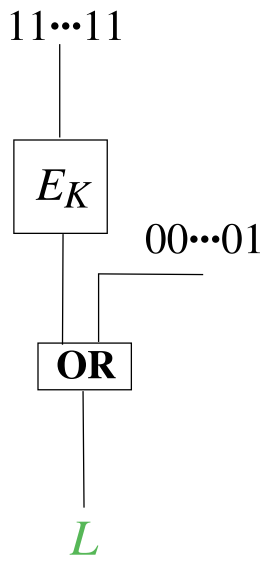
Because users of encryption **often** assume, wrongly, that they have it! Achieving IND-CPA + integrity-of-ciphertexts implies IND-CCA [BN] and non-malleablity-CCA, so an encryption scheme with Integrity-of-Ciphertexts is **far less likely** to be misused.



This sort of encryption-scheme usage, to **bind** together a private message, is very common in the literature and in practice. But is **completely bogus** when using IND-CPA encryption.

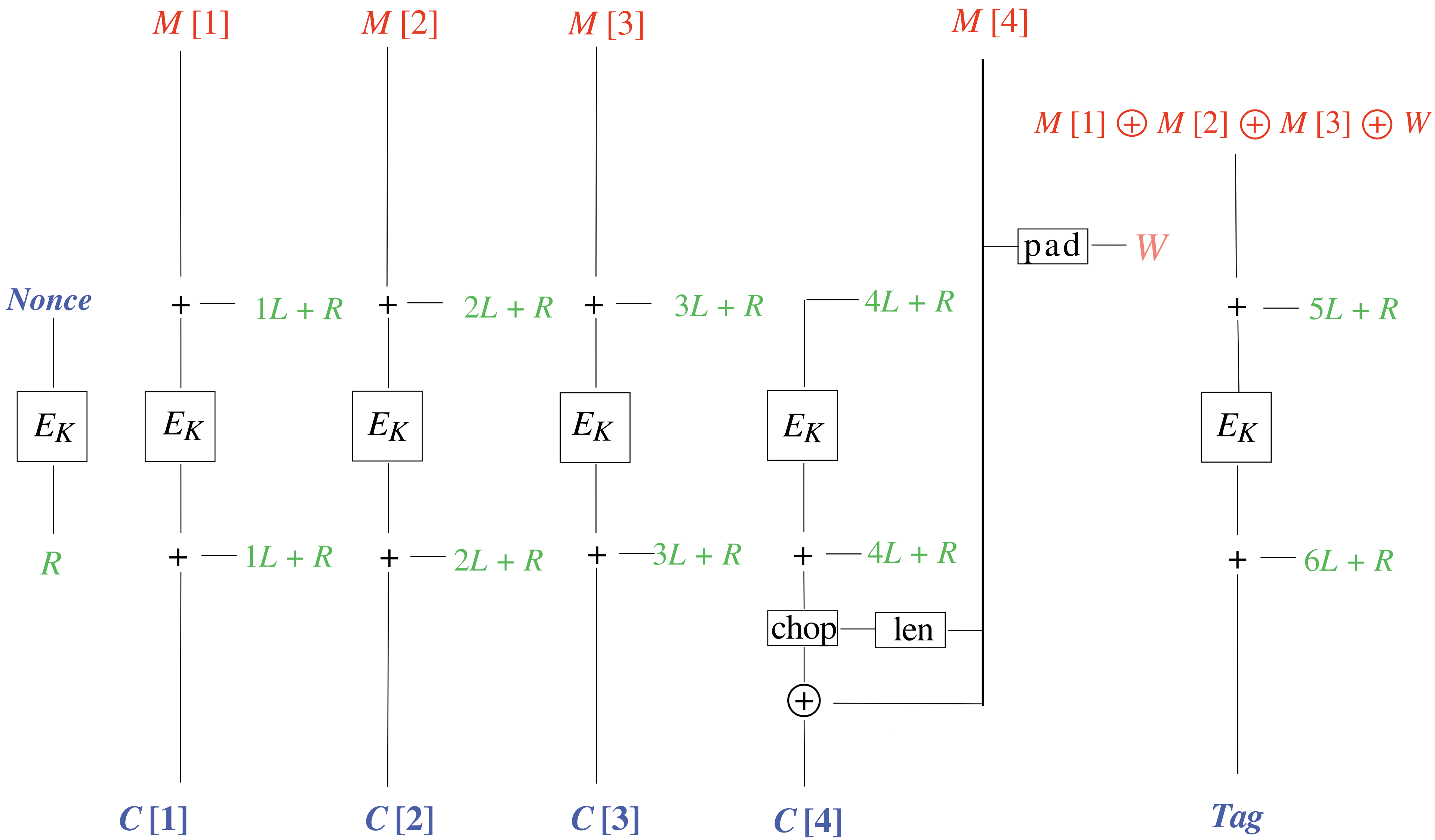
OCB (full final block)





