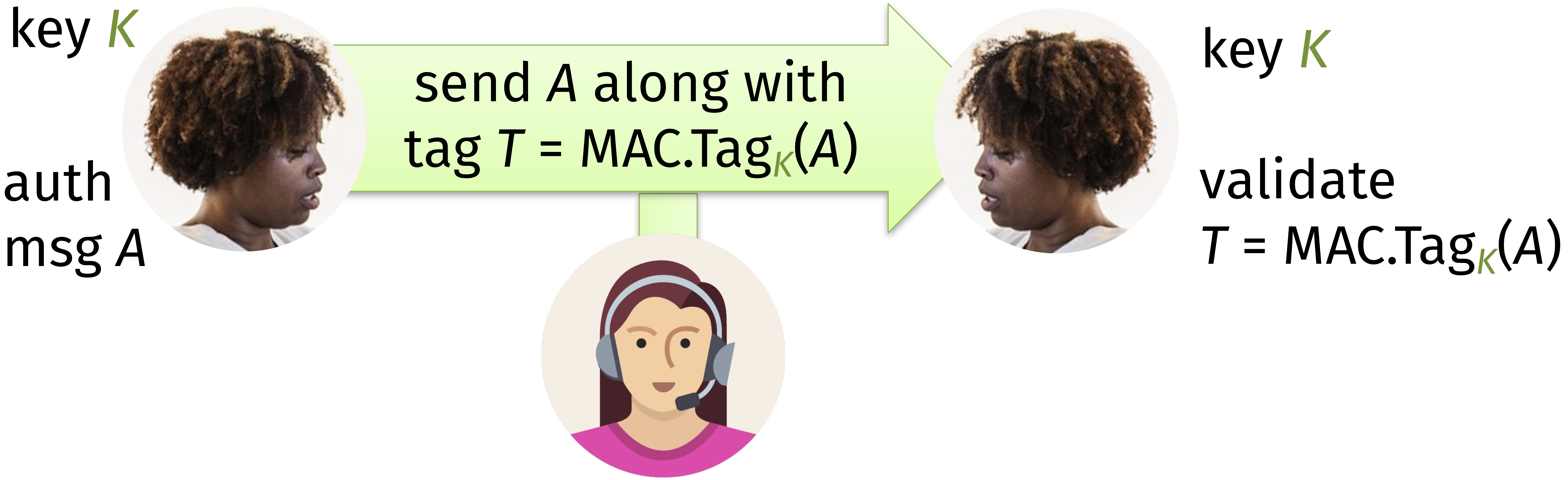
Lecture 10: Variable-length MACs

- Homework 5 has been posted on Gradescope, due Monday 3/2
- Required reading: portions of two textbooks
  - The Block Cipher Companion (section 4.4)
  - The Hash Function BLAKE (sections 2.1, 2.2, 2.4)

Message authentication code (MAC)



MACs stop an actively malicious Mallory from:
- injecting a new message and tag (A^*, T^*)
- tampering with an existing one

Definition: Message authentication code

Algorithms

- **KeyGen:** choose key $K \leftarrow \{0,1\}^\lambda$
- **Tag**$_K$ $(A \in \{0,1\}^\alpha) \rightarrow$ tag $T \in \{0,1\}^\tau$
  - Usually deterministic
  - Prefer short tags: $\tau < \alpha$
- **Verify**$_K$ $(A, T \in \{0,1\}^\tau) \rightarrow$ *yes* / *no*
  - Recompute T* = $\mathrm{MAC}_K(A)$ tag
  - Check if T* == T

