Lecture 5: MACs for long messages

Posted on piazza.com

- Lab 3: due on Monday 2/11 at 11pm
- Week 3 reading
- Lab 2 answers

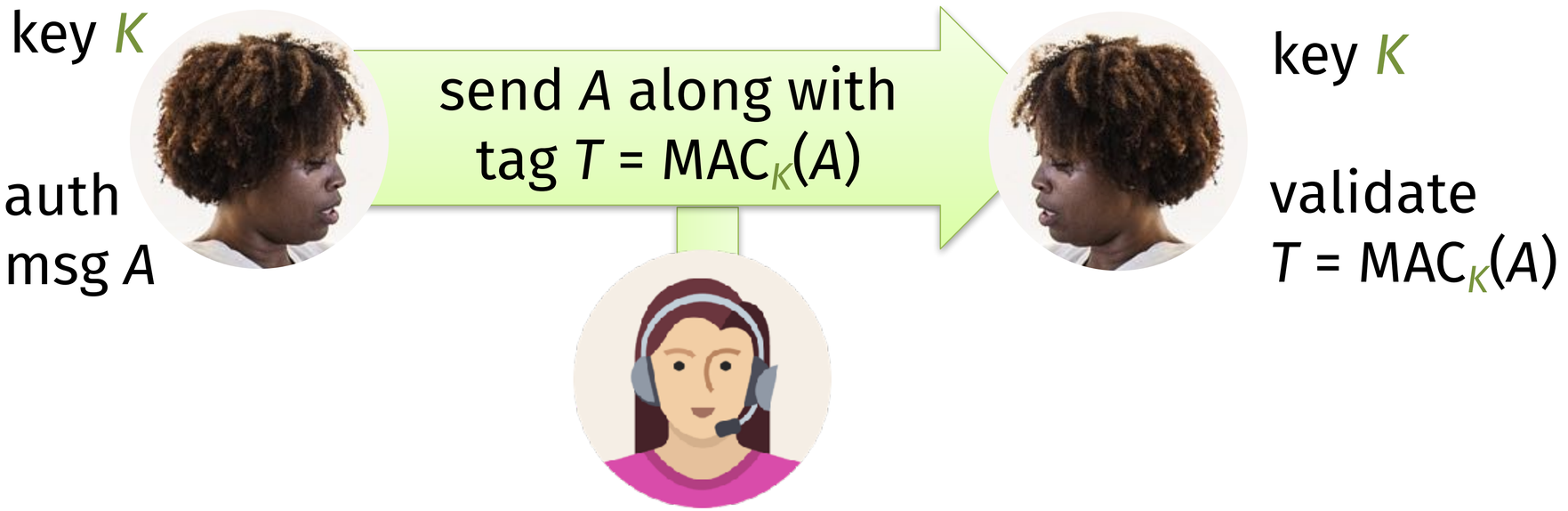
Random functions of different lengths

We will explore the design of random-looking functions R: $\{0,1\}^{\texttt{in}} \rightarrow \{0,1\}^{\texttt{out}}$

Case:	in < out	in = out (permutation)	in > out	in = ∞
One *public* function	Stream cipher ←	Codebook	→ Compression function	→ Hash function / random oracle
Family of functions indexed by *private* key		↓ Block cipher	→ Message auth code	↗

"If *A* hasn't explicitly queried *R* on some point x, then the value of *R*(x) is completely random… at least as far as *A* is concerned." —*Katz & Lindell*

Message authenticity



Objective of actively malicious Mallory:
inject a new message and tag (A, T)
or tamper with an existing one

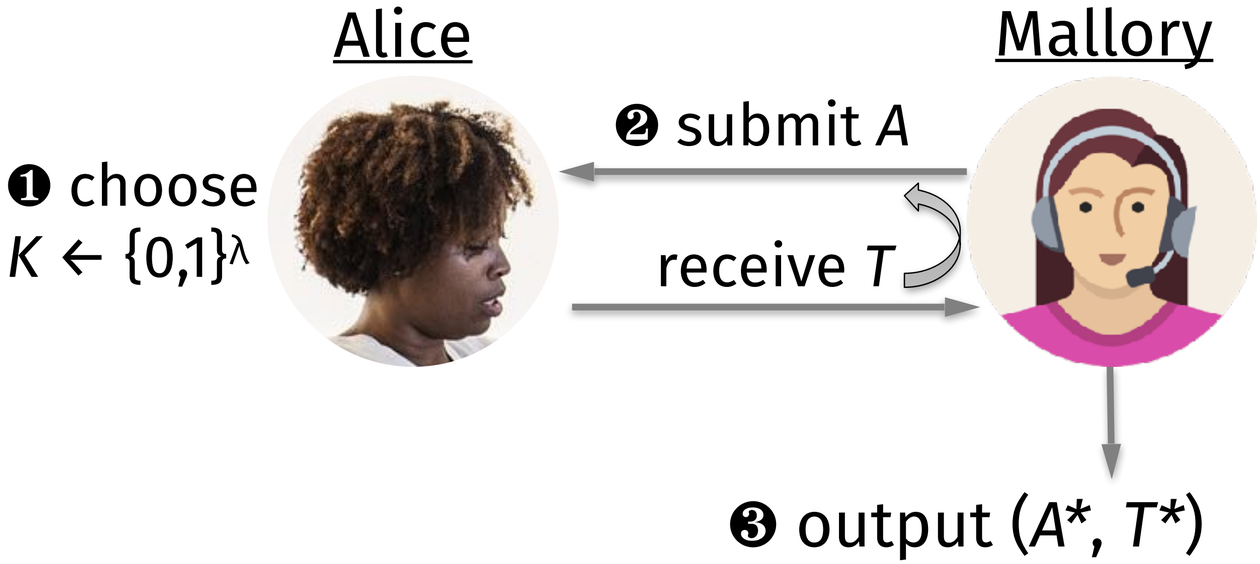
Definition: Message authentication code

Algorithms

- **KeyGen:** choose key $K \leftarrow \{0,1\}^{\lambda}$
- $\mathbf{MAC}_K(A \in \{0,1\}^{\alpha}) \rightarrow$ tag $T \in \{0,1\}^{\tau}$
  - Can be randomized
  - But usually deterministic
  - Prefer short tags: $\tau < \alpha$
- $\mathbf{Verify}_K(A, T \in \{0,1\}^{\tau}) \rightarrow yes/no$