11•••11
E_K
00•••01
OR
L

OCB (short final block)



procedure Encrypt (, $Nonce$, M)

$L = E_K(1^{128}) \vee 0^{127}1$ *// Do during key-setup*
$R = E_K(Nonce)$

Let $m = \max\{1, \lceil |M|/128 \rceil\}$
Let $M[1], \ldots, M[m]$ be strings s.t. $M[1] \cdot \cdot M[m] = M$ and $|M[i]| = 128$ for $1 \leq i < m$

$\text{Offset} = L + R$
for $i = 1$ to $m - 1$ **do**
  $C[i] = E_K(M[i] + \text{Offset}) + \text{Offset}$
  $\text{Offset} = \text{Offset} + L$

if $|M[m]| = 128$ **then** $\text{Mask} = E_K(\text{Offset}) + \text{Offset}$
  $C[m] = M[m] \oplus \text{Mask}$
  $\text{Offset} = \text{Offset} + L$
  $\text{PreTag} = M[1] \oplus \cdots \oplus M[m-1] \oplus M[m] + \text{Offset}$
  $\text{Tag} = E_K(\text{PreTag})$
else $W = \text{pad}(M[m])$
  $\text{Mask} = E_K(\text{Offset}) + \text{Offset}$
  $C[m] = M[m] \oplus$ (last $|M[m]|$ bits of Mask)
  $\text{Offset} = \text{Offset} + L$
  $\text{PreTag} = M[1] \oplus \cdots \oplus M[m-1] \oplus W + \text{Offset}$
  $\text{Offset} = \text{Offset} + L$
  $\text{Tag} = E_K(\text{PreTag}) + \text{Offset}$

return ($Nonce$, $C[1] \cdot \cdot C[m]$, $T[1..tagLen]$)

OCB Advantages

(1) Fully parallelizable - important for HW and SW

(2) Arbitrary domain - any bitstring can be encrypted

(3) Short ciphertexts - $|M| + |Nonce| + |T|$

(4) Fewer block-cipher calls - ceiling{ $|M| / n$ } + 2

(5) Nonces - counter is fine - needn't be unpredictable

(6) Short key - OCB defined as using one AES key

(7) Fast key setup - one AES invocation to make L

(8) Addition version - three 128-bit adds per block
128-bit xor per block

(9) XOR version - four 128-bit xors per block,

one

OCB/xor

Gray codes and GF(2^{128})

Addition is less pleasant than you might think

- Add-with-carry unavailable from C
- Dependency among instructions slows things down

```
L1:     add ecx, edi
        adc edx, ebp
        adc edx, ebp
```

4.1 cycles

```
L1: xor ecx, edi
    xor edx, ebp
    xor eax, ebp
    dec eax
    jne L1
```

