ALE: AES-Based Lightweight Authenticated Encryption

Andrey Bogdanov[1], Florian Mendel[2], Francesco Regazzoni[3,4], Vincent Rijmen[5], Elmar Tischhauser[5]

[1]Technical University of Denmark
[2]IAIK, Graz University of Technology, Austria
[3]ALaRI - USI, Switzerland
[4]Delft University of Technology, Netherlands
[5]Dept. ESAT/COSIC, KU Leuven and iMinds, Belgium

Authenticated Encryption (AE)

- Is cryptography about encryption?
  - Yes, but not only!
  - Encryption alone is not enough in numerous applications
  - One might even argue that authentication is really what is needed in most cases

- Authenticated encryption

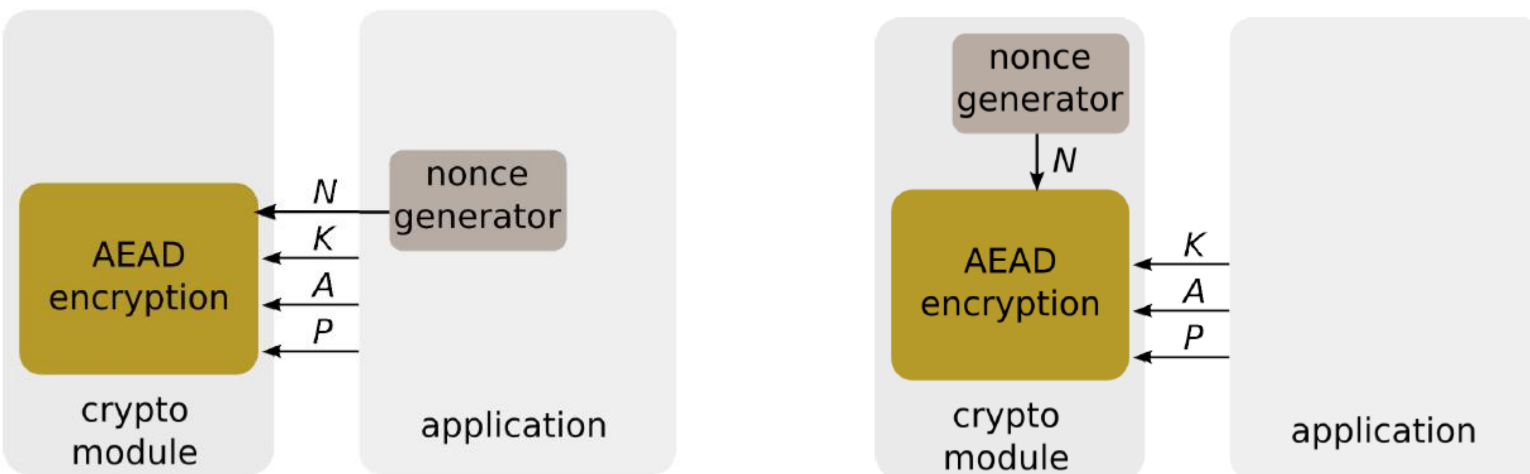
AE: (P,K) -> (C,T) with T authentication tag

- Authenticated encryption with associated data

AEAD: (A,P,K) -> (A,C,T) with A associated data transmitted in plaintext

The assumption of nonce

- Nonce N = number used once, freshness
- Nice but might be difficult to enforce in sometimes