Security game

Even after viewing many (A, T) pairs,
Mallory cannot ***forge*** a new one

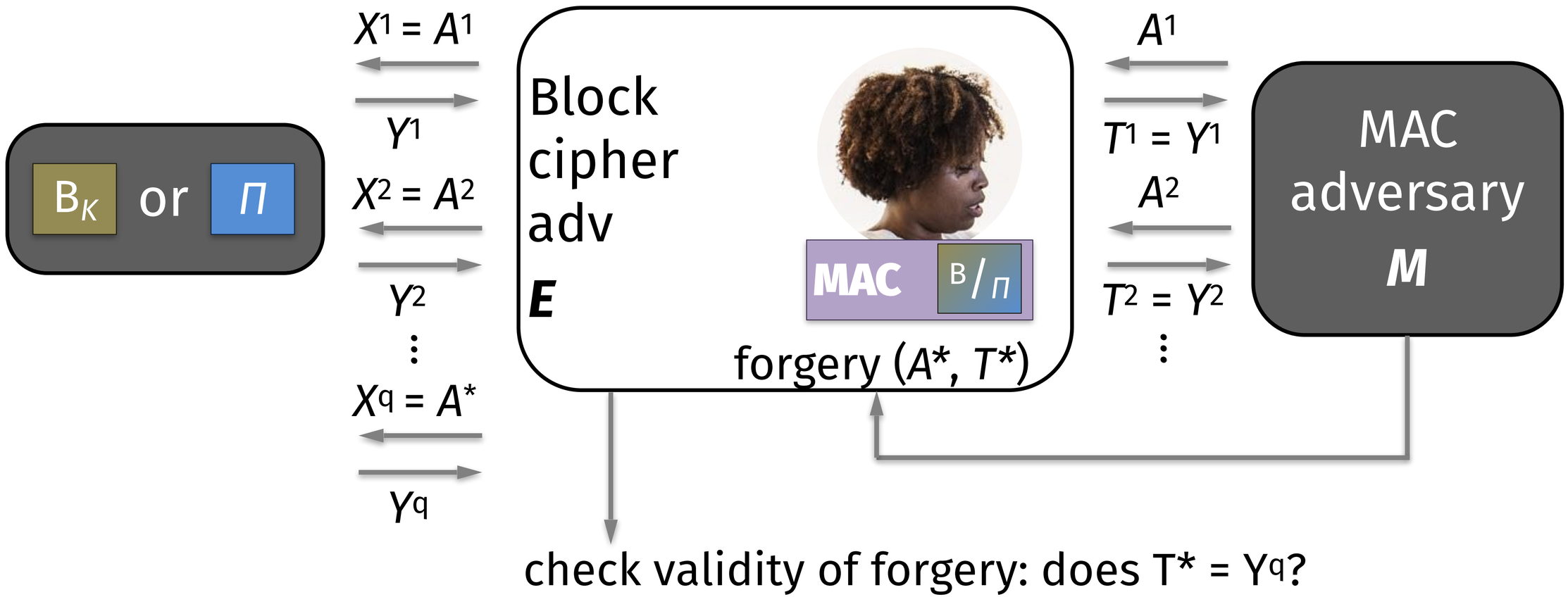


Mallory wins if this is a valid forgery, and it's new

Block cipher → MAC reduction

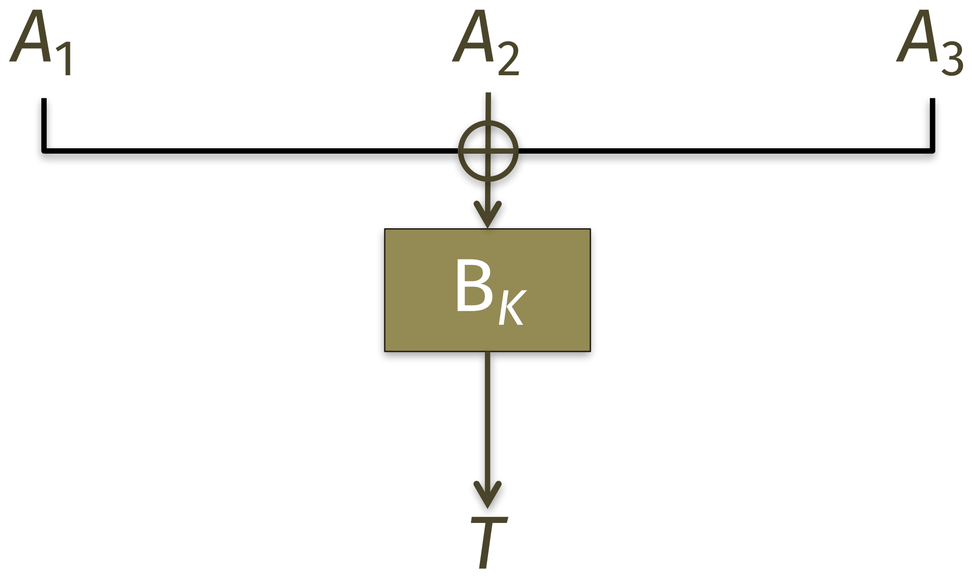
If we restrict $|A| = |T|$ = block length of a block cipher, simply applying the block cipher suffices!

$$\mathrm{MAC}_K(A) = \mathrm{B}_K(A)$$



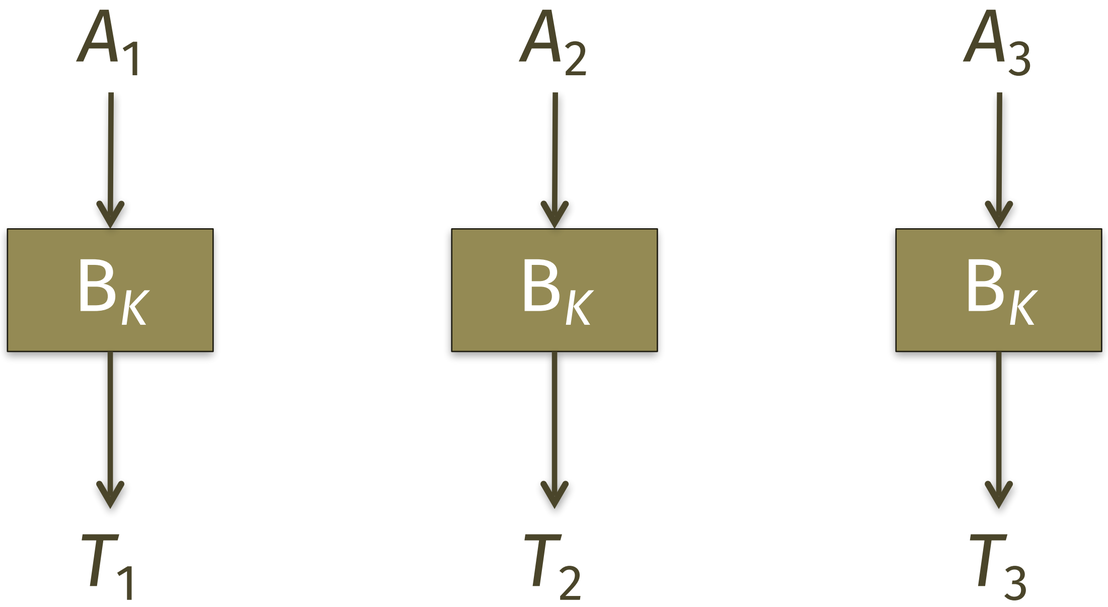
Variable length MACs?

Extensions that fail (even with 1 query!)	*How to produce a forged message*
1. XOR all message blocks together, authenticate the result	Find another message with same XOR



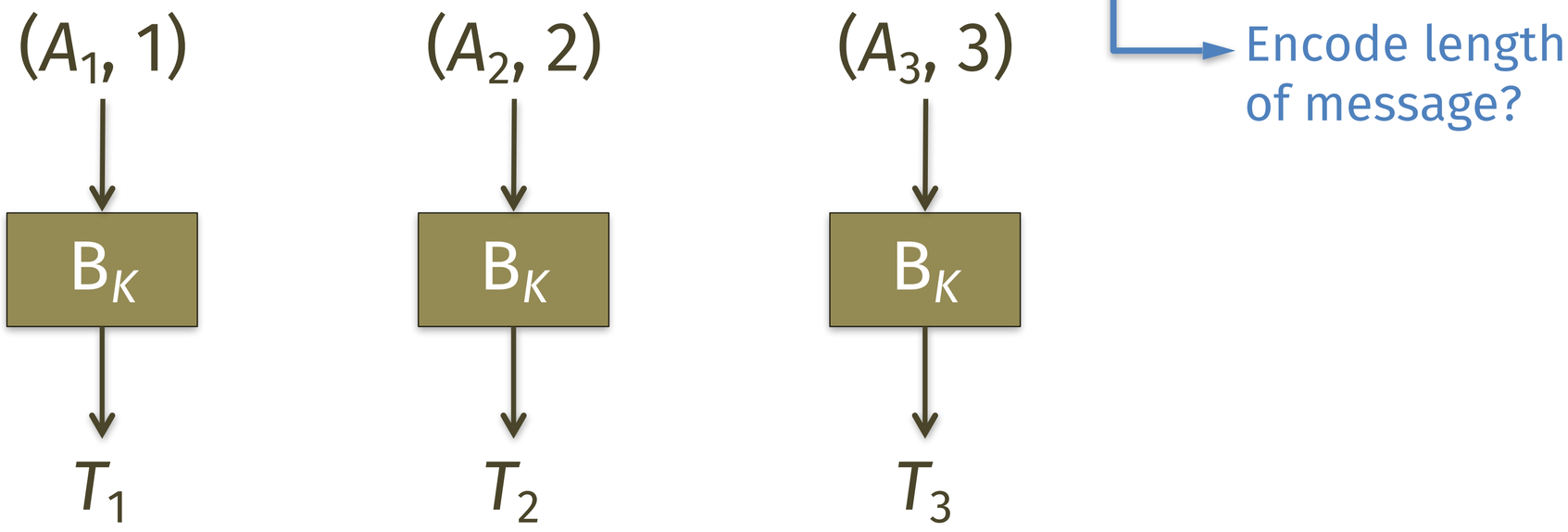
Variable length MACs?

Extensions that fail (even with 1 query!)	*How to produce a forged message*
1. XOR all message blocks together, authenticate the result	Find another message with same XOR
2. Auth each block separately	Change order of blocks



Variable length MACs?

Extensions that fail (even with 1 query!)	*How to produce a forged message*
1. XOR all message blocks together, authenticate the result	Find another message with same XOR
2. Auth each block separately	Change order of blocks
3. Auth each block along with sequence #	Drop blocks from the end of the message

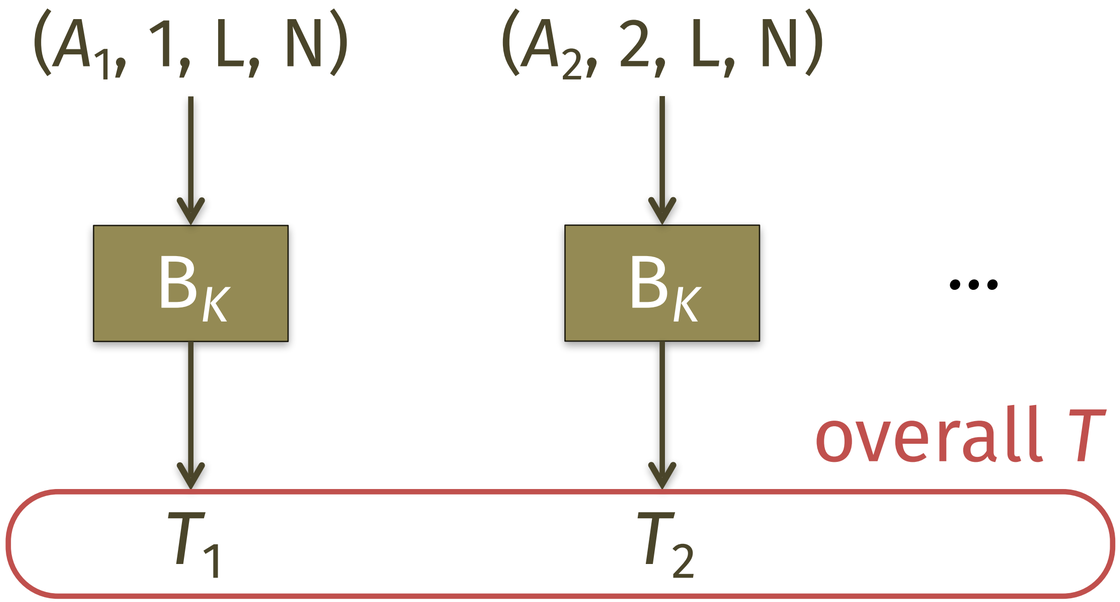


A construction that works

Four inputs per block:

- A_S = part of the message (using ¼ block length at a time)
- S = this block's sequence number
- L = length of overall message
- N = nonce chosen for this message

Thm. If B_K is (t, ε)-pseudorandom, then this construction yields a MAC that is (t, ε')-EU-CMA for ε' negligibly close to ε.



