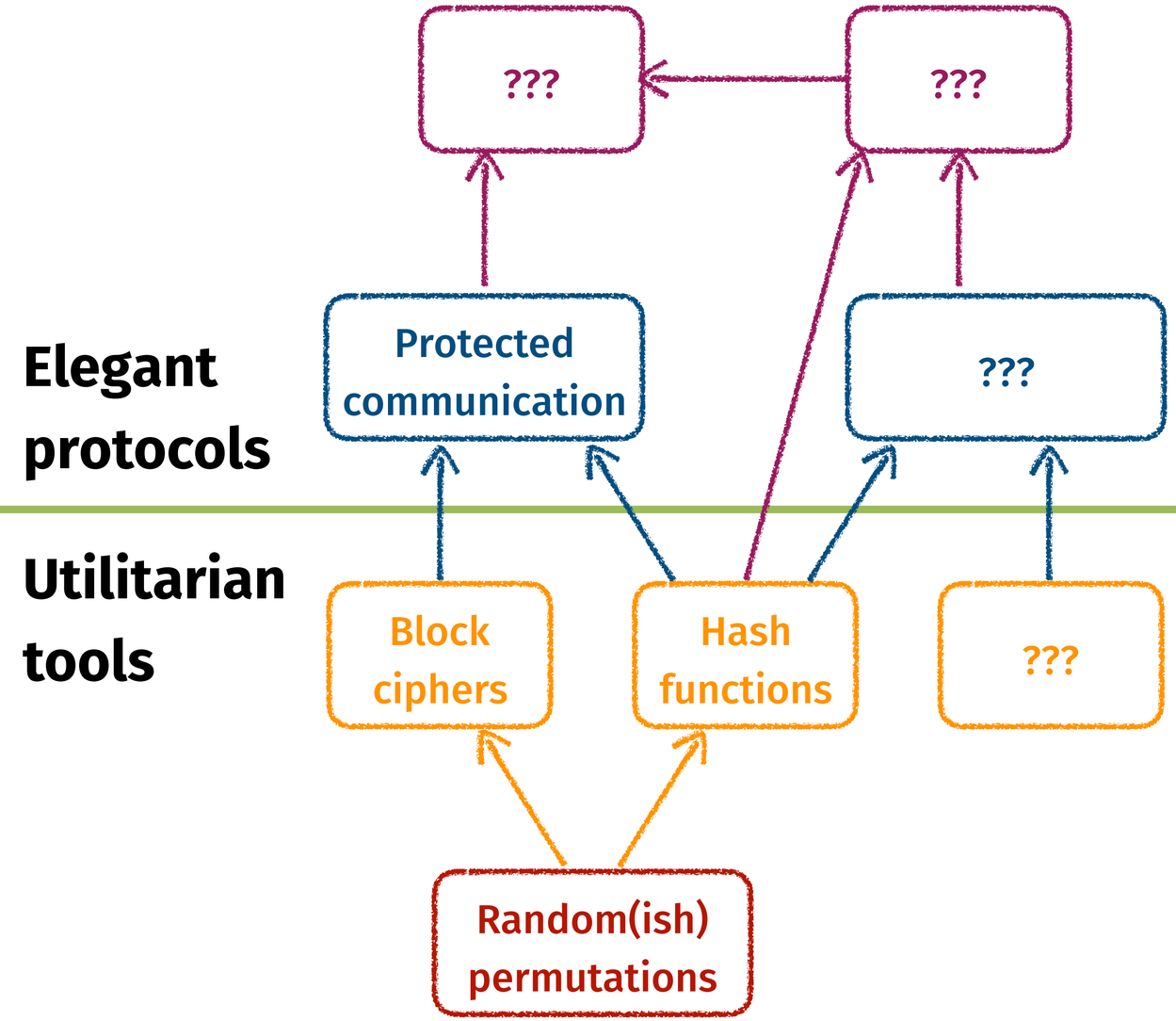
Lecture 12: Authenticated Encryption, continued

- Homework 6 has been posted, due after spring break
  - We need to adjust question 2; please wait until we post an update
- No discussion section tomorrow
- Have a good spring break!