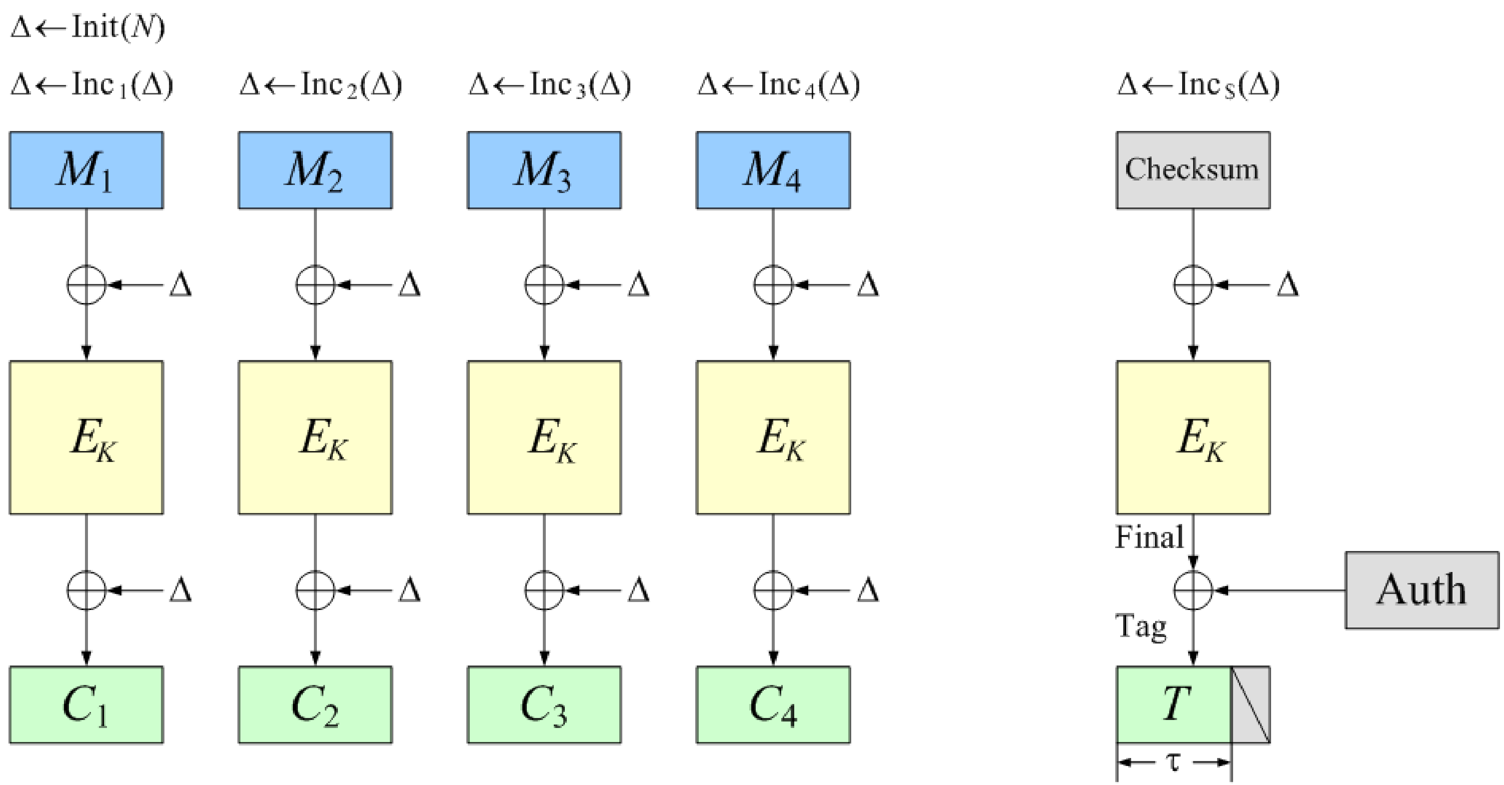


David McGrew, DIAC'12 slides

- Good news: Nonce can be "just" a counter!

Nonce-based: AES-OCB

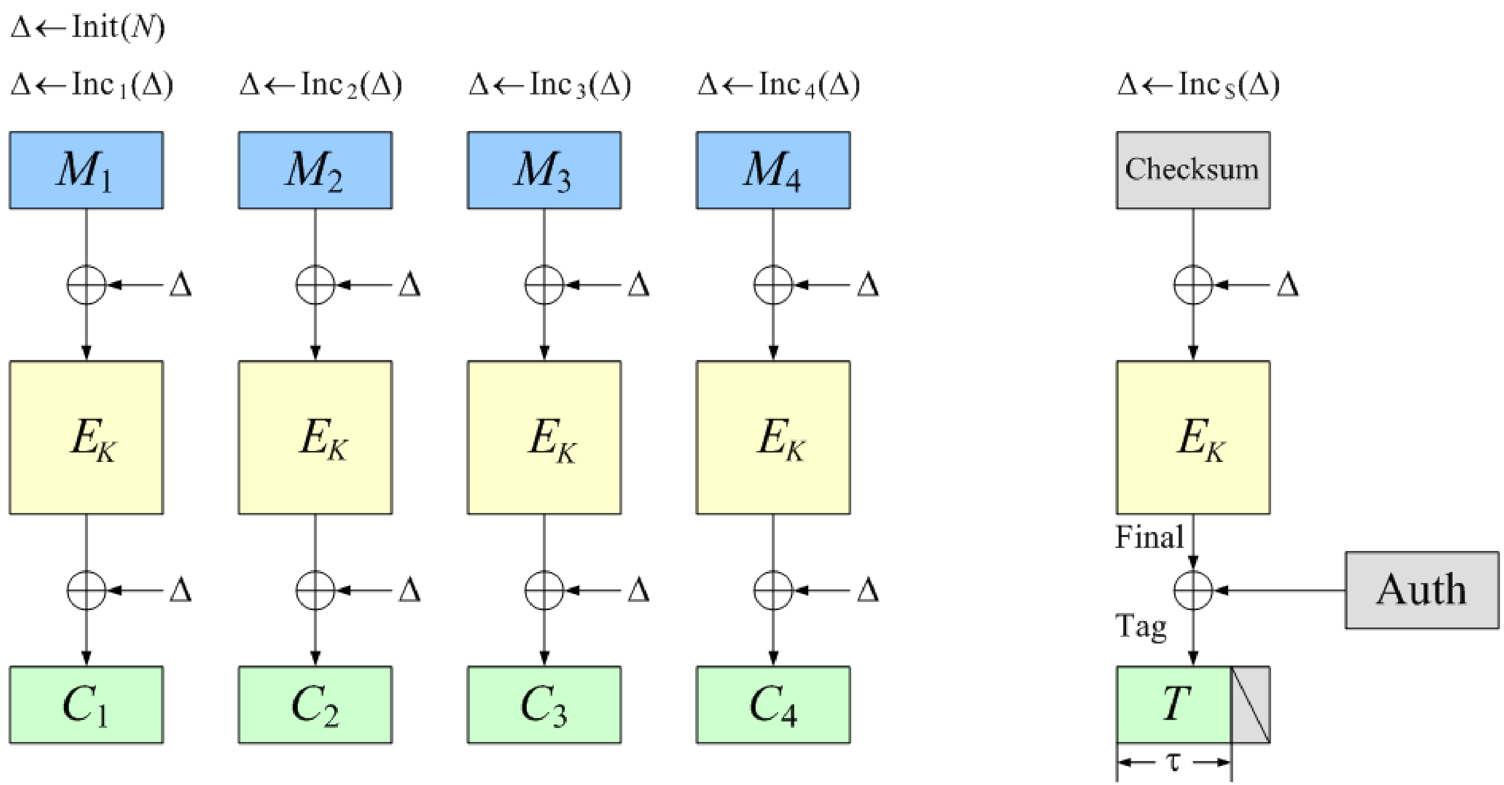
[RBBK01]
[BR02]
[R02]
[R04]
[KR11]



- Init(N): initialization function
- Inc: increment function
- Checksum = M1 xor M2 xor… Mn