Terrible performance

- *Bad throughput:* invoke B_K four times as much as minimally necessary
- *Long tag:* want tag length τ == security parameter λ, indep of msg length α

We can do better!

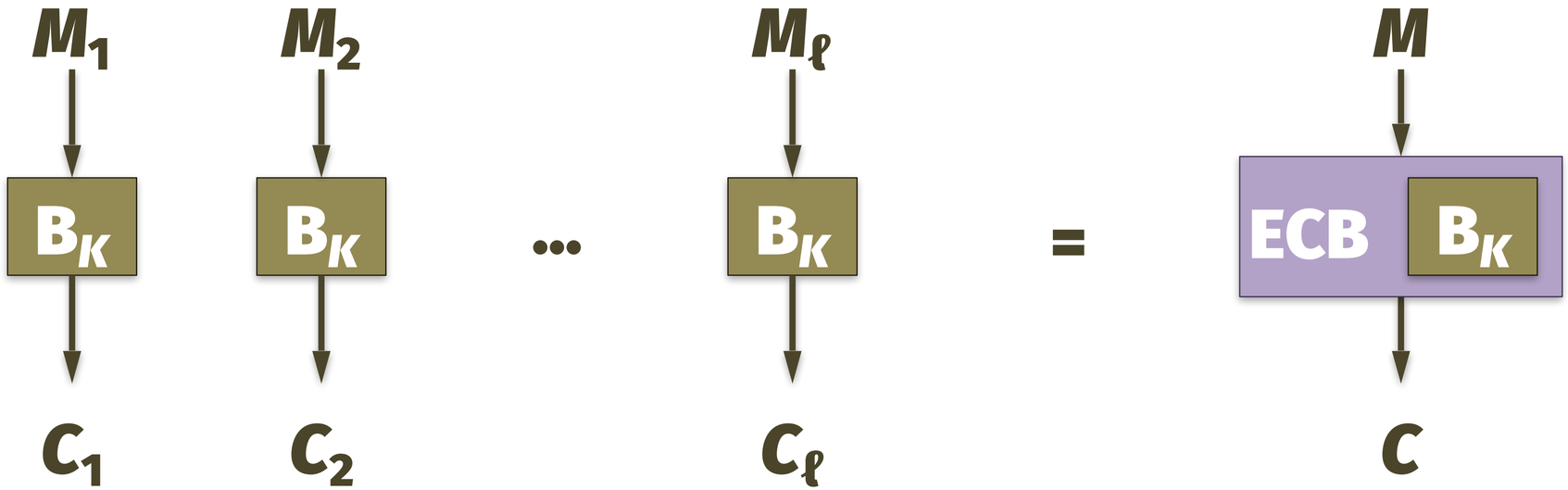
- New objective: find better constructions of MACs from block ciphers
- Insist that $\tau = 1$ block in length, at most
- Security-space tradeoff
  - Can truncate the tag to $l < \tau$ bits in length, if desired
  - Ideally, the MAC still requires 2^l effort to forge

Supporting longer messages

Def. A *mode of operation* connects multiple calls to a block cipher (with one key K)

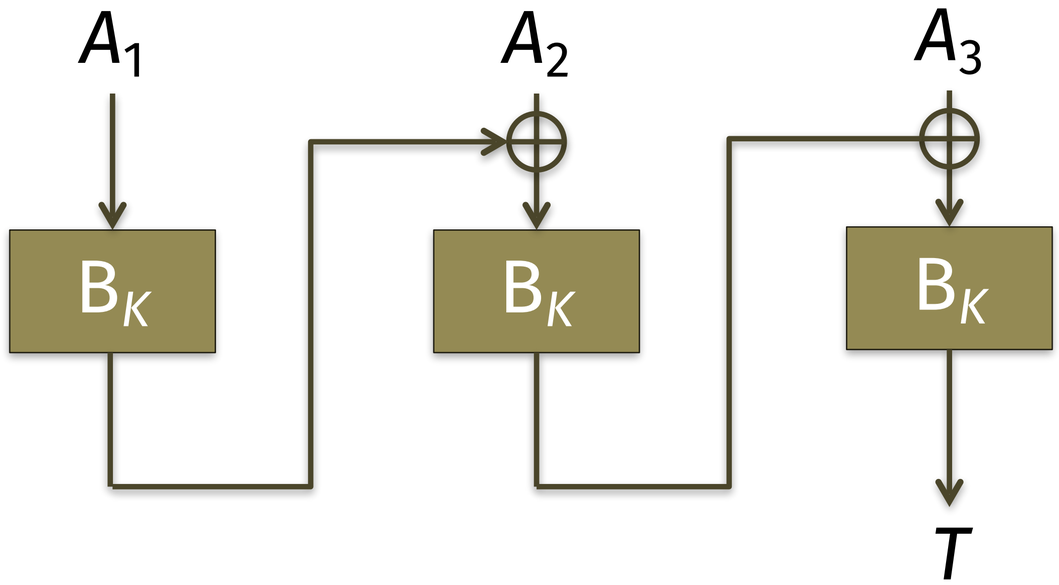
One simple mode: process each block of the message independently

This is called *Electronic Codebook (ECB)* mode



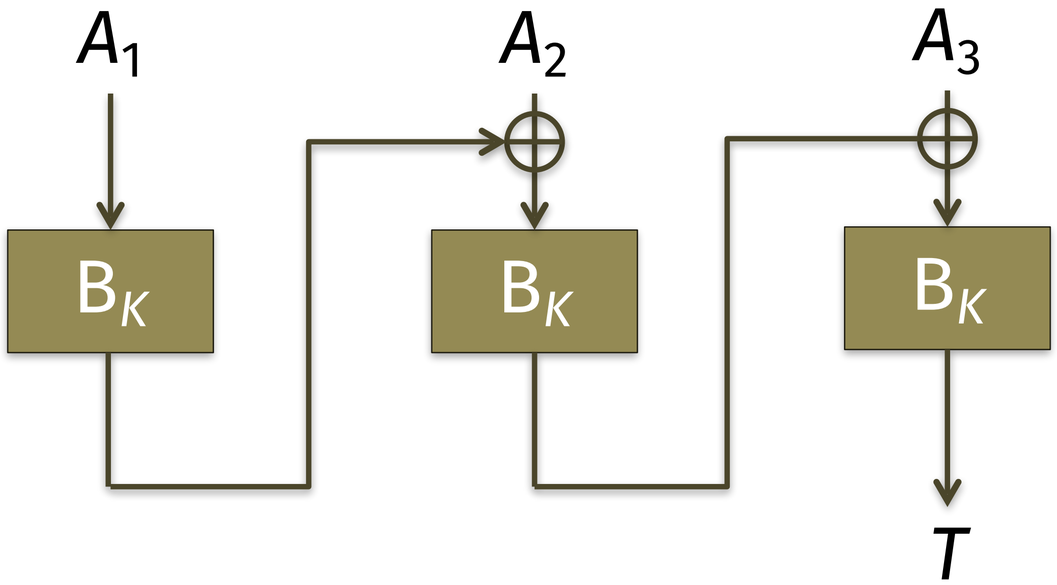
CBC-MAC: cipher block chaining

- 1st block simply runs the underlying block cipher (just like ECB mode)
- Subsequent inputs to the block cipher depend on both new input + prior output!
- Only the final block tag is revealed ⇒ important for performance *and* security



CBC-MAC: cipher block chaining

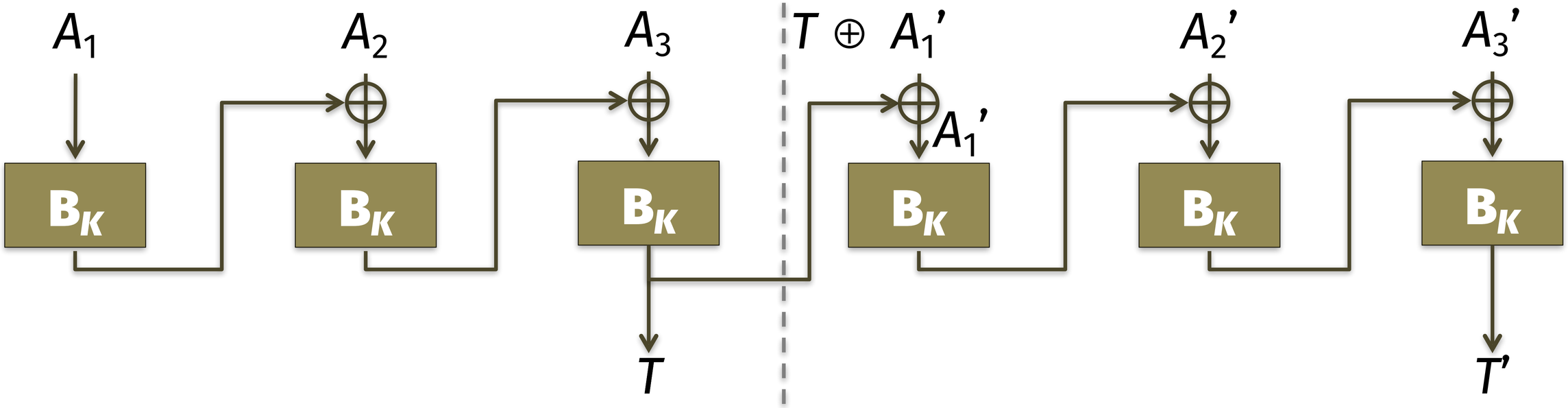
Thm. If B_K is pseudorandom, then CBC-MAC B_K is an EU-CMA MAC
…for any *pre-specified* fixed length that is a *multiple* of the block length