Requirements

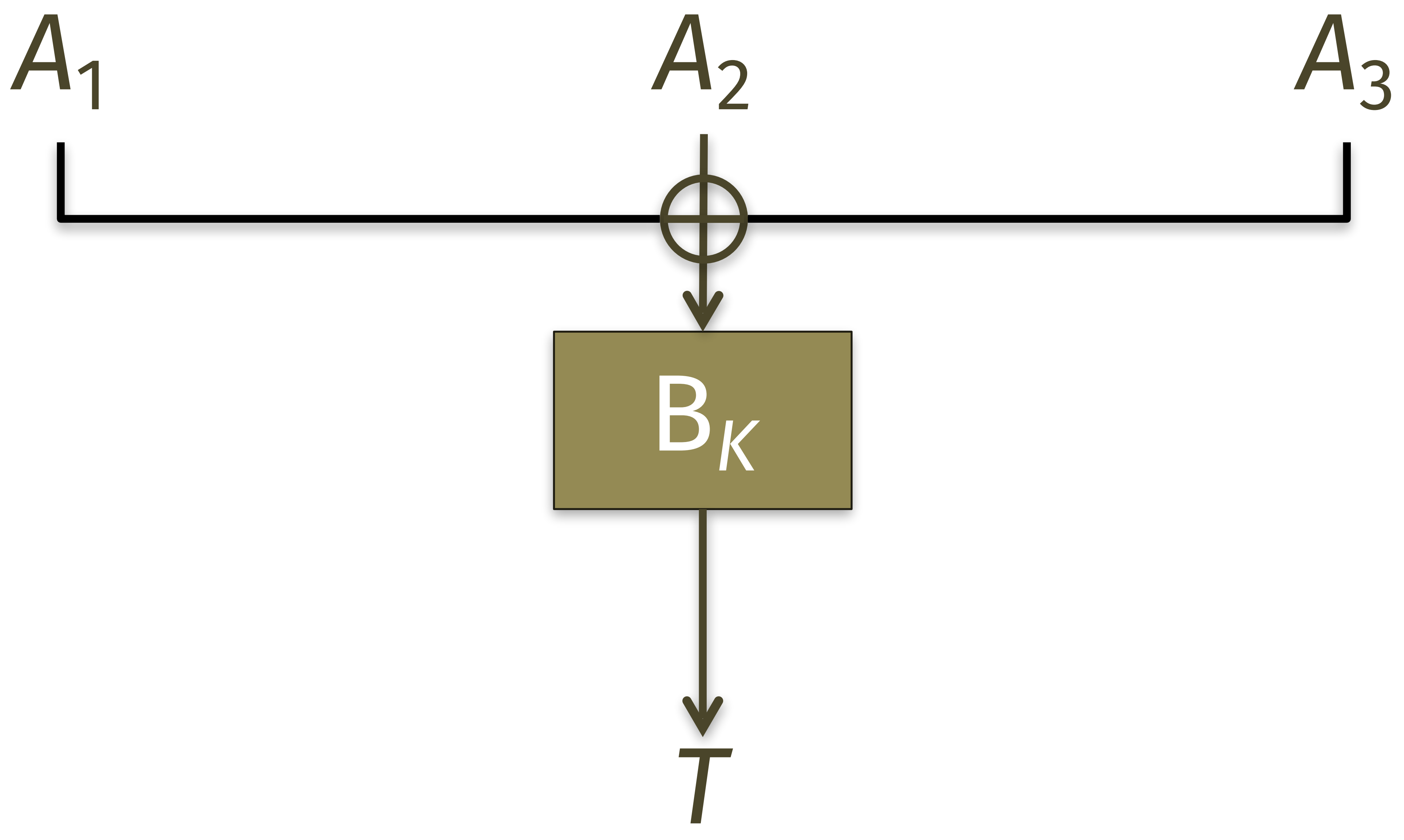
- **Performance:** Fast algorithms
- **Correctness:** For all K, tags made by MAC_K are accepted by Verify_K
- **EU-CMA:** Mallory cannot forge tags, with the restriction that she can't Verify tags produced by MAC

Recap: MAC for one-block messages

- For our first MAC, let's restrict $|A| = |T|$ = block length of a block cipher
- In this case, simply applying the block cipher suffices to build a MAC!
  - MAC.KeyGen runs BlockCipher.KeyGen to sample a key K
  - $\text{MAC.Tag}_K(A) = \text{BlockCipher.Encipher}_K(A)$
  - $\text{MAC.Verify}_K(A, T)$ re-computes the MAC tag and checks equality with T