Crypto in this course

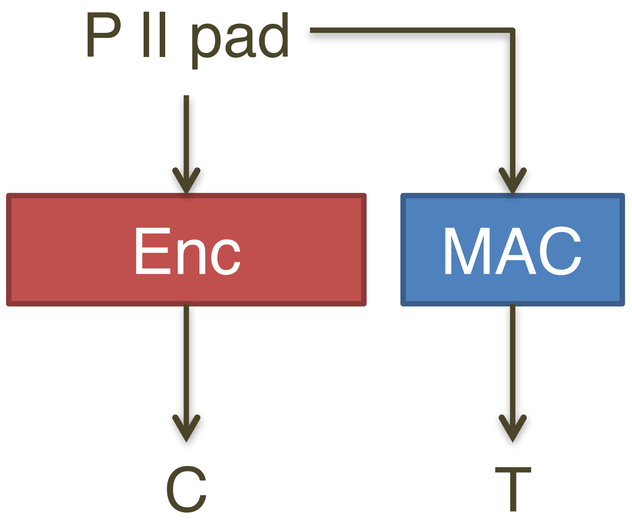


Elegant protocols

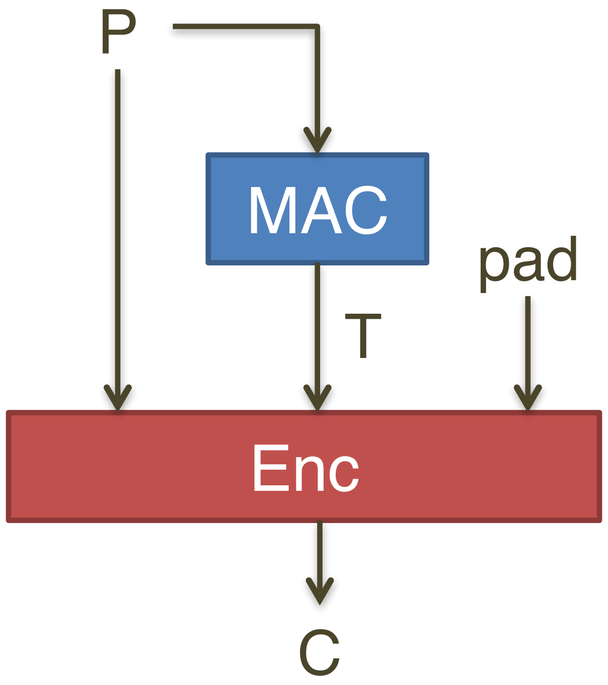
Utilitarian tools

Combining Enc and MAC *generically*

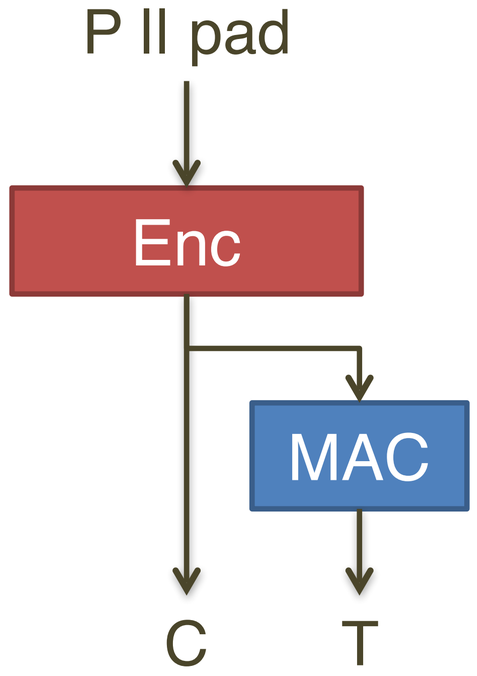
Enc and MAC



MAC then Enc



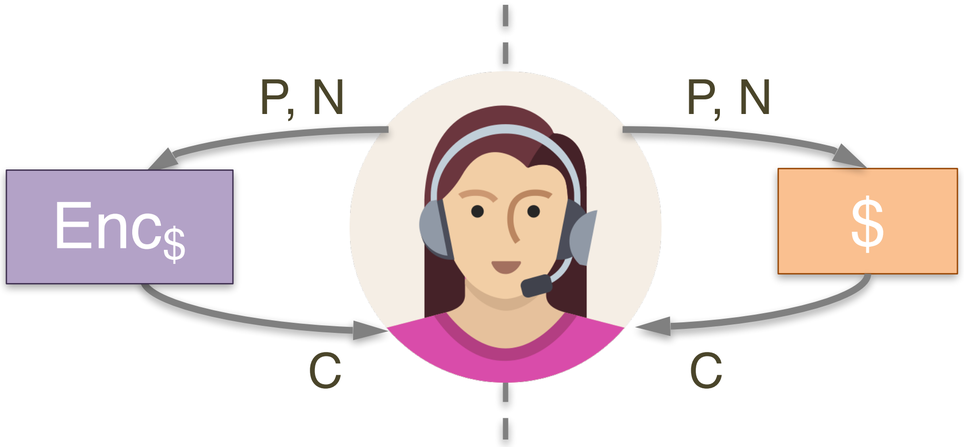
Enc then MAC



Confidentiality	None	CPA	CCA!
Integrity	*Plaintext integrity*: Cannot make CT that decrypts to message that sender never encrypted		*Ciphertext integrity*: Cannot make new valid CTs, only know sender-made ones

Security Definitions: Encryption xor Authentication

IND$-CPA for nonce-respecting Eve:



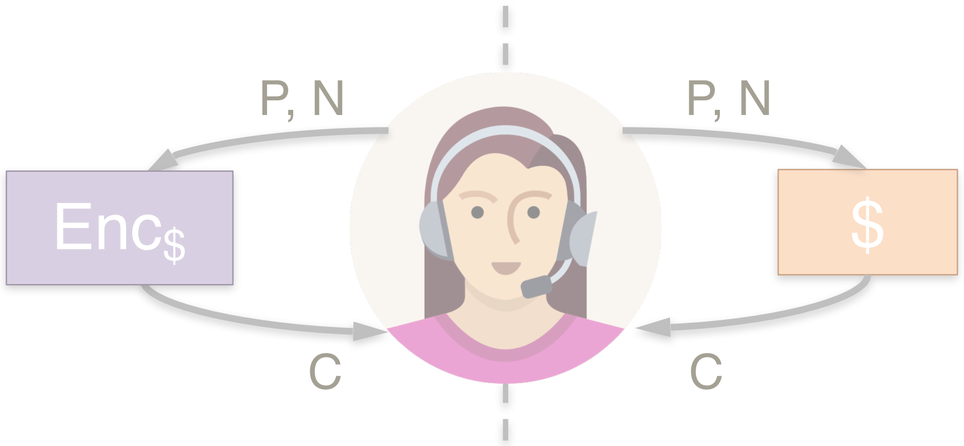
Restriction: Mallory can't re-use nonce

EU-CMA:

Restriction: Mallory cannot verify a MAC tag that Alice produced

Security Definition: Encryption *and* Authentication

IND$-CPA for nonce-respecting Eve:



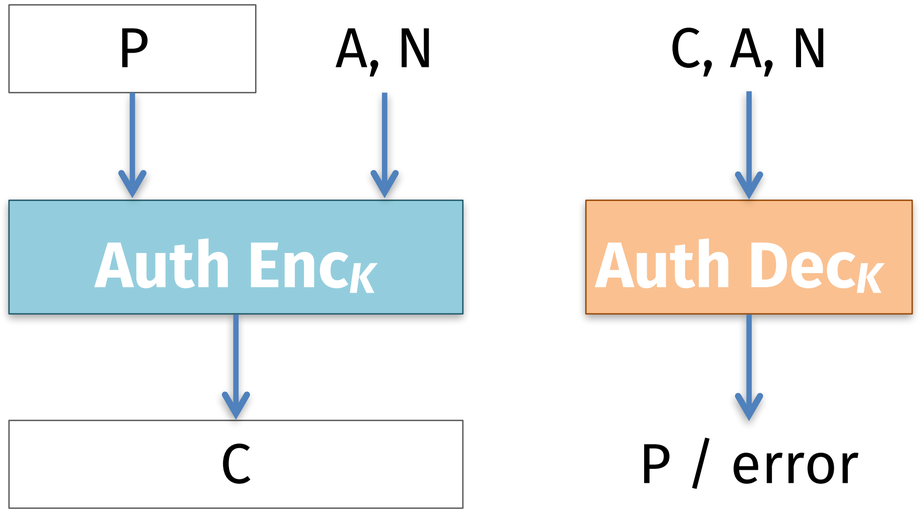
EU-CMA:

Restrictions

- Cannot reuse nonce
- Cannot ask to decrypt CT made by Alice

Def. Authenticated Encryption with Associated Data (AEAD)

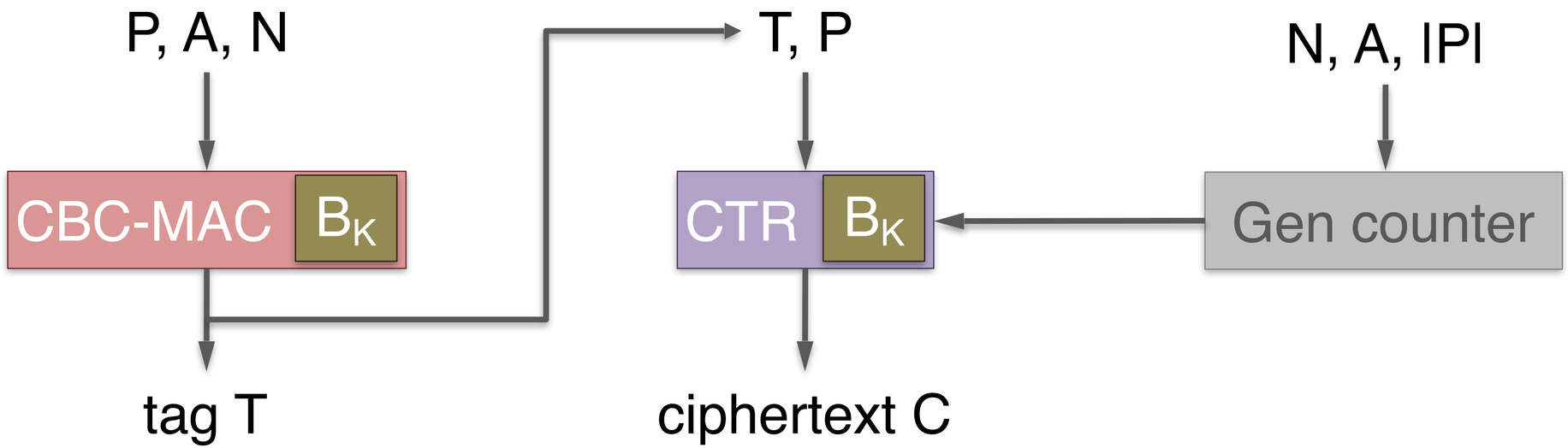
- KeyGen: randomly choose K, as usual
- AuthEnc_K(authenticated data A, private + auth data P, nonce N) → ciphertext C of length $|C| \geq |P|$
- $\text{AuthDec}_K(C, A, N)$ → P or error



Benefits

- Better security: satisfies Moxie's doom principle, resists some side channel attacks
- Simplicity: developers have fewer decisions (i.e., opportunities for mistakes)
- Performance: save in time + space costs, also often only need 1 key