2.5 cycles

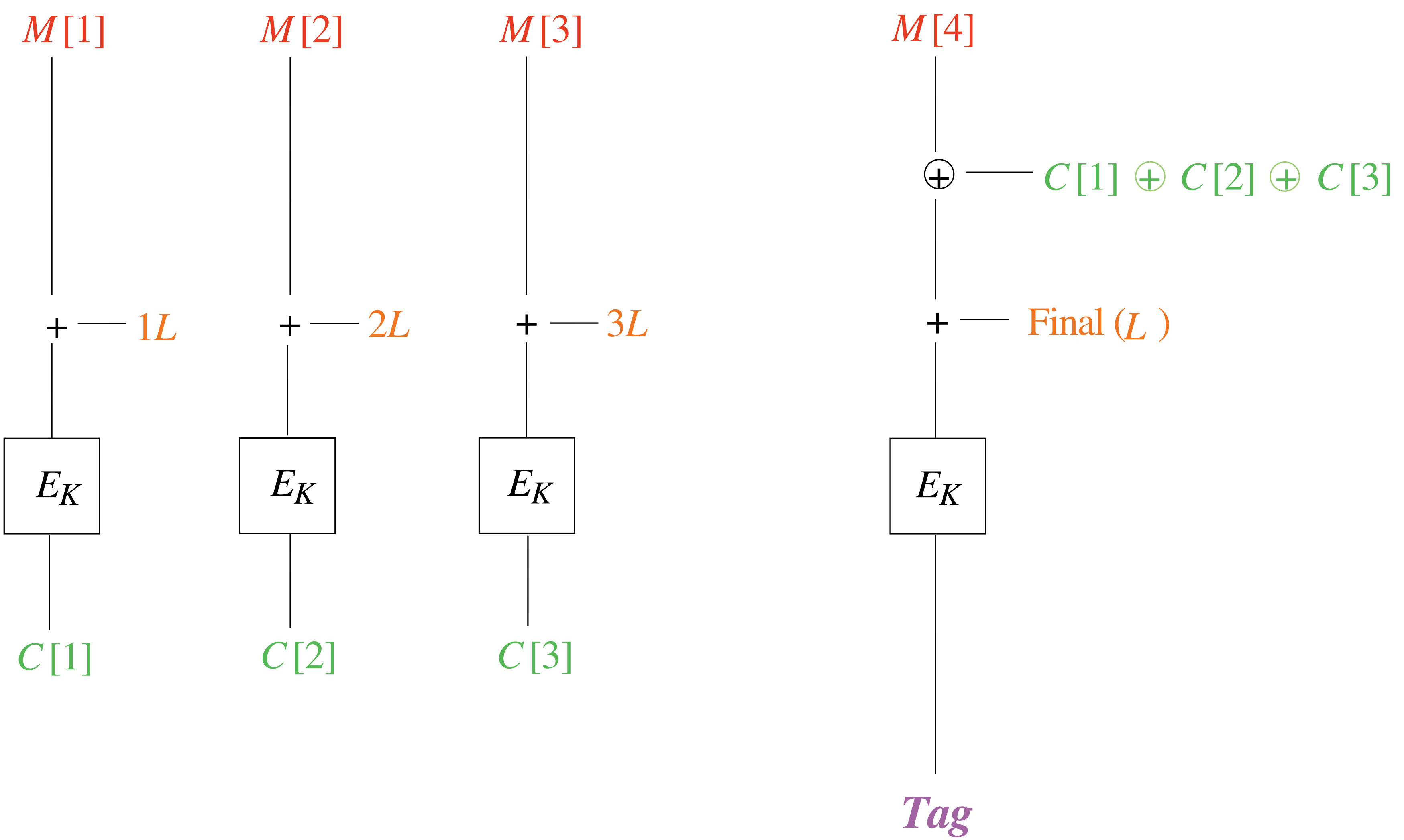
```
dec eax
jne L1
```

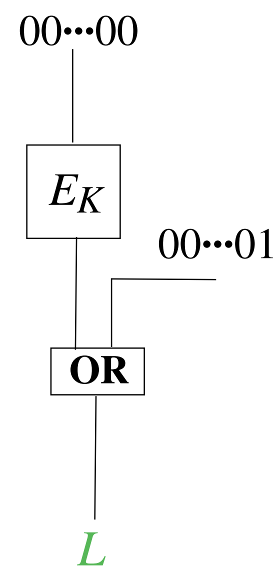
Offset(i+1) = Offset(i) `xor` L(`ntz`(i))