CBC-MAC: cipher block chaining

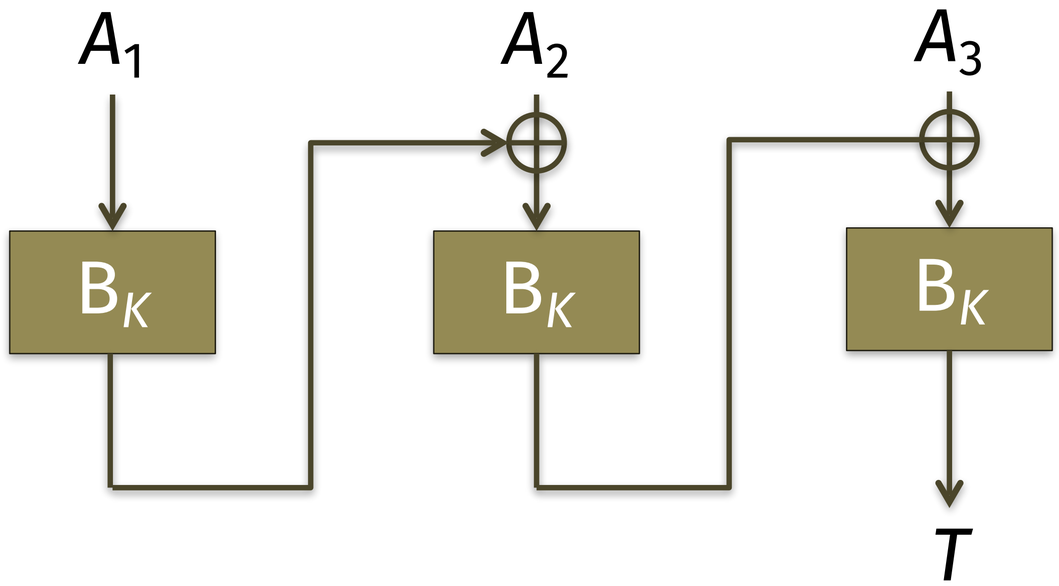
Thm. If B_K is pseudorandom, then CBC-MAC B_K is an EU-CMA MAC
…for any *pre-specified* fixed length that is a *multiple* of the block length

CBC-MAC **insecure** if recipient doesn't know length in advance!
(Padding won't help)



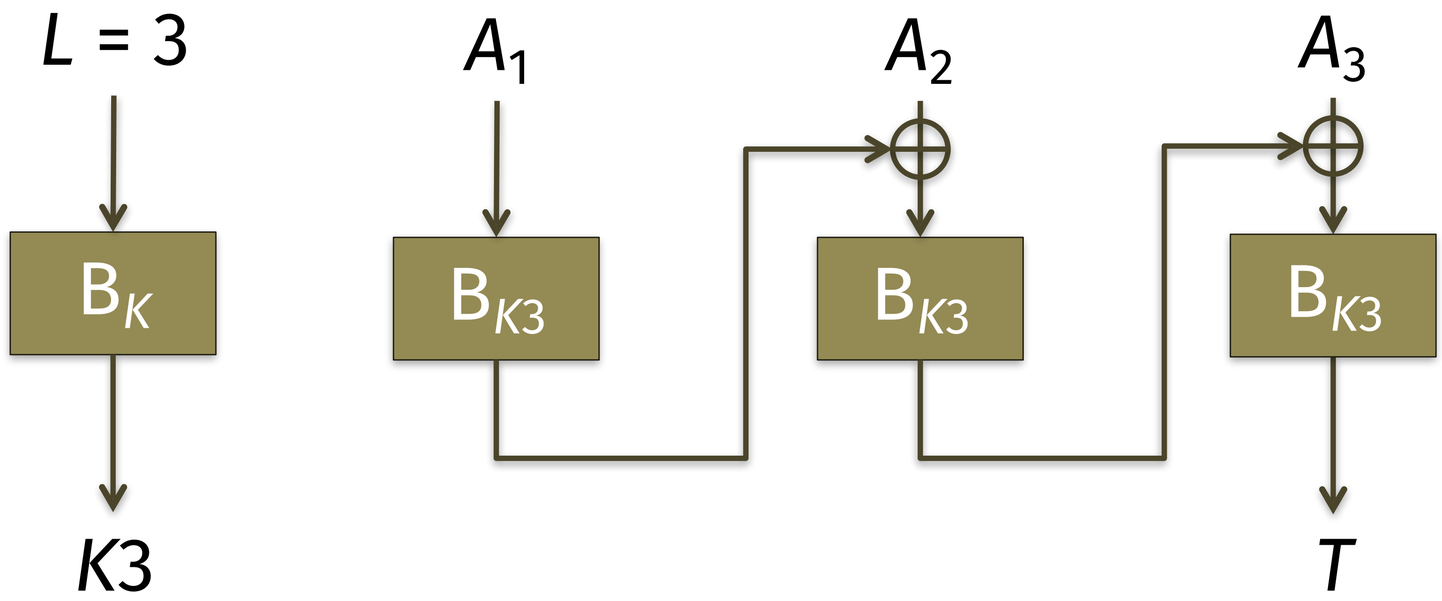
How to fix CBC-MAC?

Proposed fix:	How to implement:	Comment:



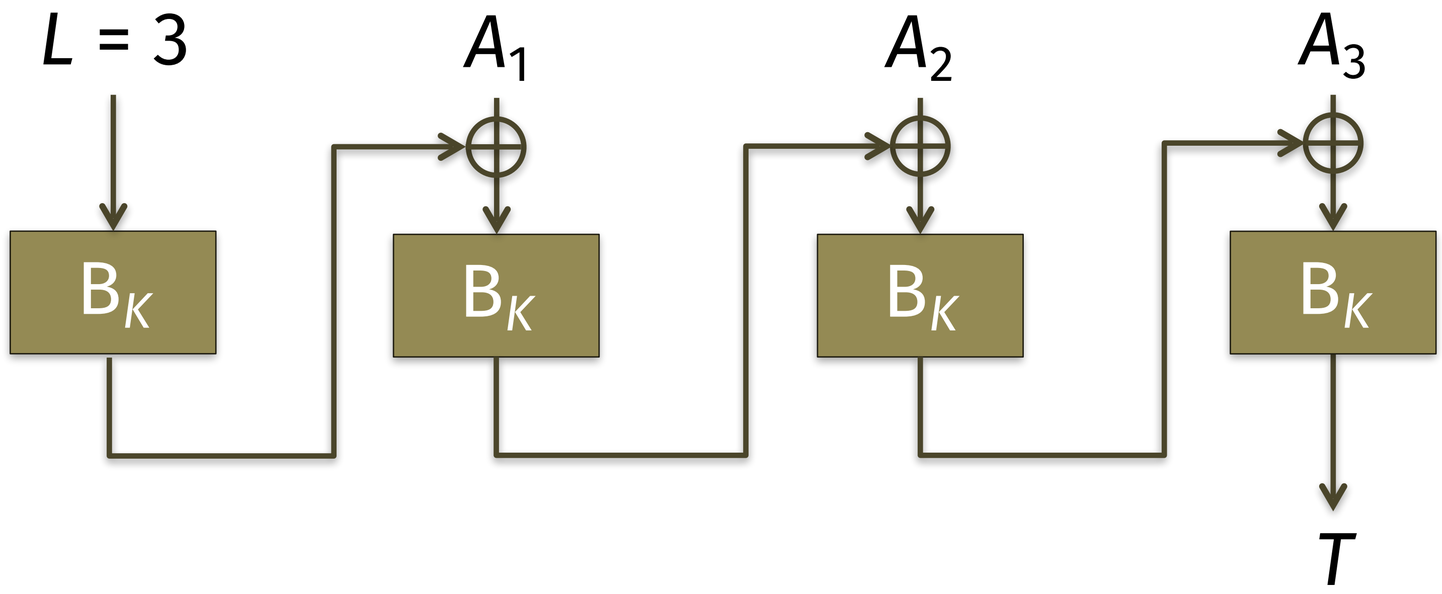
How to fix CBC-MAC?

Proposed fix:	How to implement:	Comment:
1. One key per length	Let $K_L \leftarrow B_K(L)$ be MAC key for msgs of length L	Poor key agility



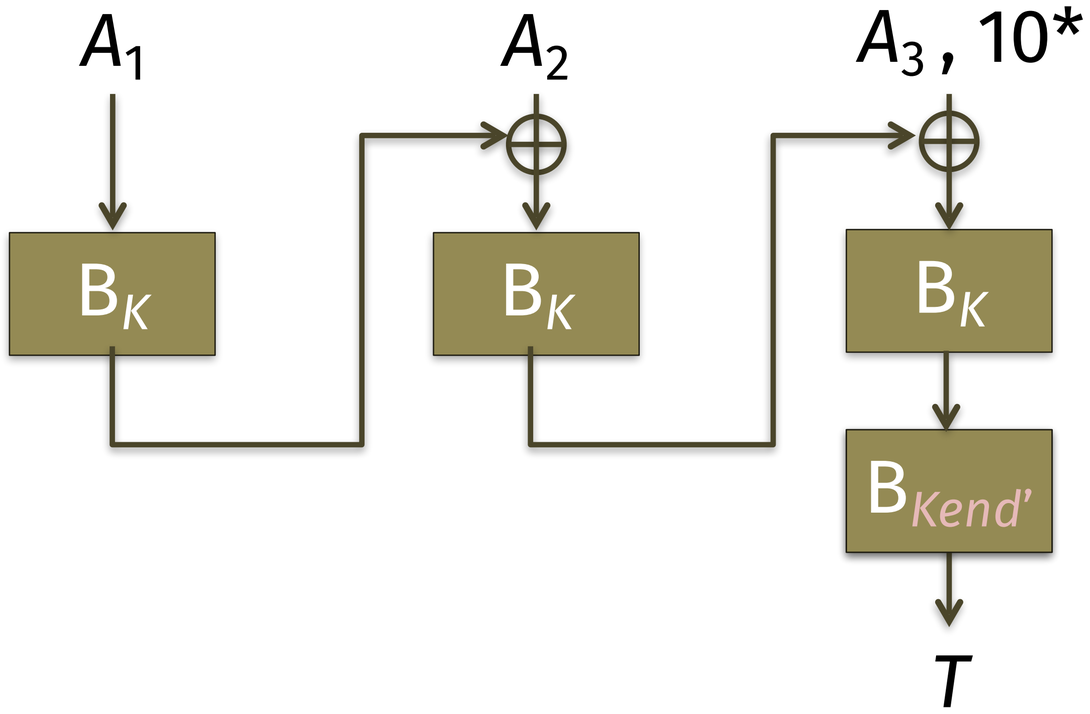
How to fix CBC-MAC?