Nonce-based: AES-OCB

[RBBK01]
[BR02]
[R02]
[R04]
[KR11]

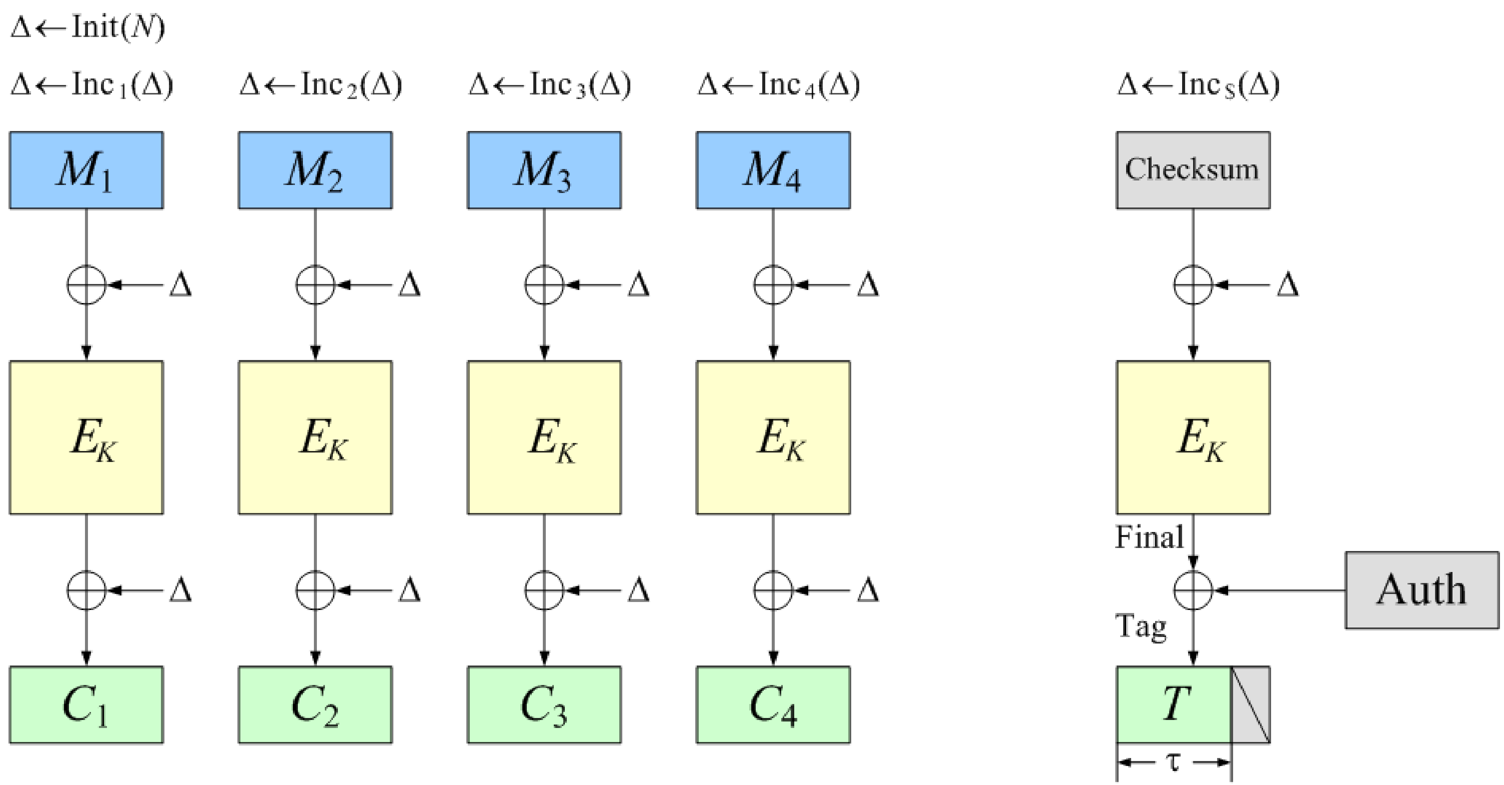


+

- 1 AES-128 call per block
- perfectly parallelizable
- only forgery with nonce reuse
- associated data

Nonce-based: AES-OCB

[RBBK01]
[BR02]
[R02]
[R04]
[KR11]



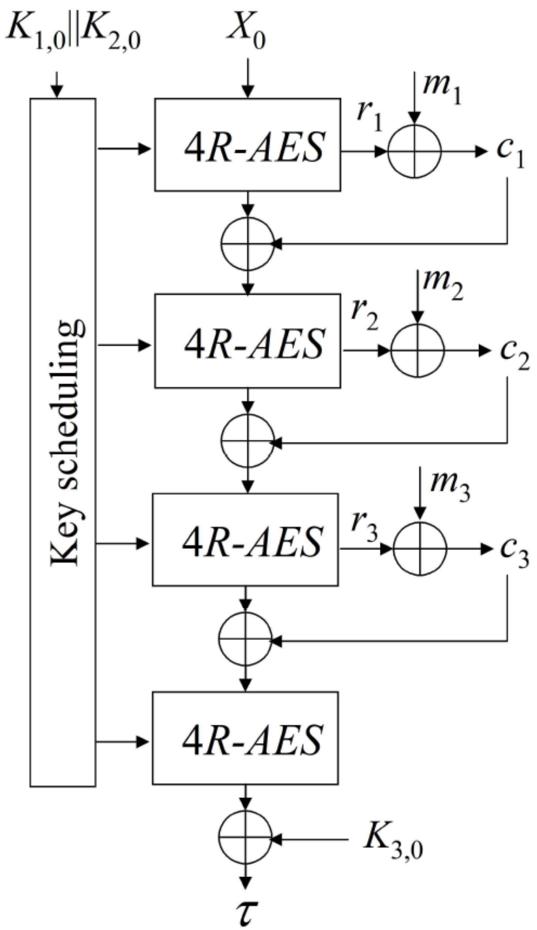
+

- 1 AES-128 call per block
- perfectly parallelizable
- only forgery with nonce reuse
- associated data