where $L(0) = L$ and

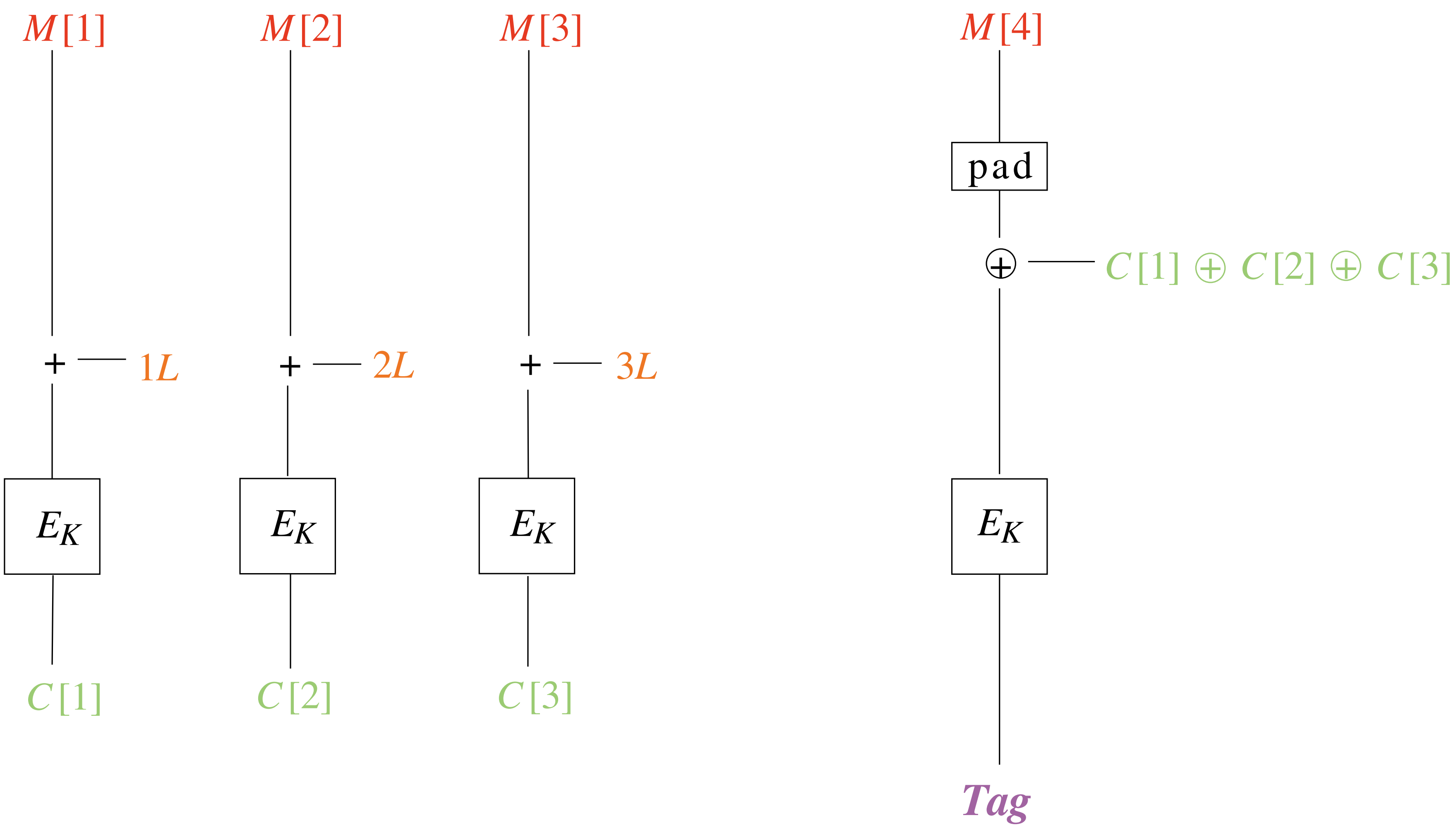
$$L(j+1) = \begin{cases} L(j) \ll 1 & \text{if } \texttt{lsb}(L(j)) = 0 \\ L(j) \ll 1 \ \texttt{xor CONST} & \text{otherwise} \end{cases}$$

PMAC (full final block)




00•••00
E_K
00•••01
OR
L

PMAC (short final block)



PMAC Advantages

(1) Fully parallelizable - important for HW and SW
(2) Arbitrary domain - any bitstring can be MACed
(3) Deterministic - uses no nonces or random values
(4) Short MACs - up to 128 bits, but 64 bits is enough
(5) Fewer block-cipher calls - ceiling{ | *M* | / *n* }
(6) Short key - PMAC defined as using one AES key
(7) Fast key setup - one AES invocation to make *L*
(8) Addition version - two 128-bit adds per block
per block
(9) XOR version - three 128-bit xors per block,

one 128-bit xor