Proposed fix:	How to implement:	Comment:
1. One key per length	Let $K_L \leftarrow B_K(L)$ be MAC key for msgs of length L	Poor key agility
2. Prepend length	Prepend L to the message (appending won't work!)	Difficult to handle streaming data

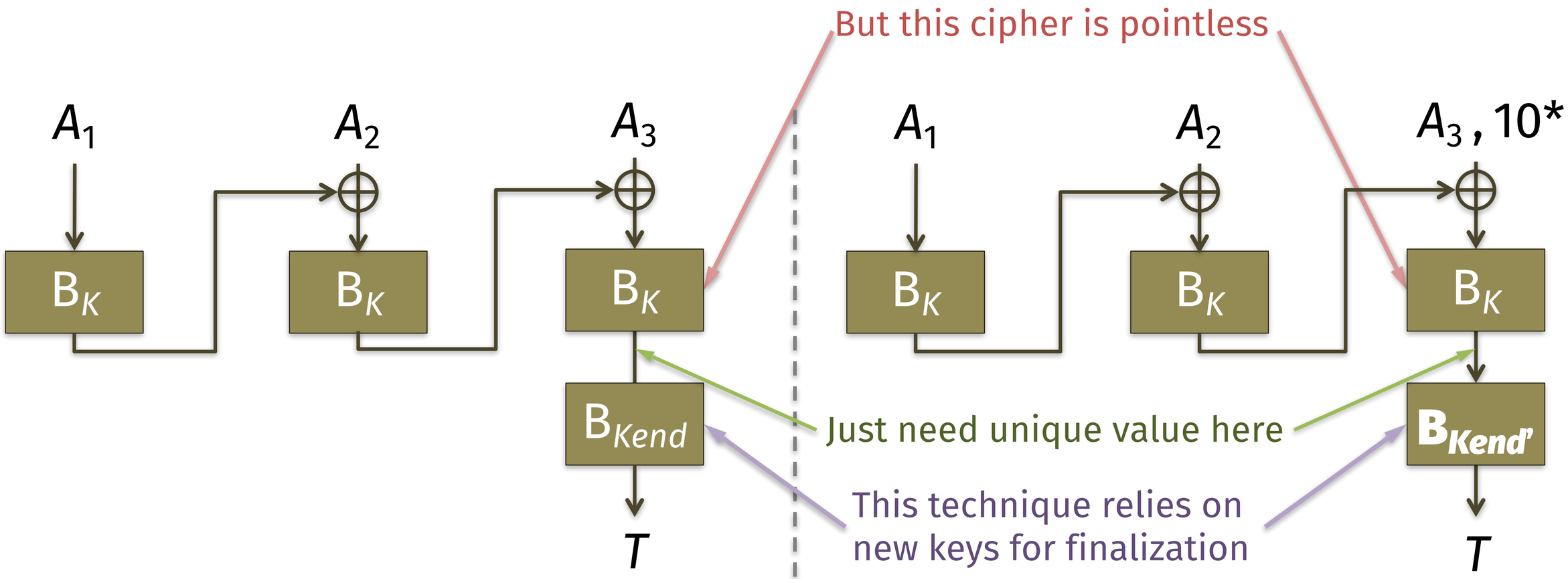


How to fix CBC-MAC?

Proposed fix:	How to implement:	Comment:
1. One key per length	Let $K_L \leftarrow B_K(L)$ be MAC key for msgs of length L	Poor key agility
2. Prepend length	Prepend L to the message (appending won't work!)	Difficult to handle streaming data
3. Finalize last block	Perform another crypto operation at the end	Preferred! Don't need L in advance

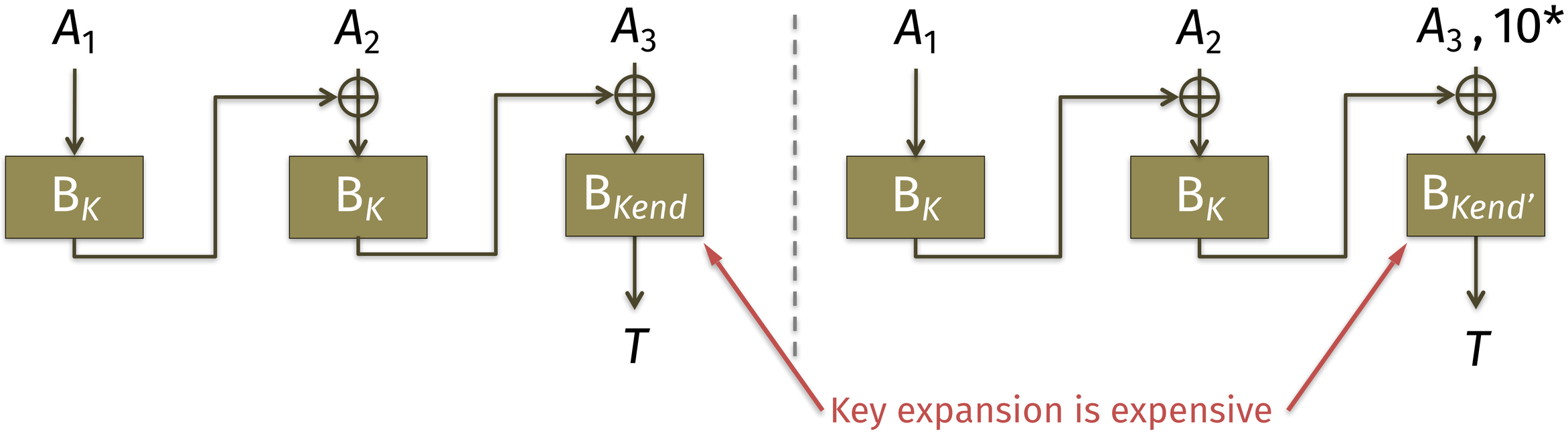


EMAC



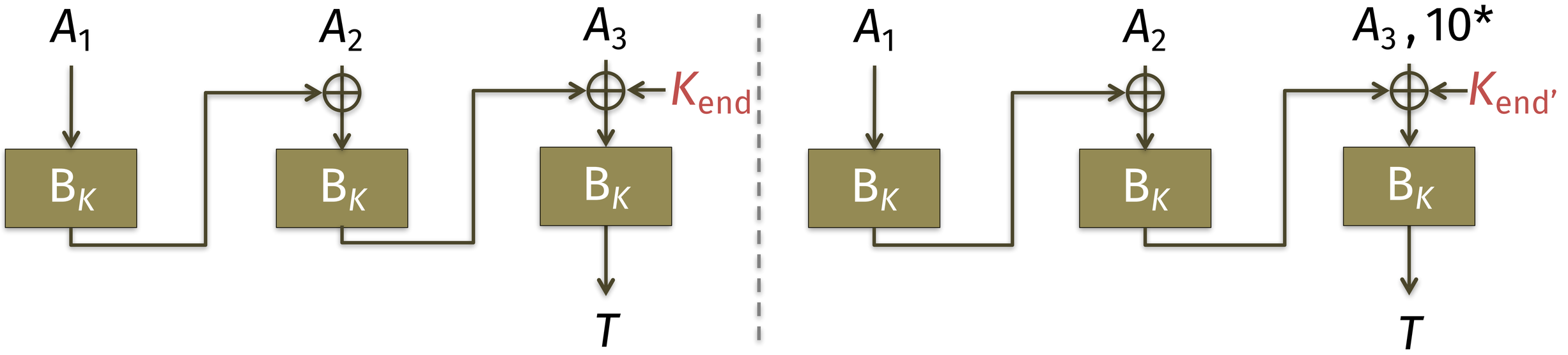
Formally: encipher the result of a target collision resistant function

FMAC



Save one block cipher call

Cipher-based MAC (CMAC, aka XMAC)



- Don't use extra keys to encrypt. Instead use them to influence the final block.
- Designed by Black and Rogaway, 2000

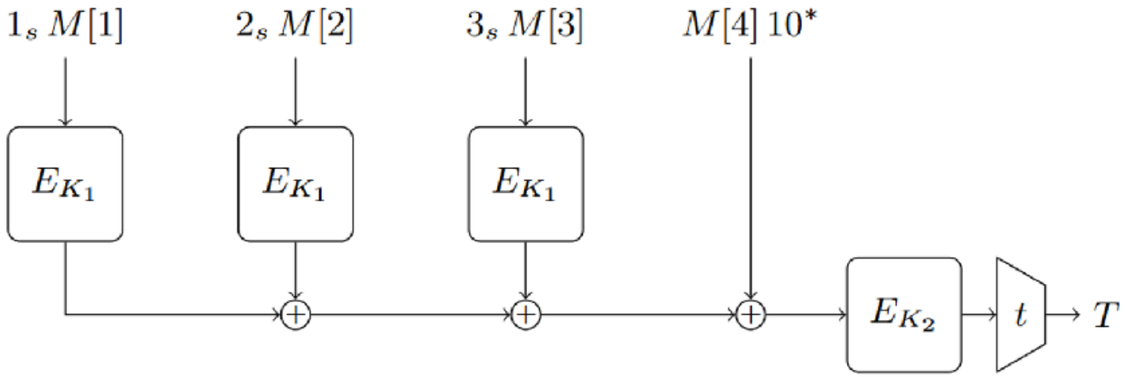
One-key CBC-MAC (OMAC)