Counter with CBC-MAC (CCM) *[Housley, Whiting, Ferguson 2003]*



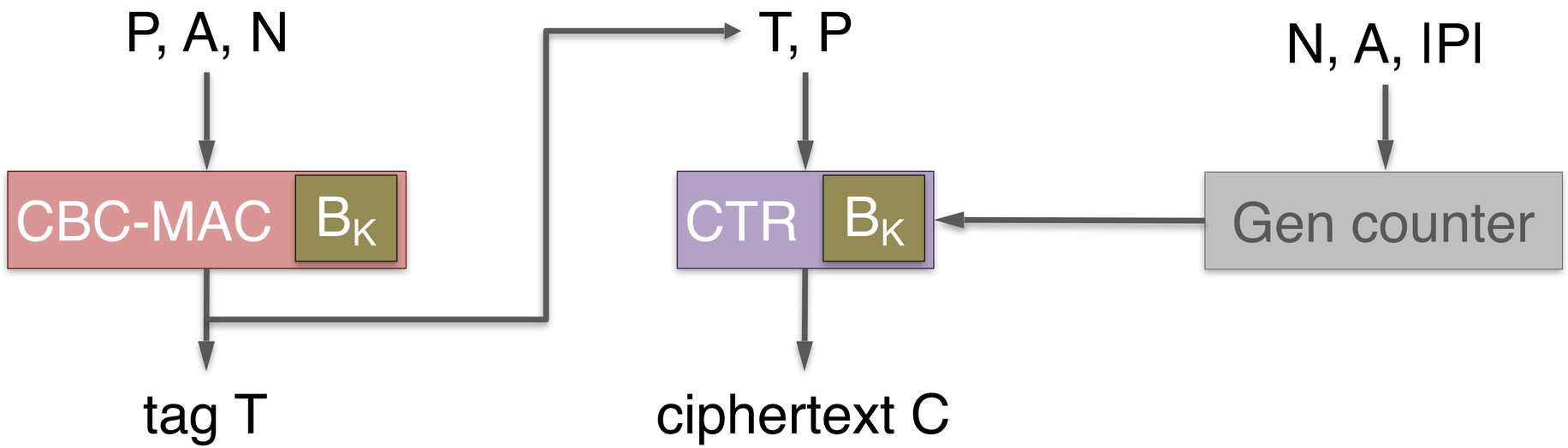
Novelties

- CTR and CBC-MAC can use same key
- Counter generation design solves CBC-MAC's length extension problem

Drawbacks

- CTR mode → nonce reuse is catastrophic for privacy
- Using CBC mode for the MAC prevents parallelization and pipelining

Performance issues with CCM



1. Running time: 2|P| block cipher calls
2. Not streaming-friendly: can't forget P after CBC-MAC, need it again for CTR
3. Cannot pipeline: must finish one CBC block cipher call before starting the next

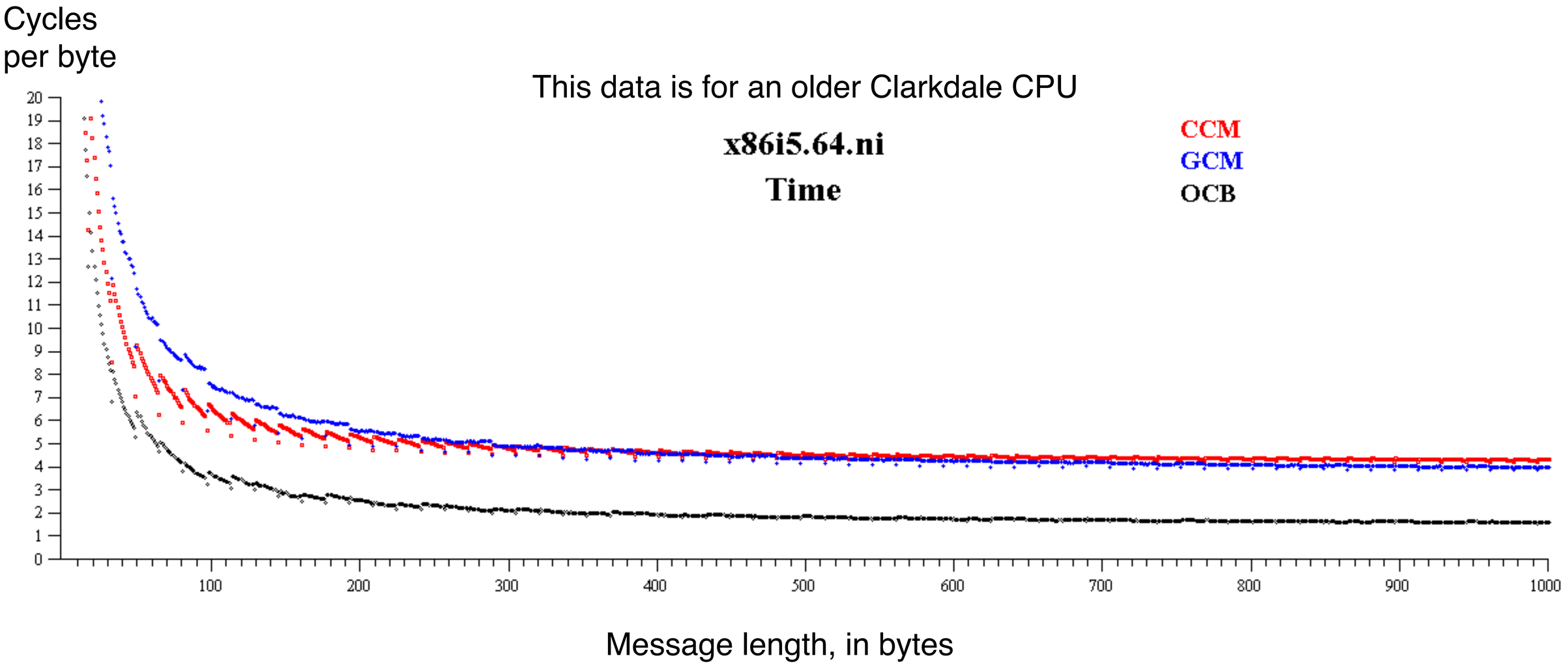
Performance comparison of CBC and CTR modes

Throughput of symmetric encryption on Skylake Core i5 running at 2.7 GHz

Data taken from https://cryptopp.com/benchmarks.html

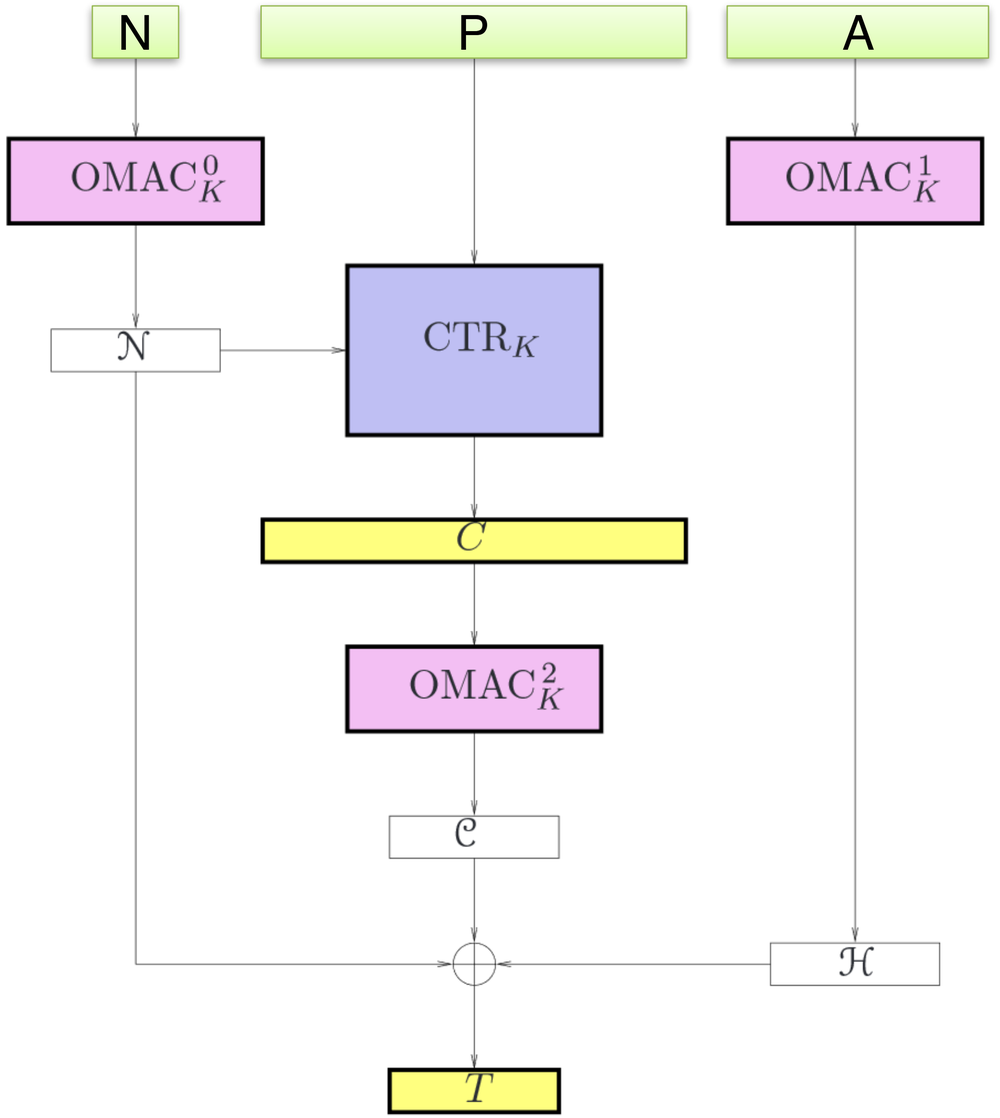
Mode of operation	Key length	GiB/second	Cycles per byte
AES in CTR mode	128 bit	4.4	0.57
	192 bit	3.8	0.67
	256 bit	3.3	0.77
AES in CBC mode	128 bit	1.0	2.40
	192 bit	0.90	2.80
	256 bit	0.79	3.20