-

- enc/dec different
- state 4x128 bits
- (patents pending)

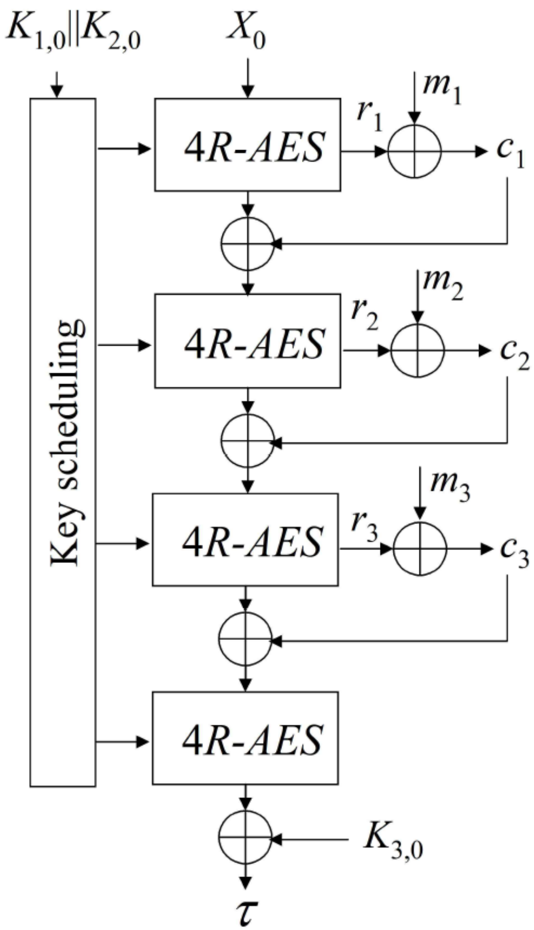
ASC-1



$X_0 = E_K(0^{70}||00||Cntr)$

$K_{1,0} = E_K(0^{70}||01||Cntr)$, $K_{2,0} = E_K(0^{70}||10||Cntr)$, $K_{3,0} = E_K(l(M)||0^6||11||Cntr)$

ASC-1



+

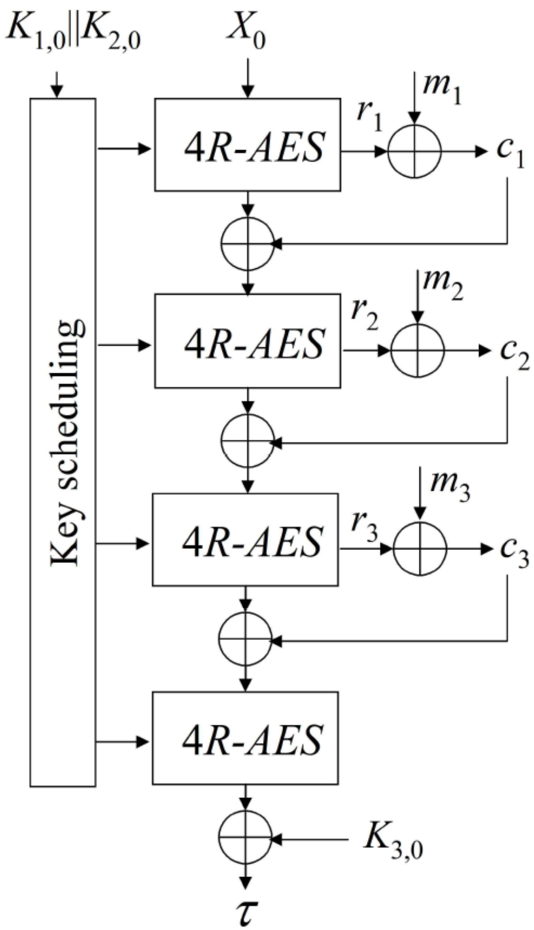
- only 4 AES-128 rounds per block
- enc/dec similar

$X_0 = E_K(0^{70}||00||Cntr)$

$K_{1,0} = E_K(0^{70}||01||Cntr)$, $K_{2,0} = E_K(0^{70}||10||Cntr)$, $K_{3,0} = E_K(l(M)||0^6||11||Cntr)$

[JK11]

ASC-1



$X_0 = E_K(0^{70}||00||Cntr)$

$K_{1,0} = E_K(0^{70}||01||Cntr)$, $K_{2,0} = E_K(0^{70}||10||Cntr)$, $K_{3,0} = E_K(l(M)||0^6||11||Cntr)$