- Designed by Iwata & Kurosawa 2003
- Save on key length by deriving the finalization keys from the original

Alternative MAC constructions

XOR-based MACs

- Parallelizable!
- Best in lightweight environments
- Resolve the problem from the start of lecture via lots of sweat
- Example: PMAC, in OCB mode

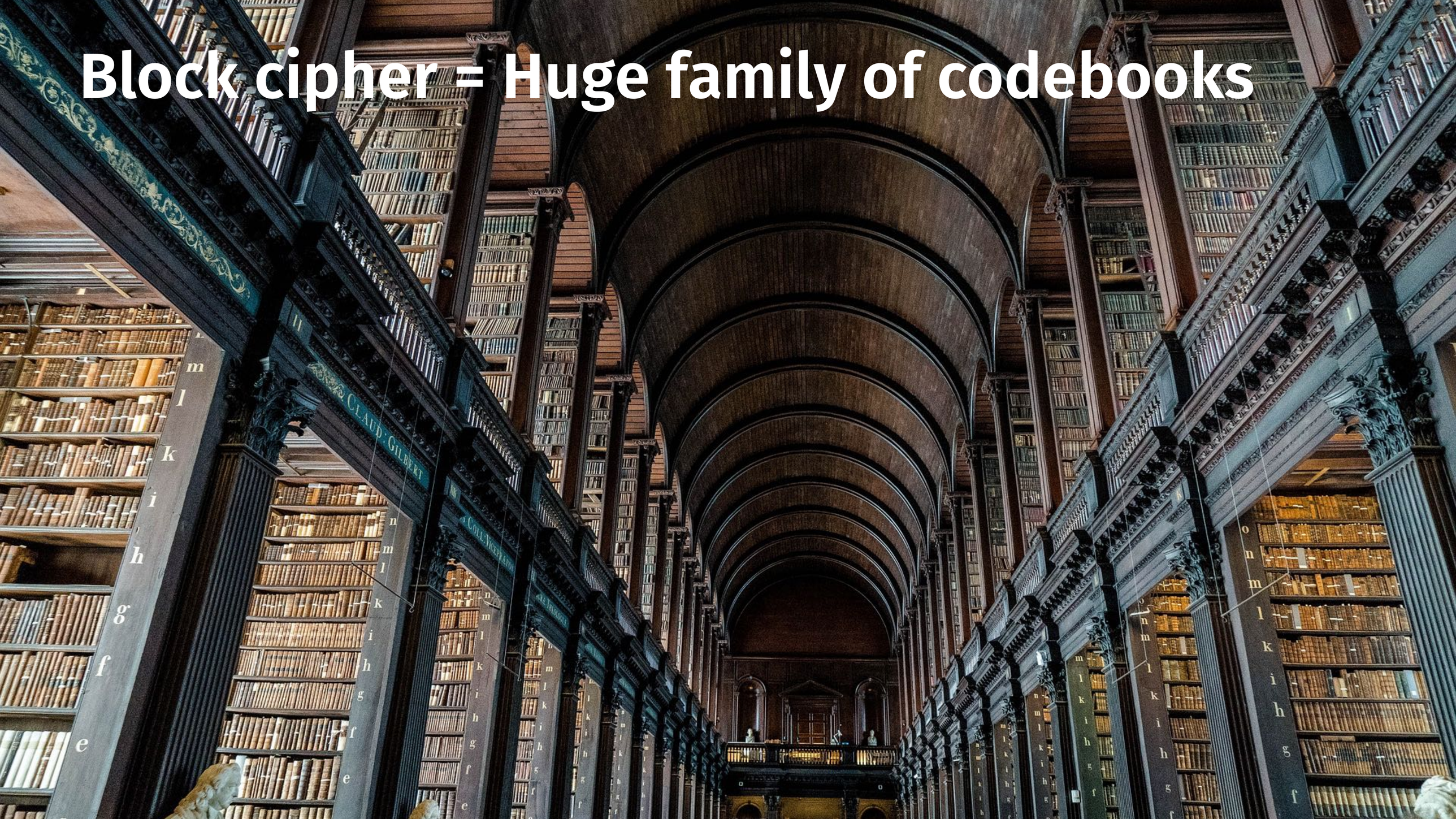


Hash-based MACs

- Probably the best standalone MAC to use today
- Requires a new primitive…

$$\text{HMAC}(K, M) = H\left((K \oplus \text{opad} \parallel H\left(K \oplus \text{ipad}\right) \parallel M\right)$$

Block cipher = Huge family of codebooks

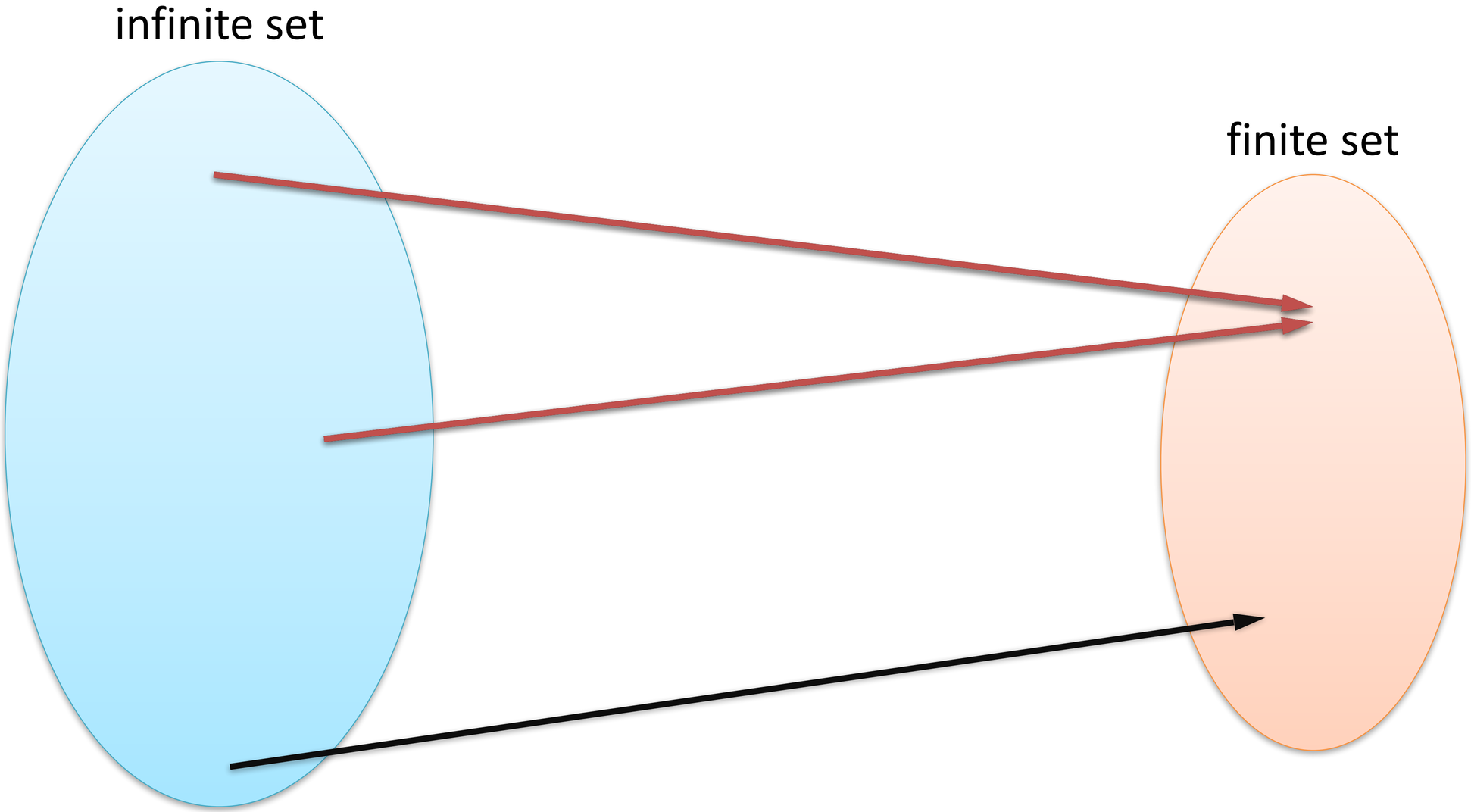


Hash function = 1 public codebook

- Hash function $H : \{0,1\}^{\infty} \rightarrow \{0,1\}^{out}$
- Compresses long messages into short digests
  - Cannot invert!
- We have already seen one in the labs: SHA-256

X	*Y*
aba	nr
abs	mb
ace	yd
act	wv
add	je
ado	hg
aft	uv
age	zm
ago	ds
aha	ae
aid	kf
⋮	⋮
zip	cy
zoo	dx

Hash function: length-reducing → collisions exist



Bellare Rogaway 1993: “Random oracles are practical”

Two step process:

1. Pretend we have random oracle *R*, produce crypto protocol P^R that uses it
2. Replace *R* with an “appropriately chosen” function *H*, and hope that it works

We argue that the random oracle model -- where all parties have access to a public random oracle -- provides a bridge between cryptographic theory and cryptographic practice. In the paradigm we suggest, a practical protocol P is produced by devising and proving correct a protocol P^R for the random oracle model and then replacing oracle accesses by the computation of an “appropriately chosen” function H. This paradigm yields protocols much more efficient than standard ones while retaining many of the advantages of provable security.