Performance vs message length

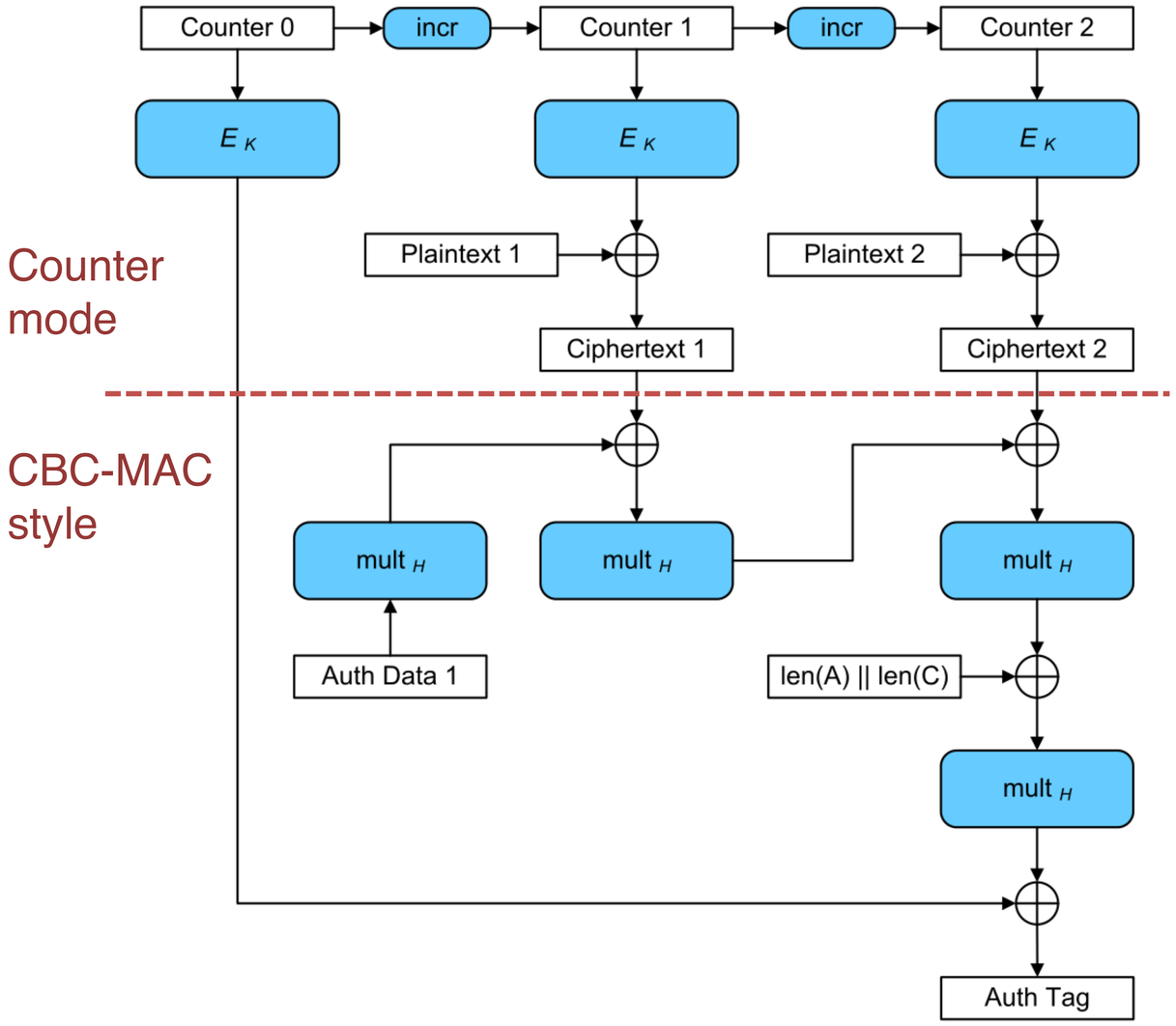


EAX mode *[Bellare, Rogaway, Wagner 2004]*

- Released one year after CCM
- Replaces CBC-MAC with OMAC
  - Well… CMAC used in NIST standard
- Novelties
  - Can pre-process associated data!
    (A can be reused across multiple P)
  - CTR and OMAC can be executed concurrently for online streaming
  - Can re-use key for CTR + all OMACs