+

- only 4 AES-128 rounds per block
- enc/dec similar

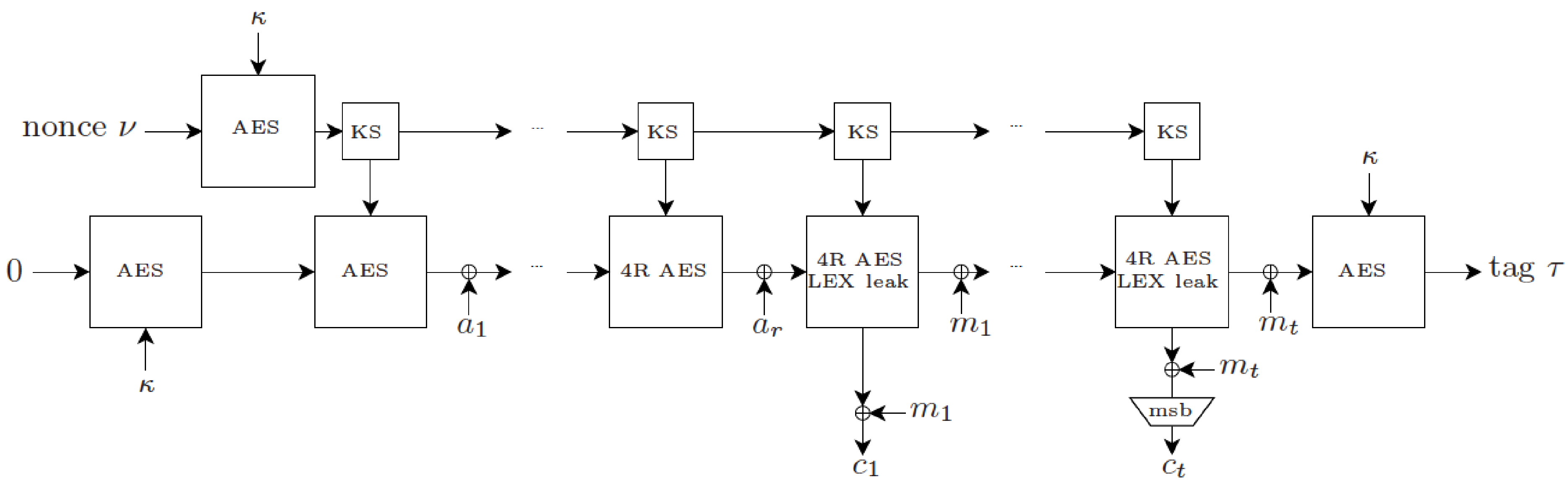
-

- state 4x128 bits
- serial
- state recovery with nonce reuse
- slow in compact ASIC implementation
- no associated data

Our Goal

- Design of a dedicated AE scheme which would:
  - require less operations on average
  - be compact in hardware (for both encryption and decryption)
  - have low power and low energy figures
  - be good in software
    - PC (AES-NI)
    - Embedded (usually not parallelizable)
  - rely on some previous cryptanalysis

ALE



a_i = associated data

m_i = message

c_i = ciphertext

AES = AES-128

κ = 128-bit key

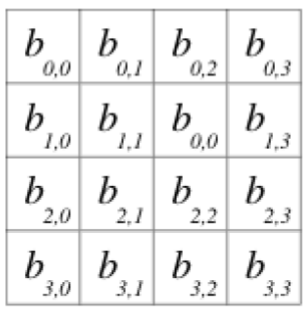
τ = tag

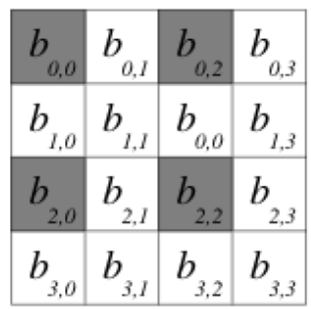
Initialization: nonce, AES with master k, 0, AES with master k, AES with ks

Processing Associated Data: xor with state, 4R AES

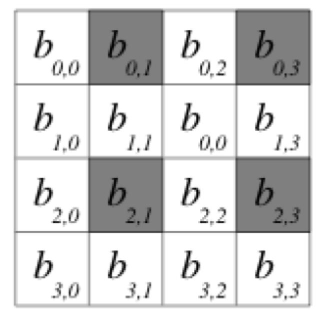
Processing Message: xor with message, 4R AES LEX leak