So… what is an "appropriately chosen" *H*?

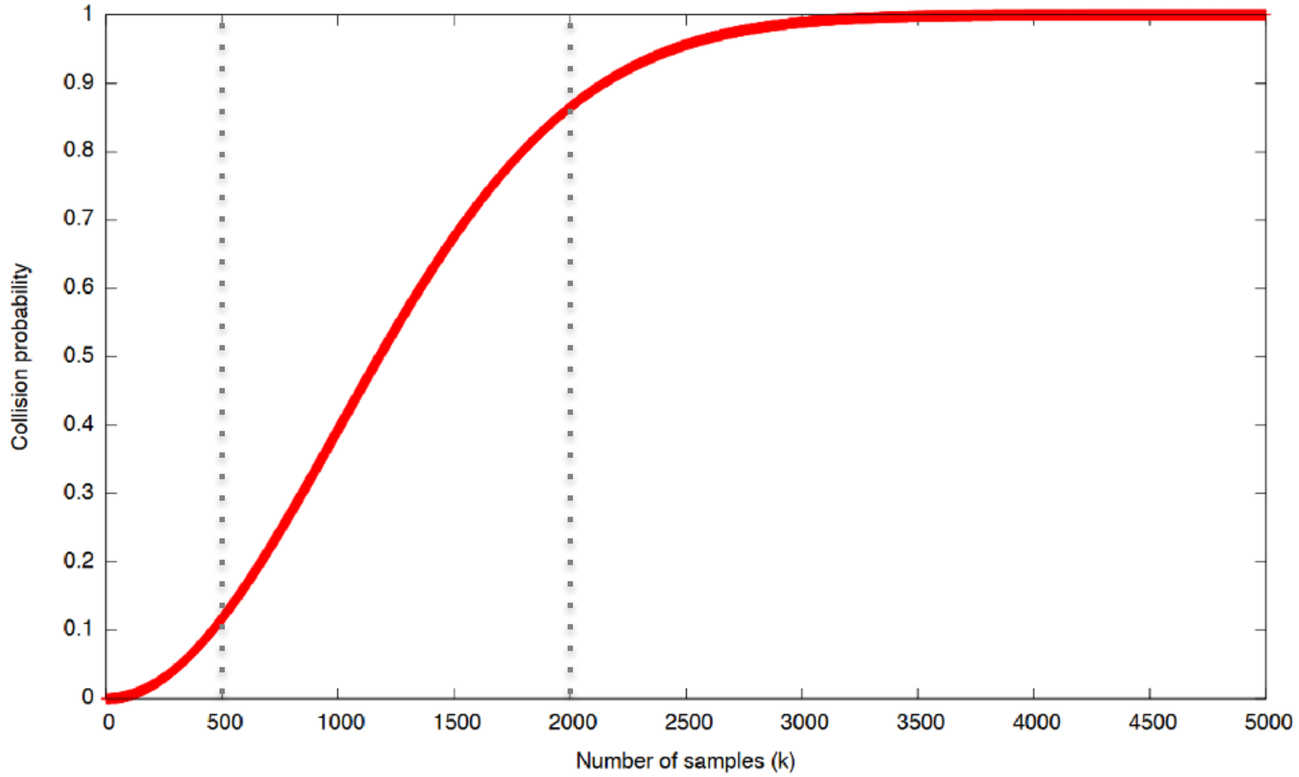
Def. A *hash function H*: $\{0,1\}^* \rightarrow \{0,1\}^\eta$ is an efficiently-computable function that accepts unbounded input and outputs strings of a fixed length η

Security notions against adversaries who *possess the code* of *H*

stronger

- *Preimage resistance*: given $\boldsymbol{y} = \boldsymbol{H}(\boldsymbol{x})$ for a "randomly" chosen $\boldsymbol{x}$, difficult to find any preimage $\boldsymbol{x'}$ s.t. $\boldsymbol{H}(\boldsymbol{x'}) = \boldsymbol{y}$
- *2nd preimage resistance*: given "randomly" chosen $\boldsymbol{x}$, difficult to find another $\boldsymbol{x'}$ s.t. $\boldsymbol{H}(\boldsymbol{x'}) = \boldsymbol{H}(\boldsymbol{x'})$ noticeably faster than an exhaustive search of 2^η values
- *Target collision resistance*: same as CR, except $\boldsymbol{x}$ chosen before $\boldsymbol{H}$ is known
- *Collision resistance*: given only $\boldsymbol{H}$, difficult to find two different inputs $\boldsymbol{x}$ and $\boldsymbol{x'}$ s.t. $\boldsymbol{H}(\boldsymbol{x}) = \boldsymbol{H}(\boldsymbol{x'})$ noticeably faster than a birthday bound search of $2^{\eta/2}$ values

Birthday bound



- When drawing with replacement from set of size L,

$$E[\#\text{ items to draw until first collision}] \approx \sqrt{\frac{\pi}{2}L} \approx 1.25\sqrt{L}$$

- The distribution of M is tightly concentrated around its expected value

Popular families of hash functions

Ron Rivest's Message Digest family

- 1991 MD5 (128 bits) is most well-known, still sadly in use today despite being completely broken.
- Some of the earlier iterations never reached the public domain, and all have mostly been discarded
- MD6 was a submission to the SHA-3 competition. Its security analysis originally had problems, so the authors recommended that NIST not continue it.

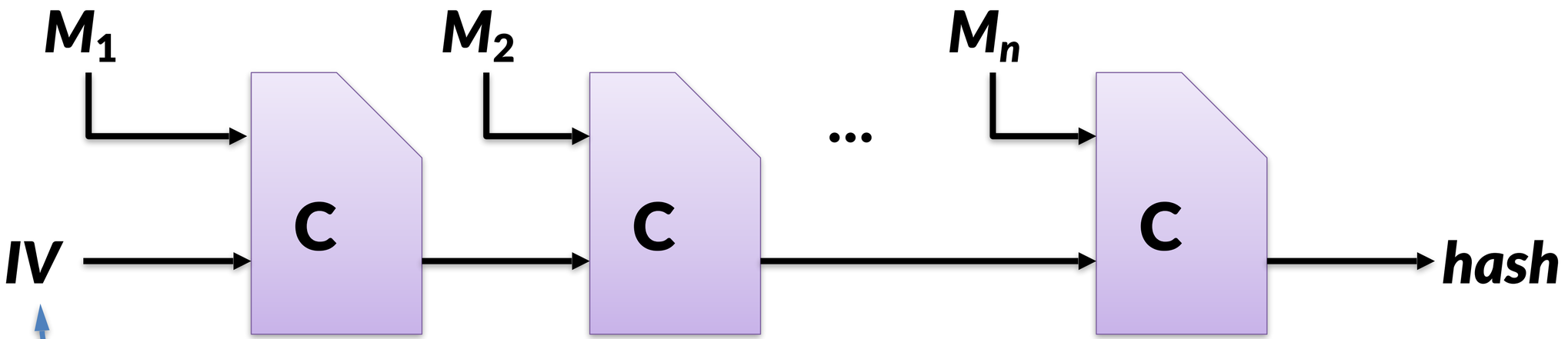
Secure Hash Algorithm (SHA) family

- NSA-designed, NIST-approved
- SHA-0: retracted before standardization
- 1995: SHA-1 (160 bits) is still in use today, though slowly fading
  - Wang, Yin, Yu 04: showed algorithm for 2^{69} step collision
  - Stevens et al 17: found collision in 2^{63} steps
- 2001: SHA-2 family (224, 256, 384, or 512 bits) is the recommended hash function to use today
- All follow a Merkle-Damgard design
- (2015: SHA-3 algorithm has different design)

Merkle-Damgård paradigm

Can build a variable-length input hash function from two primitives:

1. A fixed-length, *compressing* random-looking function
2. A mode of operation that iterates this function multiple times in a smart manner

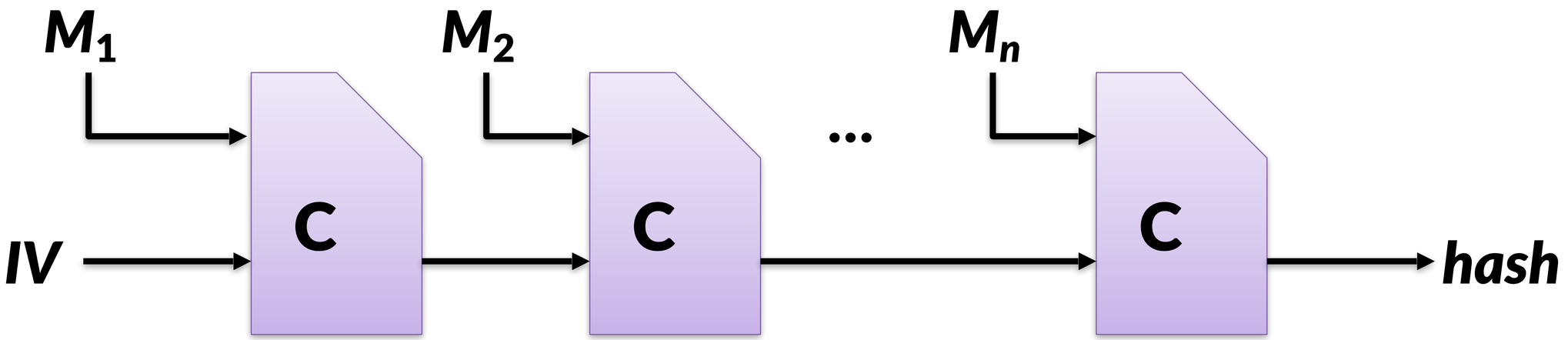


IV for hash function is typically fixed in spec, not user configurable.