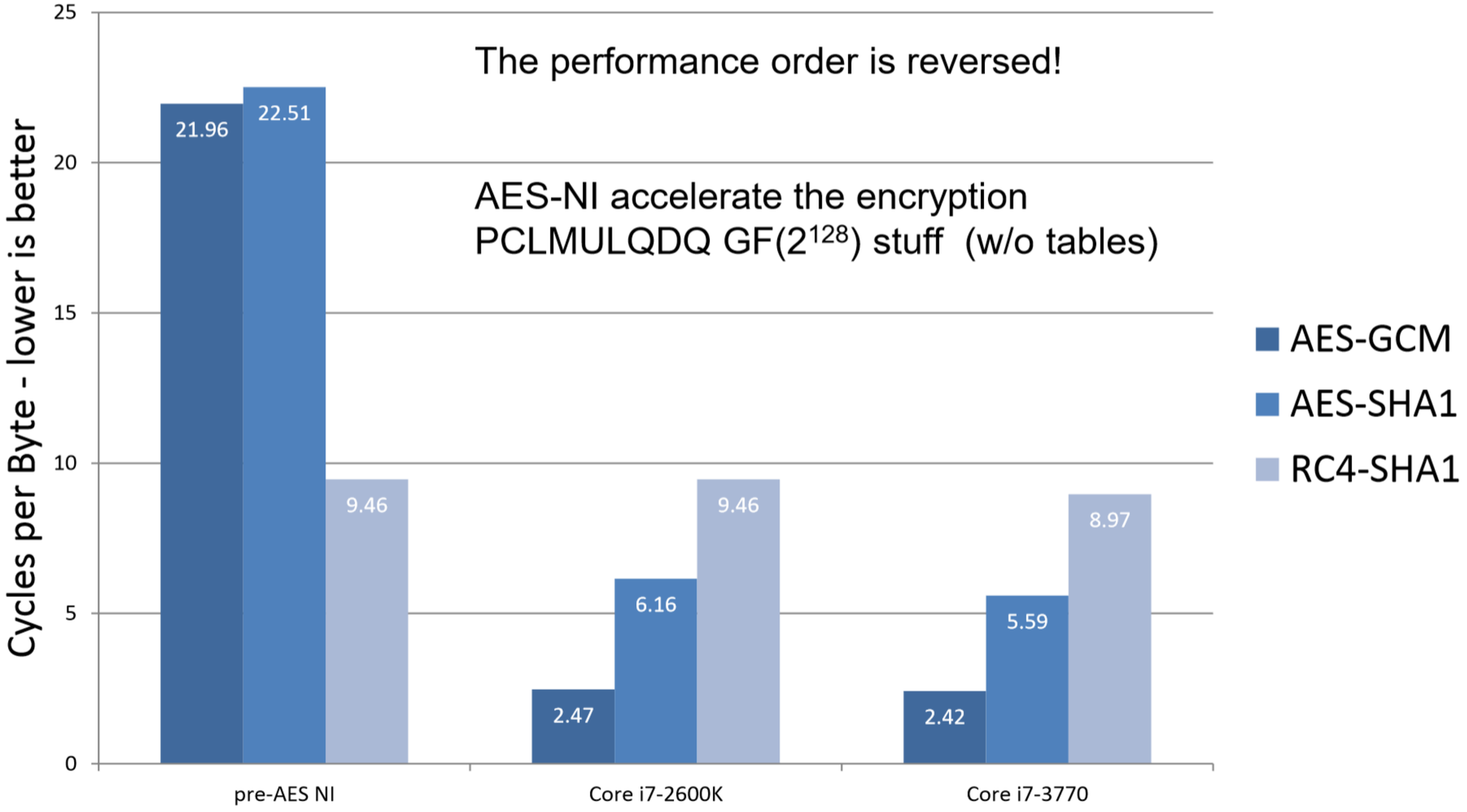


Galois/counter mode (GCM) *[McGrew, Viega 2005]*

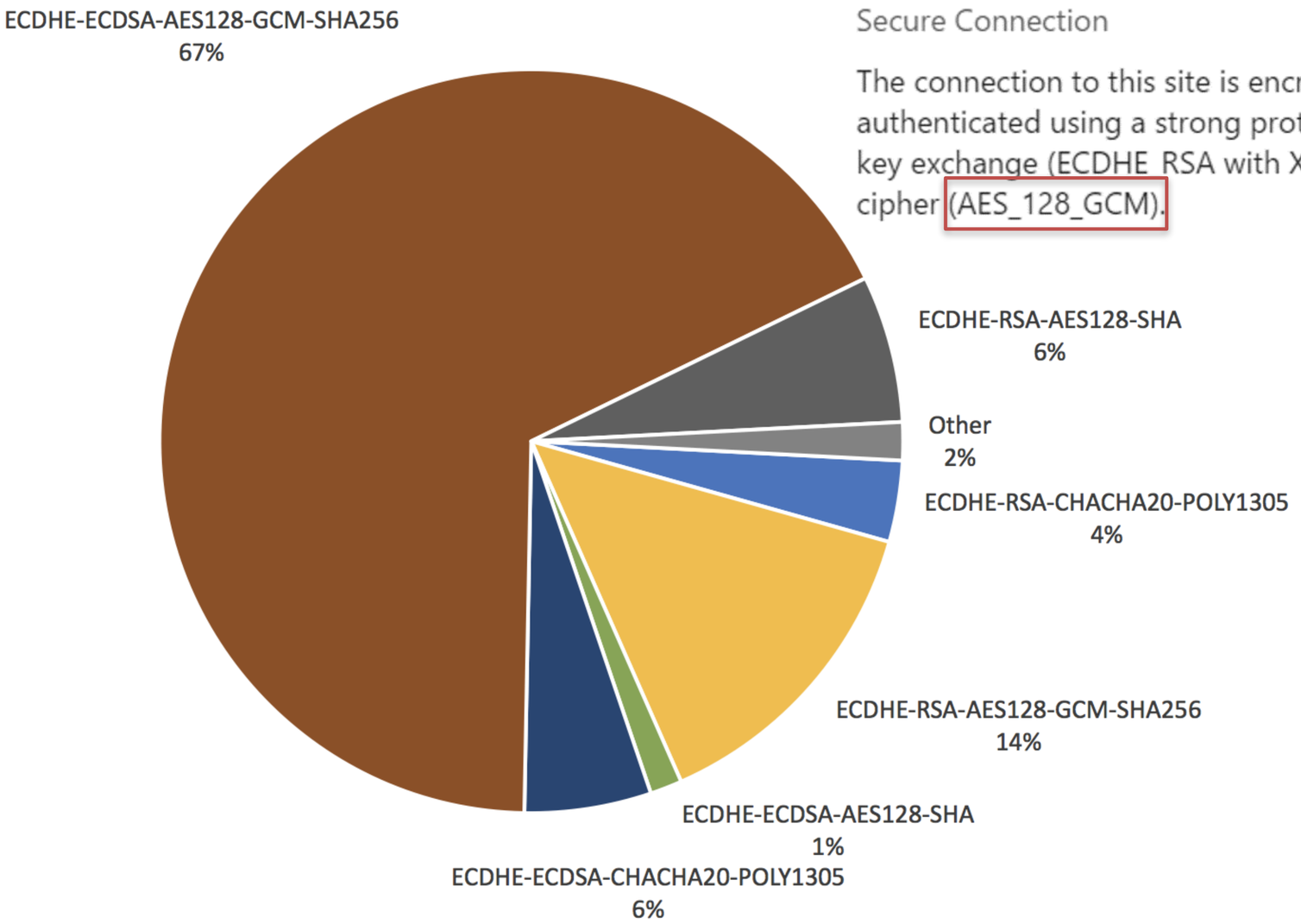


- Novelty
  - Only one round of block cipher calls
  - Intel's AES-NI includes a PCLMULQDQ instruction for mult_H in hardware
- Drawbacks
  - Assumes block length = 128 (built for AES)
  - Very difficult to implement in software, good chance of doing it wrong
  - Extensive cryptanalysis has exposed weak keys ⟹ CAESAR competition

Speed of GCM in hardware (2013)



GCM: widespread use in practice (Dec 2017)



What features could we add?

- *Even more speed:*
  only |**P**| block cipher calls (and nothing else)
- *Even more security:*
  removing the need for the nonce **N** altogether

What features could we add?