LEX leak for ALE encryption



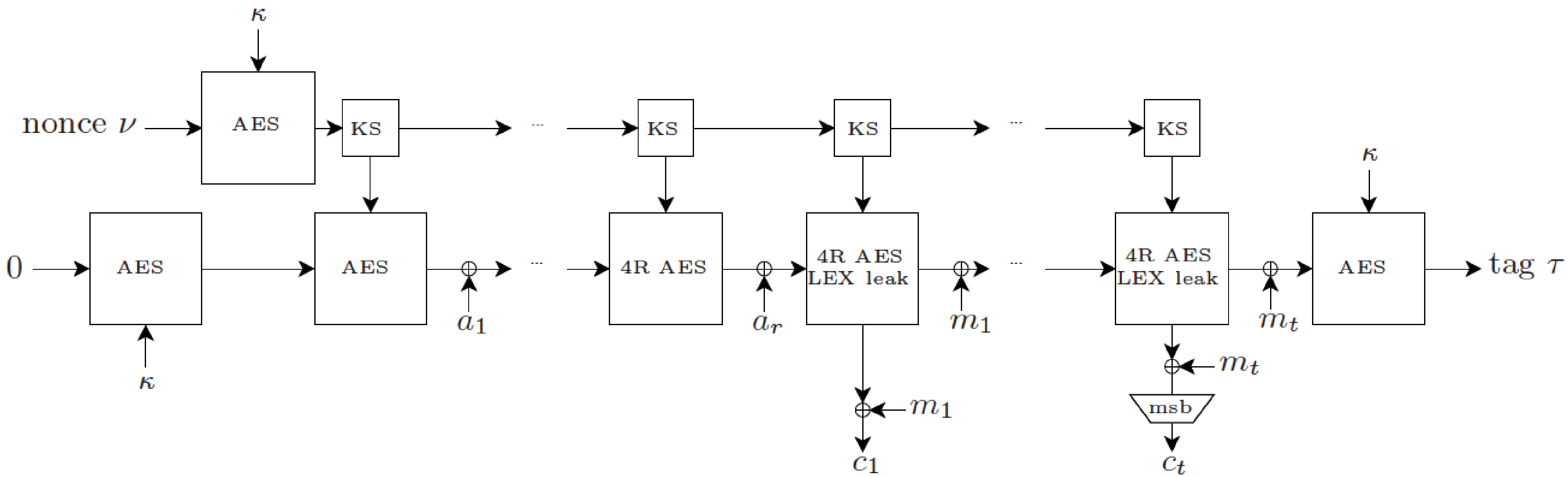


odd rounds



even rounds

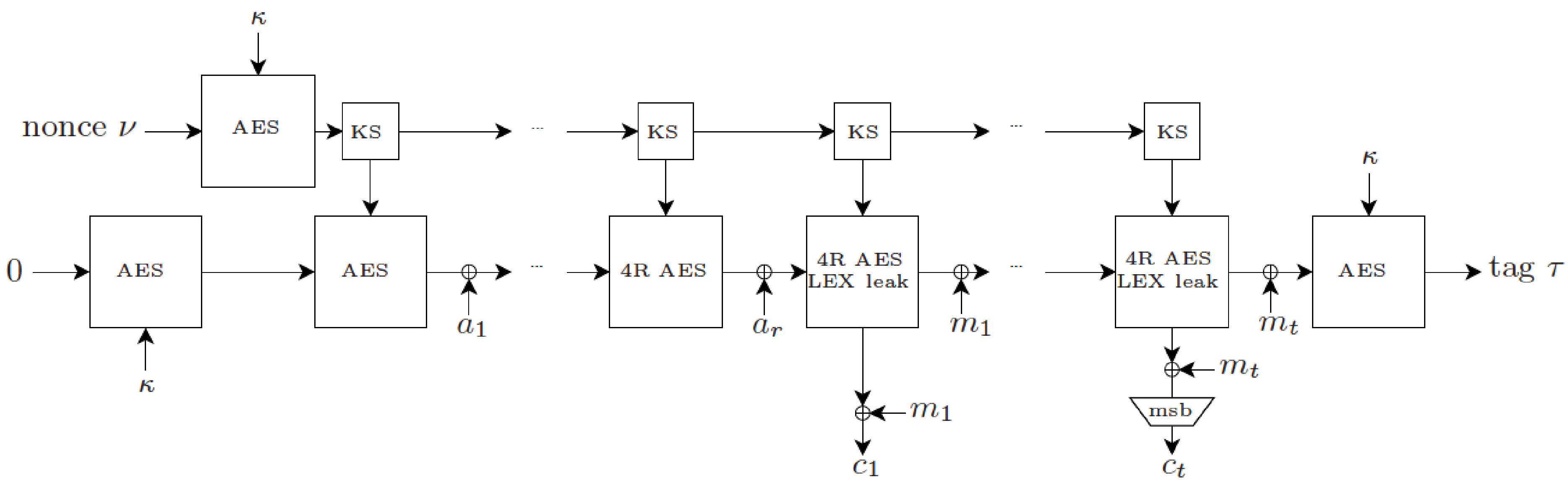
ALE



a_i	=	associated data	AES	=	AES-128
m_i	=	message	κ	=	128-bit key
c_i	=	ciphertext	τ	=	tag

Initialization: nonce, AES with master k, 0, AES with master k, AES with ks
Processing Associated Data: xor with state, 4R AES
Processing Message: xor with message, 4R AES LEX leak
Finalization: encrypt with AES

ALE



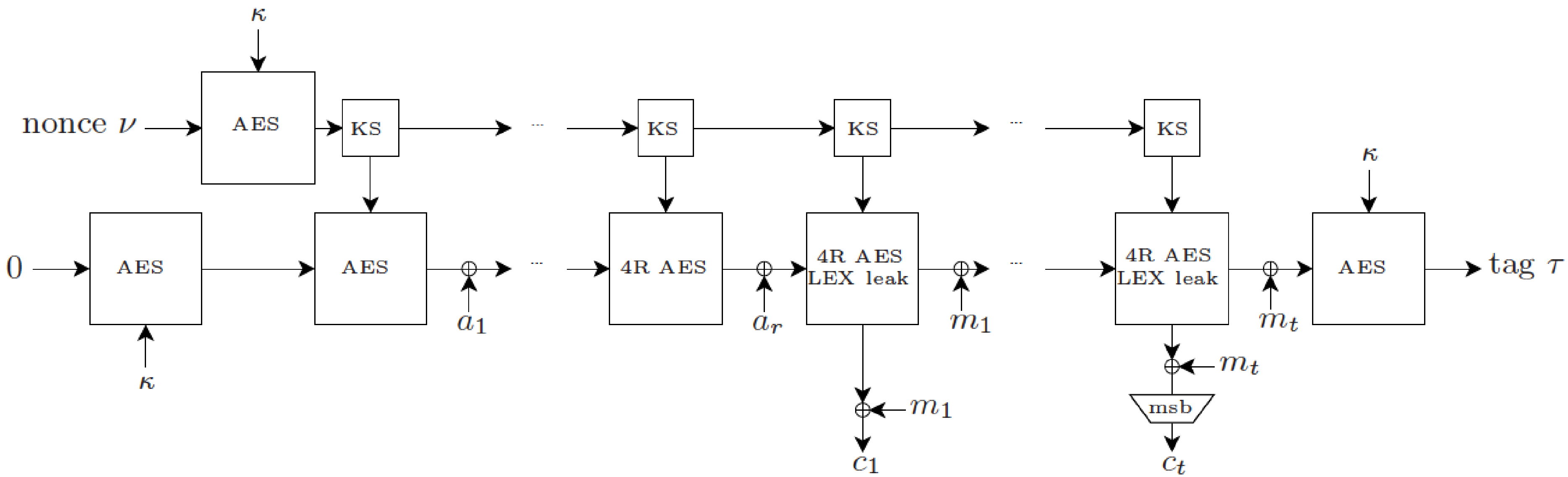
a_i	=	associated data
m_i	=	message
c_i	=	ciphertext