Concern with Merkle-Damgård design

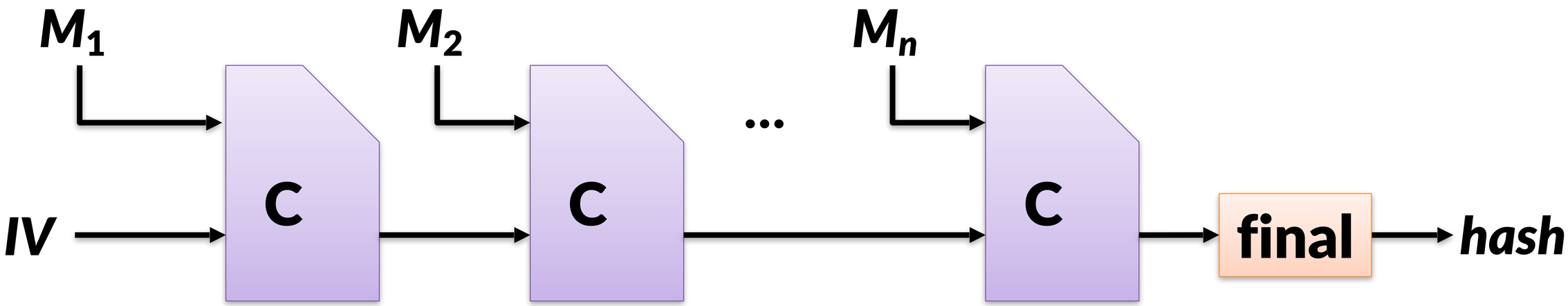
"The vulnerability of [a hash function] construction is due to its finite state, [a limitation that does] not apply to a random oracle."

–Keccak team, *Cryptographic sponge functions*



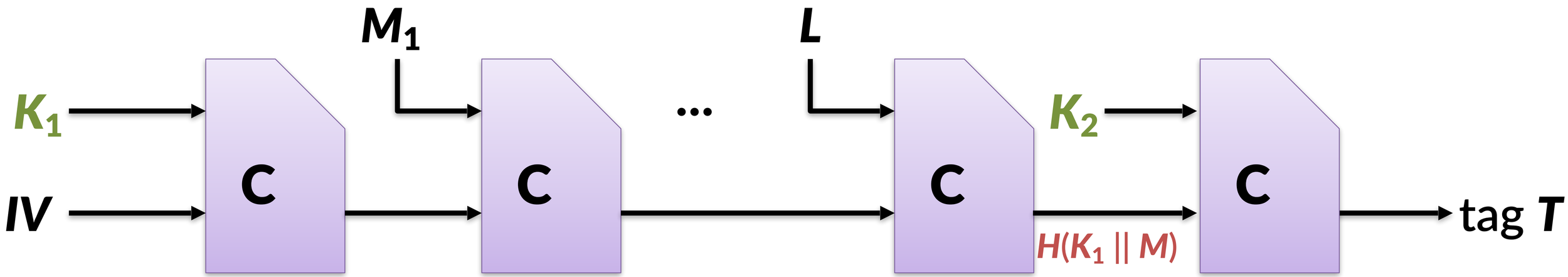
Length extension attack

Countermeasure: finalization



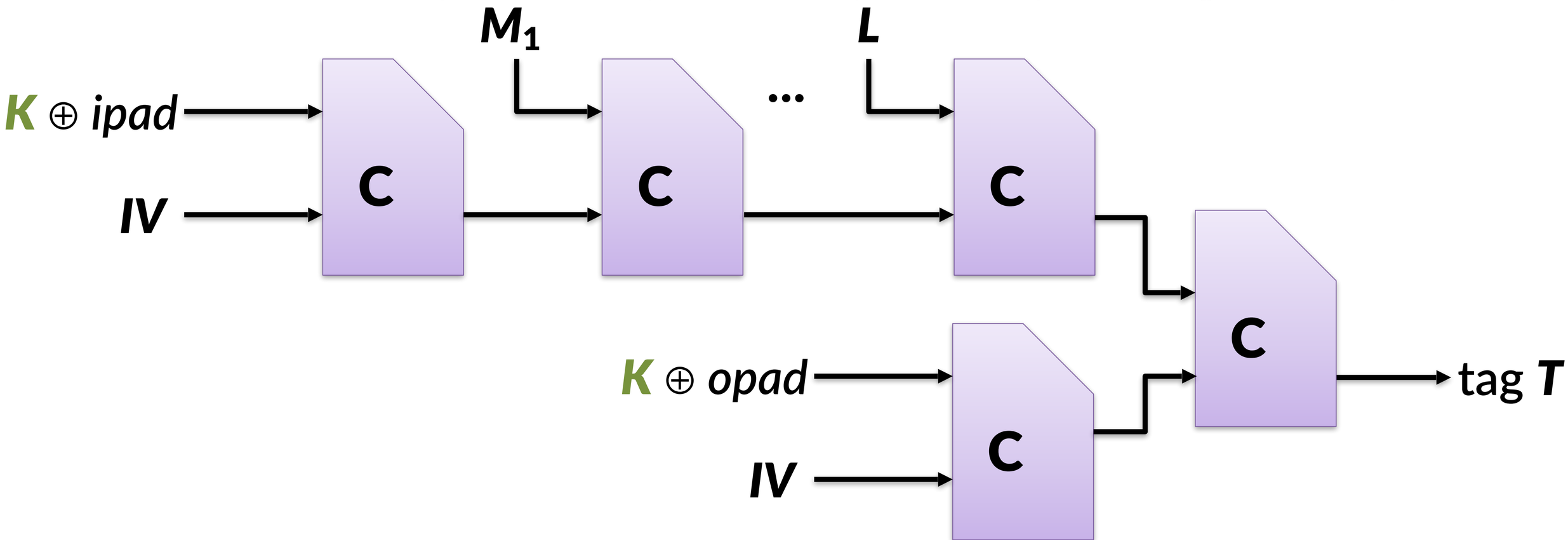
Hash function → MAC

- NMAC: let's use C one more time as a type of finalization
- In essence: $H(K_1 \,||\, M)$ pares M down to a size that K_2 can process



HMAC [Bellare Canetti Krawczyk 97]

- Recall from CMAC → OMAC story: nobody wants to carry multiple keys
- HMAC: use *C* itself to derive two "independent" keys from one initial key
  - Uses fixed constants ipad = 0x5C, opad = 0x36 repeated to equal the length of the key



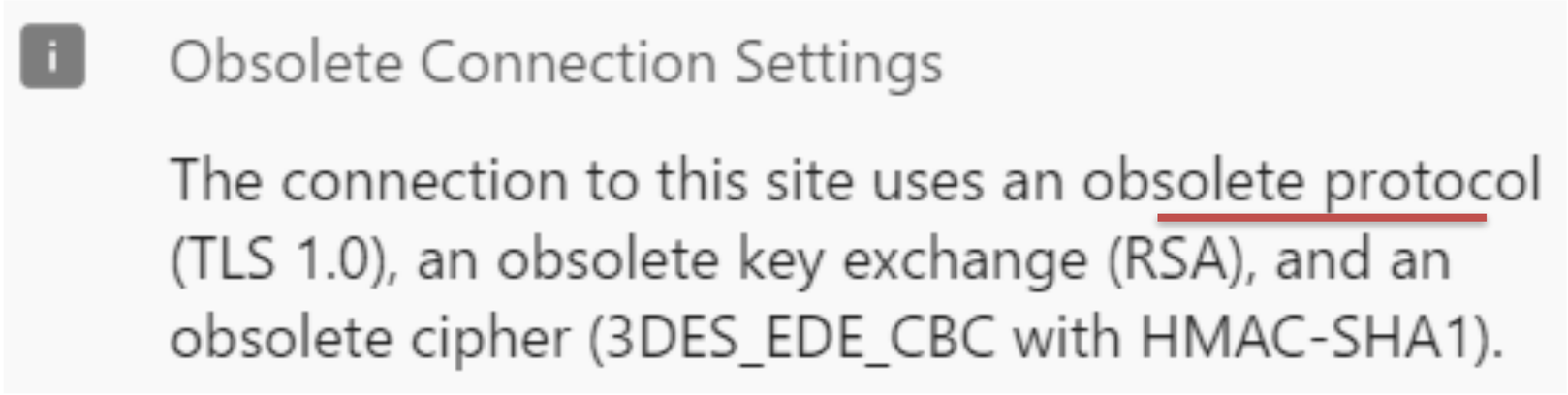
Strength of HMAC

Thm. HMAC is an EU-CMA MAC as long as:

1. The compression function C is pseudorandom
2. The Merkle-Damgard iteration mechanism is collision-resistant

Bellare (2005) removed condition #2, so HMAC applies even to hash functions like MD5 and SHA1 that are not collision resistant

https://www.bu.edu:



Random functions of different lengths

We will explore the design of random-looking functions R: $\{0,1\}^{\texttt{in}} \rightarrow \{0,1\}^{\texttt{out}}$

Case:	in < out	in = out (permutation)	in > out	in = ∞
One *public* function	Stream cipher ←	Codebook	→ Compression function	→ Hash function / random oracle
Family of functions indexed by *private* key		↓ Block cipher	→ Message auth code	↗

"If *A* hasn't explicitly queried *R* on some point x, then the value of *R*(x) is completely random… at least as far as *A* is concerned." —*Katz & Lindell*