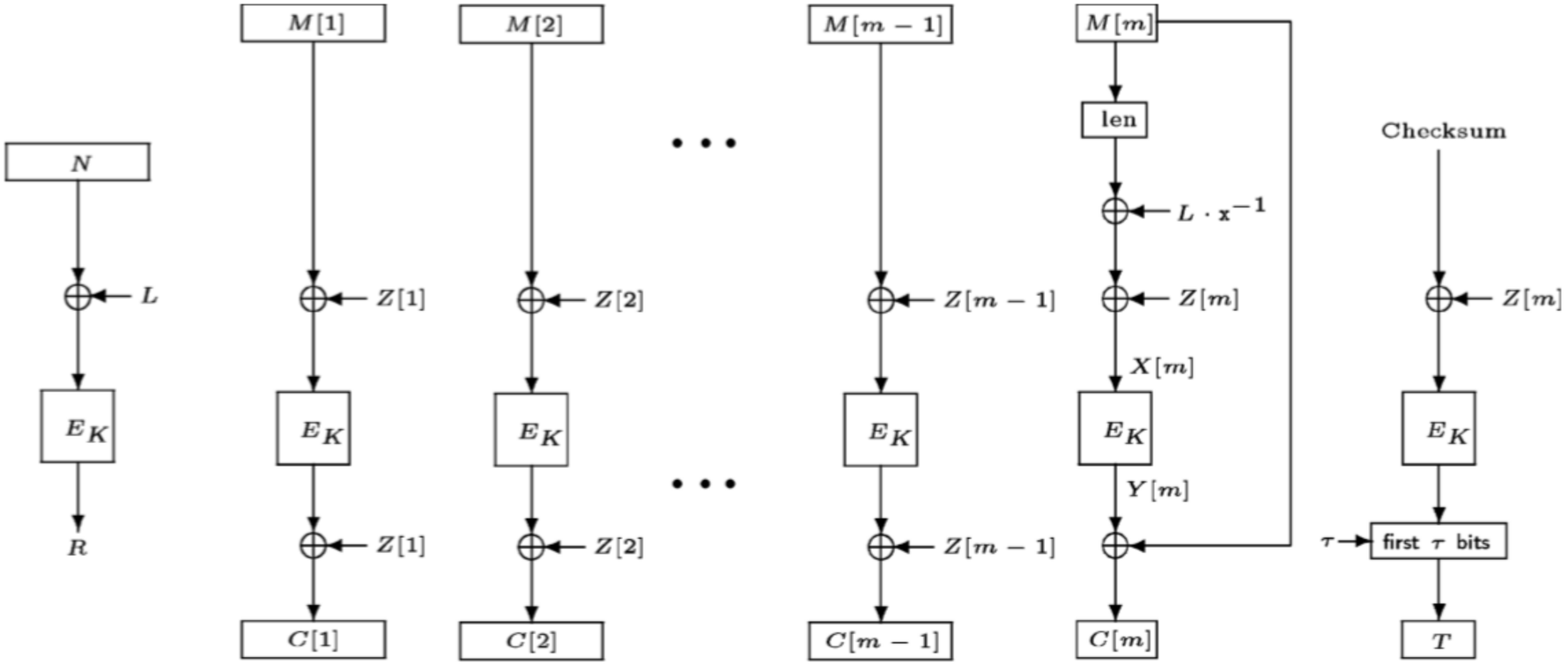
- *Even more speed:*
  only |**P**| block cipher calls (and nothing else)
- *Even more security:*
  removing the need for the nonce **N** altogether

Offset codebook mode (OCB) *[Rogaway 2002]*

History

- Jutla 2001: Integrity aware parallelizable mode (IPAM)
- Rogaway 2002: OCB

Basic idea: XEX with finalization



OCB's licenses

Taken from http://web.cs.ucdavis.edu/~rogaway/ocb/license.htm

1. License for Open-Source Software Implementations of OCB (1/9/13)
   Under this license, you are authorized to make, use, and distribute open-source software implementations of OCB. This license terminates for you if you sue someone over their open-source software implementation of OCB claiming that you have a patent covering their implementation.

2. General License for Non-Military Software Implementations OCB (1/10/13)
   This license does not authorize any military use of OCB. Aside from military uses, you are authorized to make, use, and distribute (1) any software implementation of OCB and (2) non-software implementations of OCB for noncommercial or research purposes. You are required to include notice of this license to users of your work so that they are aware of the prohibition against military use. This license terminates for you if you sue someone over an implementation of OCB authorized by this license claiming that you have a patent covering their implementation.

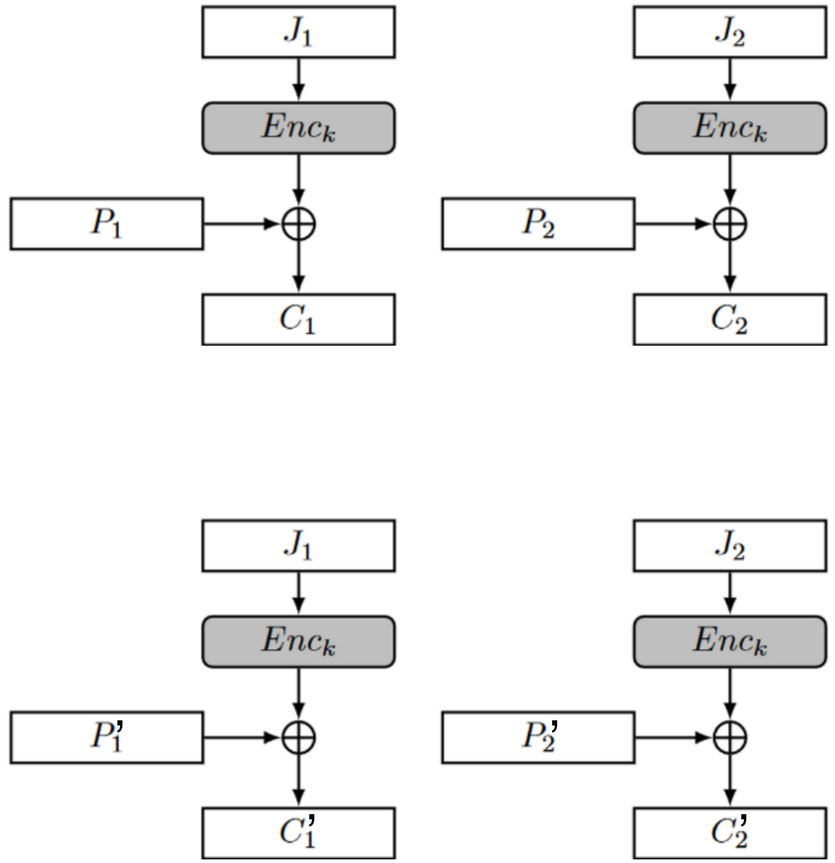
3. Patent License for OpenSSL (11/13/13)
   This license was provided at the request of the OpenSSL Software Foundation to specifically authorize use of OCB in OpenSSL.

What features could we add?

- *Even more speed:*
  only |**P**| block cipher calls (and nothing else)
- *Even more security:*
  removing the need for the nonce **N** altogether

GCM nonce reuse: destroys privacy

- For privacy, GCM = CTR
- CTR mode with repeated nonces
  = two-time pad
- We broke privacy in Lab 2



GCM nonce reuse: destroys authenticity

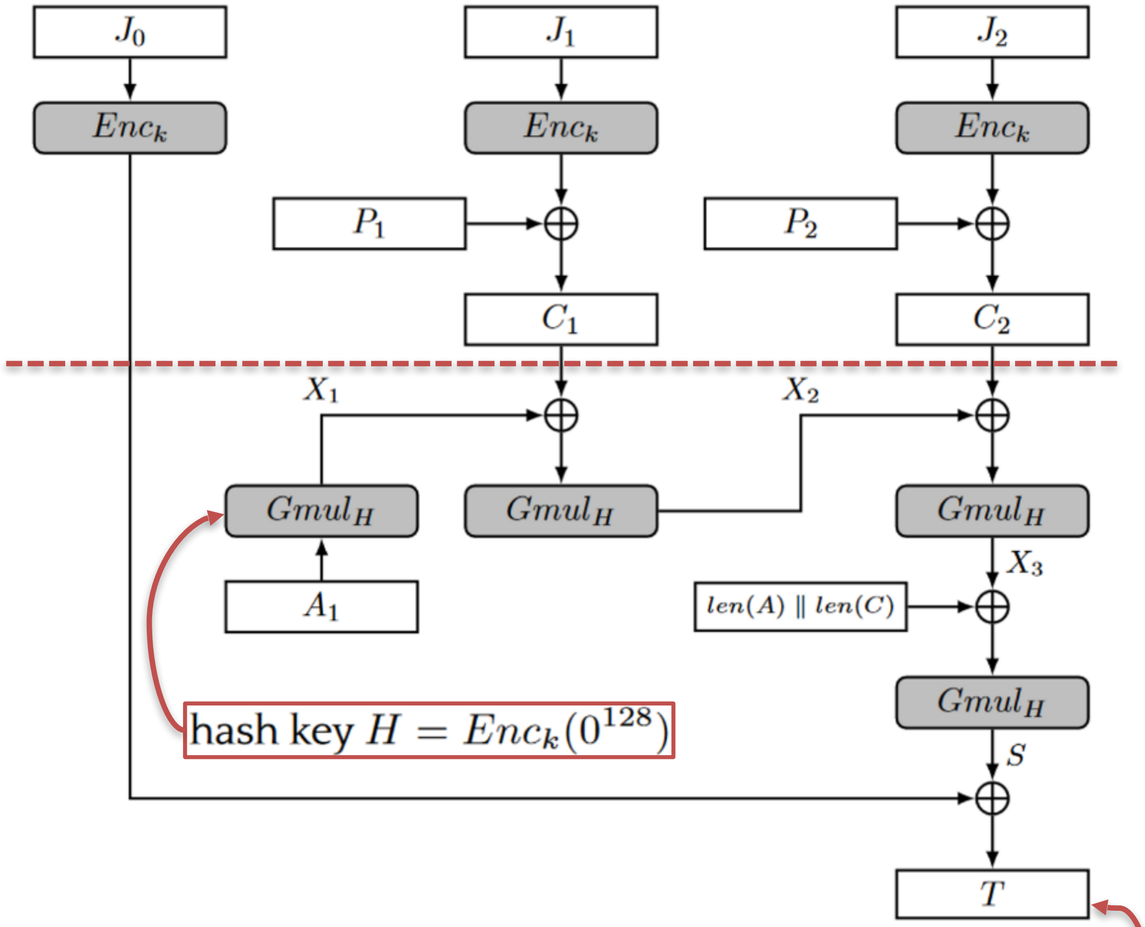
Nonce-Disrespecting Adversaries: Practical Forgery Attacks on GCM in TLS