AES	=	AES-128
κ	=	128-bit key
τ	=	tag

\+

- only 4 AES-128 rounds per block
- enc/dec similar
- state 2x128 bits
- faster in compact ASIC implementation
- associated data

ALE



a_i = associated data

m_i = message

c_i = ciphertext

AES = AES-128

κ = 128-bit key

τ = tag

+

- only 4 AES-128 rounds per block
- enc/dec similar
- state 2x128 bits
- faster in compact ASIC implementation
- associated data

-

- serial
- state recovery with nonce reuse

Assumptions for ALE

- **Assumption 1. Nonce-respecting adversary:** A nonce is only used once with the same master key for encryption

- **Assumption 2. Abort on verification failure:** No additional information returned if tampering is detected (in particular, no plaintext blocks)

Claims for ALE

- **Claim 1. State recovery:** State recovery with complexity = *t* data blocks succeeds with prob at most $t2^{-128}$
- **Claim 2. Key recovery:** State recovery with complexity = *t* data blocks succeeds with prob at most $t2^{-128}$, even if state recovered
- **Claim 3. Forgery w/o state recovery:** forgery not involving key/state recovery succeeds with prob at most 2^{-128}

Lightweight ASIC implementation for ALE

- ALE implemented using as base AES architecture the smallest available [Moradi et al., Eurocrypt 2011]

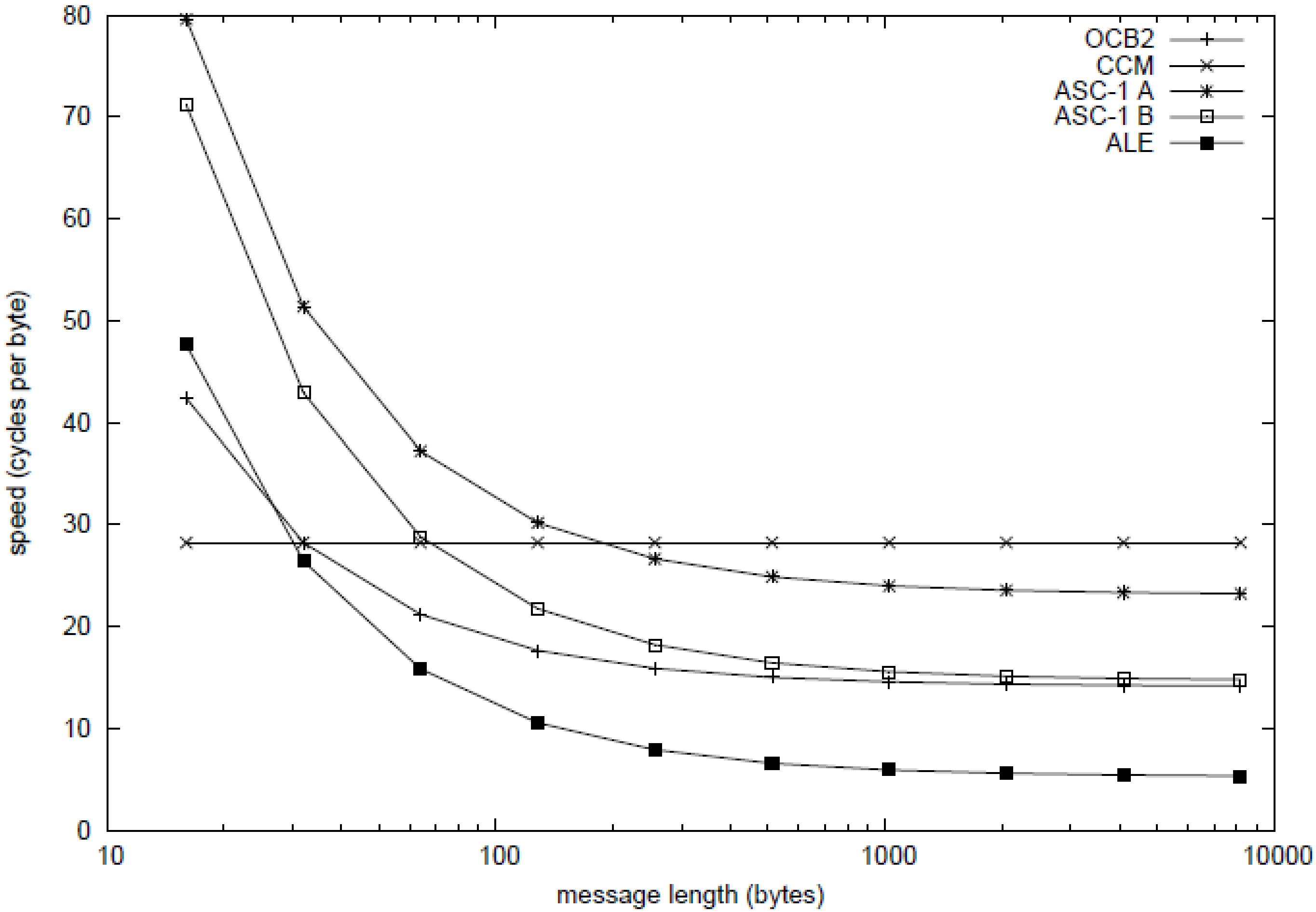
- Reference algorithms were implemented using the same starting AES

- STMicroelectronics 65 nm CMOS LP-HVT, Synopsis 2009.06, 20 MHz

Lightweight ASIC implementation for ALE

Design	Area (GE)	Net per 128-bit block (clock cycles)	Overhead per message (clock cycles)	Power (uW)
AES-ECB	2,435	226	-	87.84
AES-OCB2	4,612	226	452	171.23
AES-OCB2 e/d	5,916	226	452	211.01
ASC-1 A	4,793	370	904	169.11
ASC-1 A e/d	4,964	370	904	193.71
ASC-1 B	5,517	235	904	199.02
ASC-1 B e/d	5,632	235	904	207.13
AES-CCM	3,472	452	-	128.31
AES-CCM e/d	3,765	452	-	162.15
ALE	**2,579**	**105**	678	94.87
ALE e/d	**2,700**	**105**	678	102.32