Hanno Böck[*] Aaron Zauner[†‡] Sean Devlin[§]
Juraj Somorovsky[¶] Philipp Jovanovic[‖]

May 17, 2016

Abstract

We investigate nonce reuse issues with the GCM block cipher mode as used in TLS and focus in particular on AES-GCM, the most widely deployed variant. With an Internet-wide scan we identified 184 HTTPS servers repeating nonces, which fully breaks the authenticity of the connections. Affected servers include large corporations, financial institutions, and a credit card company. We present a proof of concept of our attack allowing to violate the authenticity of affected HTTPS connections which in turn can be utilized to inject seemingly valid content into encrypted sessions. Furthermore we discovered over 70,000 HTTPS servers using random nonces, which puts them at risk of nonce reuse if a large amount of data is sent over the same connection.



a common root in the authentication key can be constructed. Factoring these polynomials and finding the common root yields the correct value for H.

GCM nonce being chosen randomly: hurts authenticity

Nonce-Disrespecting Adversaries: Practical Forgery Attacks on GCM in TLS

Hanno Böck[*] Aaron Zauner[†‡] Sean Devlin[§]
Juraj Somorovsky[¶] Philipp Jovanovic[‖]

May 17, 2016

Abstract

We investigate nonce reuse issues with the GCM block cipher mode as used in TLS and focus in particular on AES-GCM, the most widely deployed variant. With an Internet-wide scan we identified 184 HTTPS servers repeating nonces, which fully breaks the authenticity of the connections. Affected servers include large corporations, financial institutions, and a credit card company. We present a proof of concept of our attack allowing to violate the authenticity of affected HTTPS connections which in turn can be utilized to inject seemingly valid content into encrypted sessions. Furthermore we discovered over 70,000 HTTPS servers using random nonces, which puts them at risk of nonce reuse if a large amount of data is sent over the same connection.

n	p
22	0.000000
23	0.000002
24	0.000008
25	0.000031
26	0.000122
27	0.000488
28	0.001951
29	0.007782
30	0.030767
31	0.117503
32	0.393469
33	0.864665
34	0.999665
35	1.000000

Figure 2: Probability p for nonce collision with 2^n nonces of 64 bit size

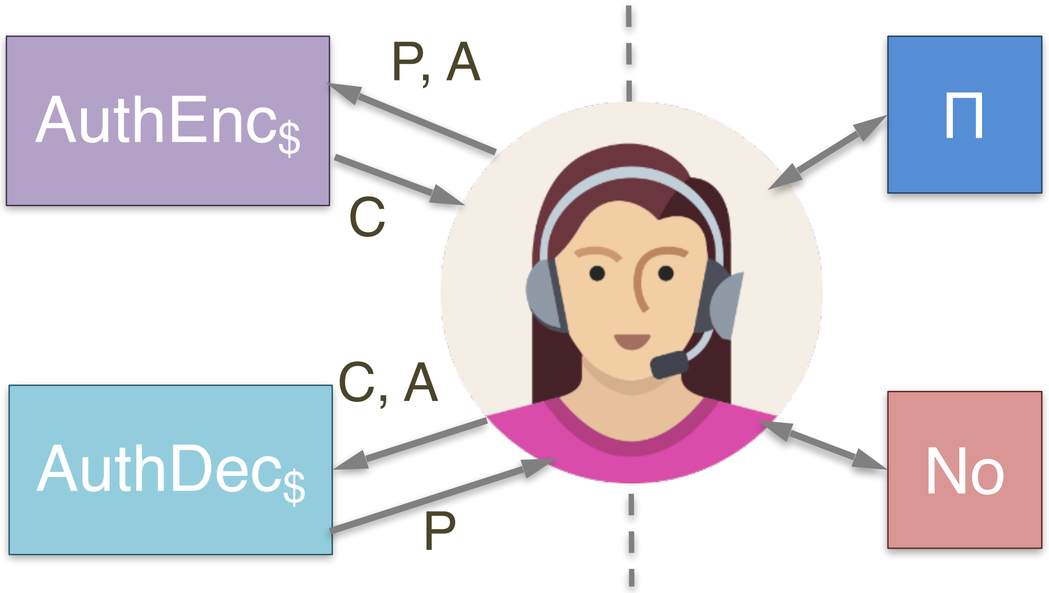
Wait... why did we want nonces in the first place?

- Main purpose for introducing nonces was to hide *frequency analysis*
  - Hide from Mallory whether Alice is encrypting the same message twice
- Then, we built systems that rely upon the nonce for more than this
- How to ensure that nonce-reuse only reveals frequency analysis?
- **Idea:** uniqueness of the message itself should also serve as a “nonce”

Misuse-resistant AE

- Novelty: Repeating *N* has limited damage
  - No impact to authenticity
  - Privacy damaged only to the extent that an adversary can detect repetitions
- Drawback: Cannot make just one pass through ***P***
  - Every bit of ***C*** must depend on every bit of ***P***
  - So, cannot output first bit of ***C*** before reading last bit of ***P***
- Corollary: can omit the nonce entirely if you want, Enc can be deterministic!
  - Note how we've come full circle on this question

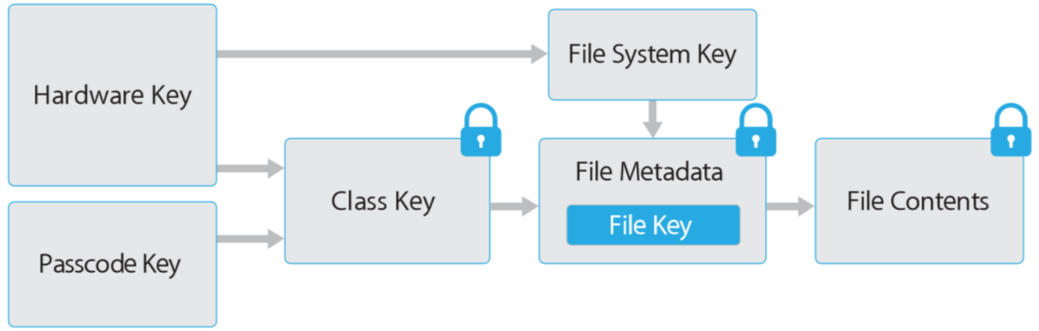
Deterministic Authenticated Encryption (DAE)



Provides key wrapping!