Lightweight ASIC implementation for ALE



Software implementation of ALE

- Target platforms:
  - Sanby Bridge 3.1GHz (using AES-NI)
  - Embedded (estimated)

- Parallel or multiple message at a time

- Standard Sandy Bridge desktop @ 3.1 GHz

- Repeated 100.000 and averaged

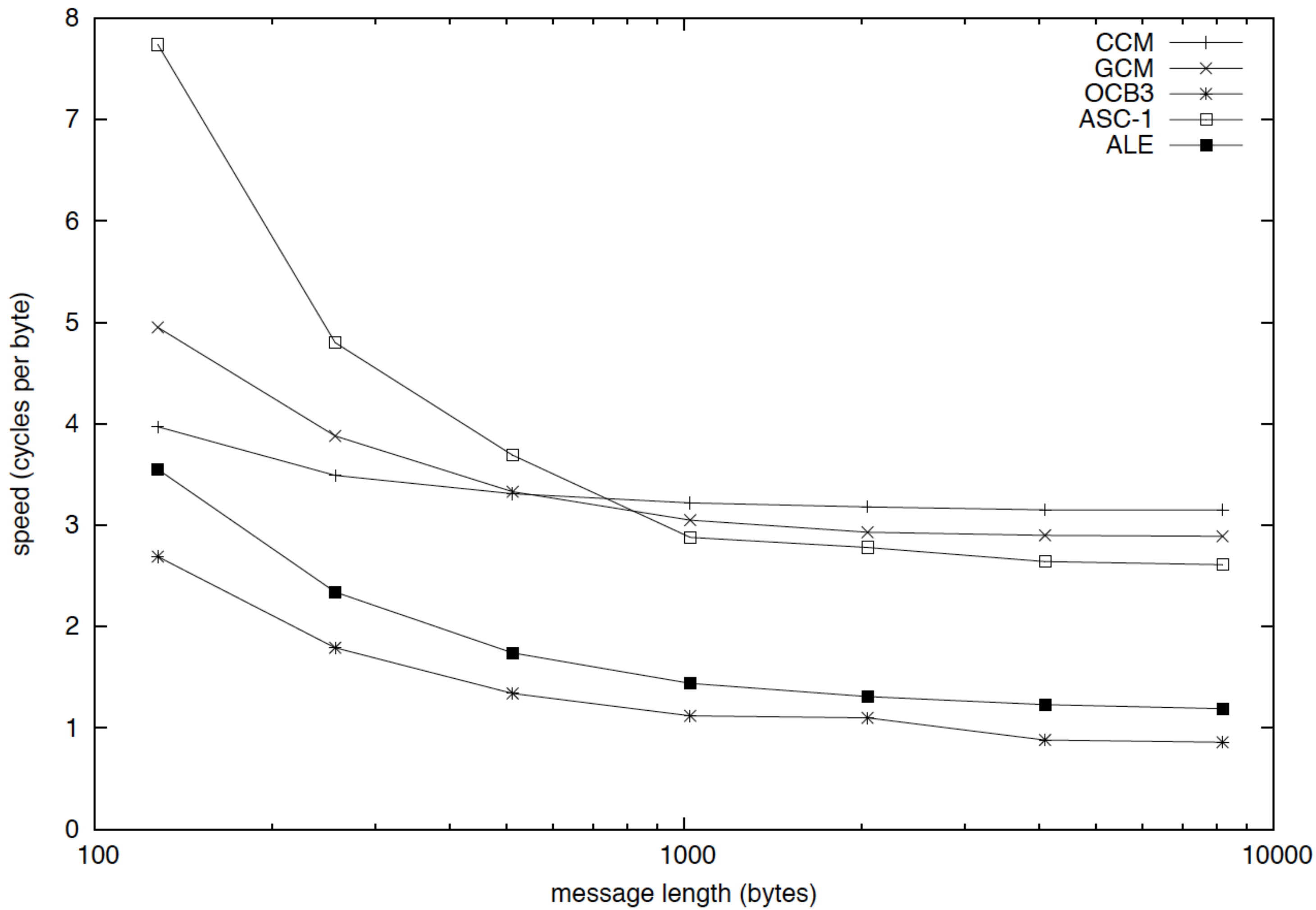
Software implementation of ALE (Sandy Bridge)

- cycles per byte (AES-NI)

Algorithm	message length (bytes) 128	256	512	1024	2048	4096	8192
ECB	1.53	1.16	0.93	0.81	0.75	0.72	0.71
CTR	1.61	1.22	0.99	0.87	0.80	0.77	0.76
CCM*	3.97	3.49	3.31	3.22	3.18	3.15	3.15
GCM	4.95	3.88	3.33	3.05	2.93	2.90	2.89
OCB3	2.69	1.79	1.34	1.12	1.00	0.88	0.86
ASC-1	7.74	4.80	3.69	2.88	2.78	2.64	2.61
ALE*	3.55	2.34	1.74	1.44	1.31	1.23	1.19

Software implementation of ALE (Sandy Bridge)

- cycles per byte (AES-NI)



Software implementation of ALE (embedded)

- Serial constructions usually do not cause large overhead
- Estimated 2 to 2.5 time faster than AES-OCB

Conclusions

- Dedicated nonce-based AES-based AEAD design
- Reuses some cryptanalysis of Pelican-MAC and LEX
- Small hardware footprint
- Fast software (measured with AES-NI, estimated embedded)

Thank you!