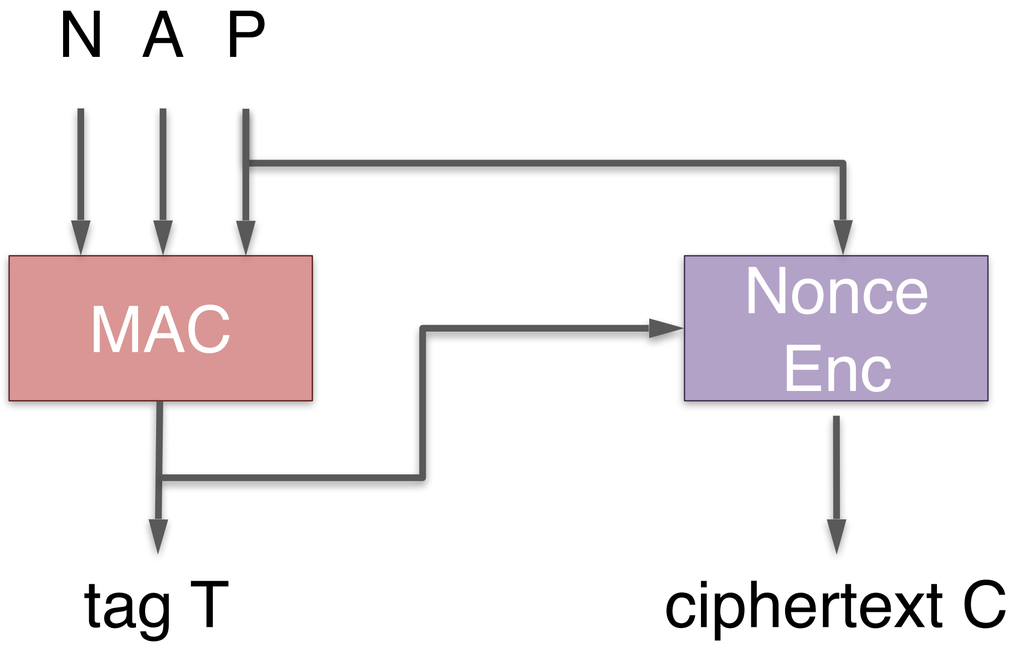
- Adversary cannot produce a valid C without knowing K
- Even C corresponding to a message P that depends on the key K somehow

Foreshadowing: will need key wrapping in full disk encryption



Synthetic Initialization Vector (SIV) *[Rogaway, Shrimpton 2006]*

- Novelty
  - Leverages structure of P, A to ensure uniqueness of CTR's nonce
  - Can be deterministic
  - Applies to a wide range of MACs
- Drawbacks
  - 2 passes, 2 keys
  - Must decrypt, then verify
  - Cannot truncate tag



Combining GCM and SIV

GCM-SIV: Full Nonce Misuse-Resistant Authenticated Encryption at Under One Cycle per Byte*

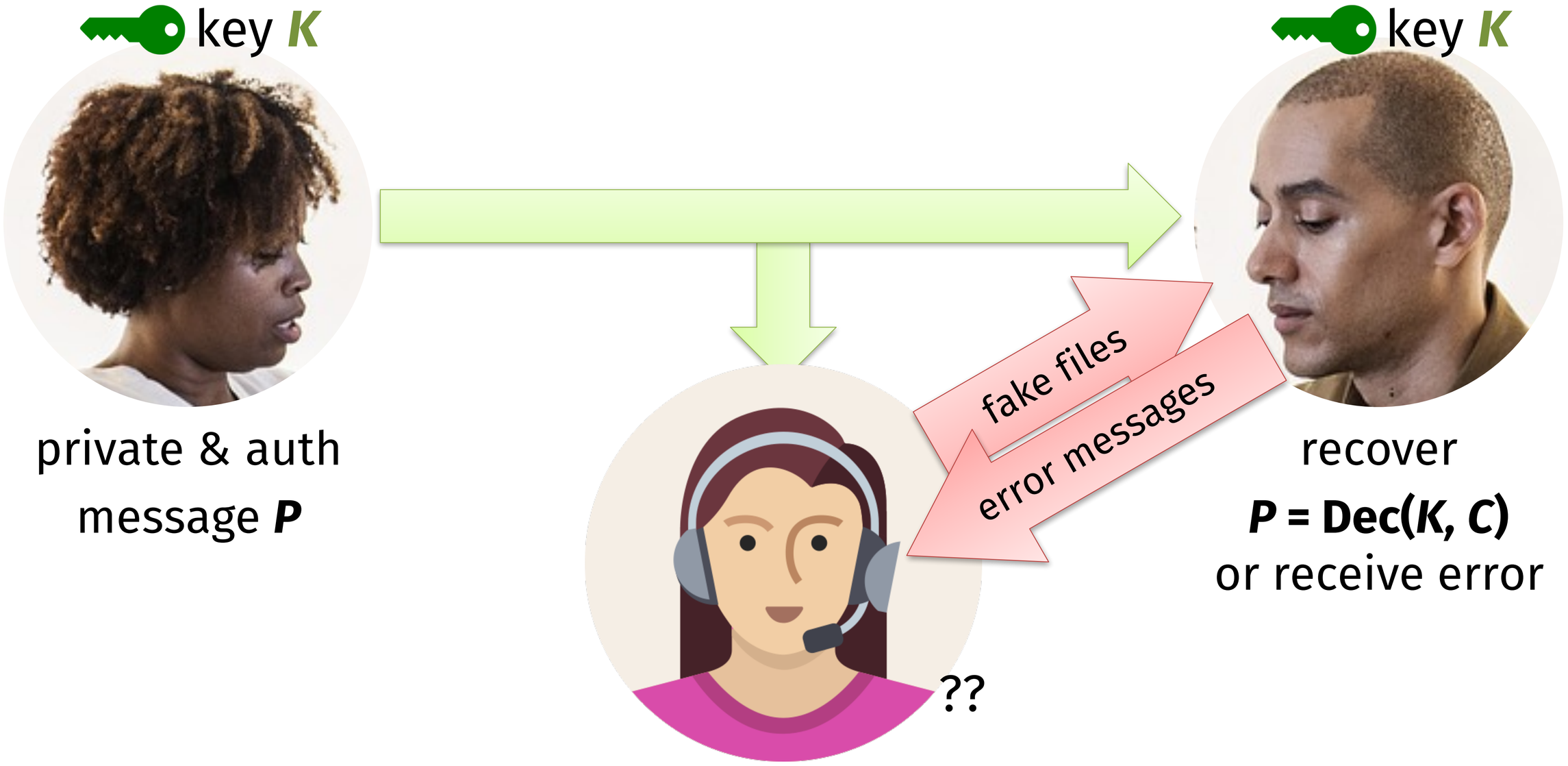
Shay Gueron† Yehuda Lindell‡

Haswell	1.17 cycle/byte
Broadwell	0.92 cycle/byte

AES-GCM-SIV: Specification and Analysis

Message Length		AES-GCM-SIV	GCM-SIV+	AES-GCM	AES-GCM-SIV	GCM-SIV+	AES-GCM
(bytes)		128-bit key	128-bit key	128-bit key	256-bit key	256-bit key	256-bit key
		ENC / DEC	ENC / DEC	ENC / DEC	ENC / DEC	ENC / DEC	ENC / DEC
16	cycles	257 / 358	129 / 133	129 / 141	306 / 445	152 / 194	154 / 201
64	cycles	361 / 456	261 / 227	193 / 190	441 / 546	292 / 305	219 / 215
1,024	C/B	1.37 / 1.17	1.25 / 0.94	0.84 / 0.79	1.69 / 1.48	1.53 / 1.22	1.1 / 1.05
2,048	C/B	1.14 / 0.88	1.09 / 0.76	0.76 / 0.71	1.43 / 1.16	1.36 / 1.03	1.00 / 0.97
4,096	C/B	1.04 / 0.76	1.01 / 0.71	0.68 / 0.67	1.31 / 1.03	1.26 / 0.96	0.93 / 0.92
8,192	C/B	0.98 / 0.69	0.97 / 0.66	0.66 / 0.65	1.24 / 0.95	1.22 / 0.92	0.91 / 0.9
16,384	C/B	0.96 / 0.66	0.95 / 0.65	0.64 / 0.64	1.21 / 0.92	1.20 / 0.9	0.89 / 0.89

Result: Protected messages, given a shared key



After spring break: How to generate and distribute keys