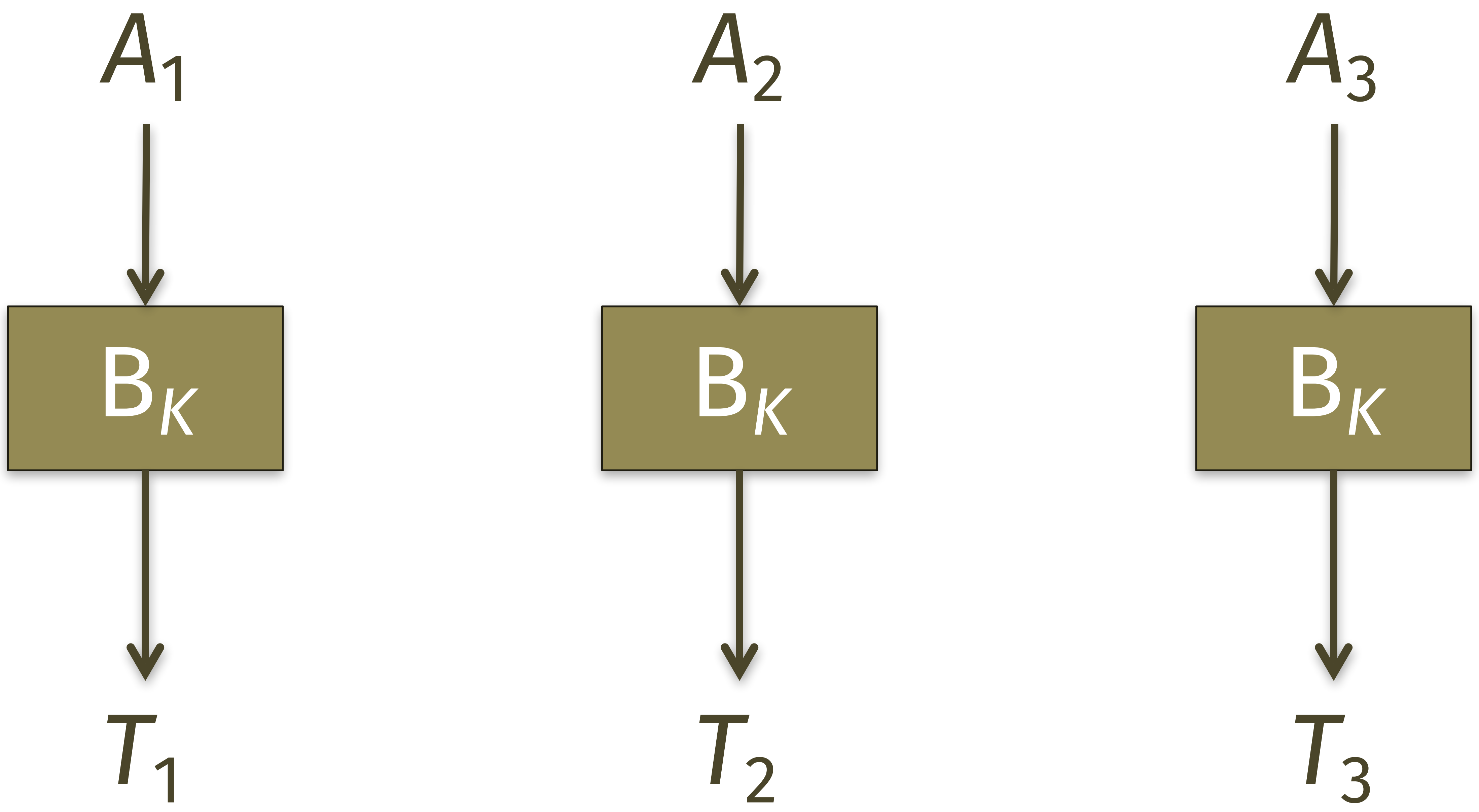
Variable length MACs?

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



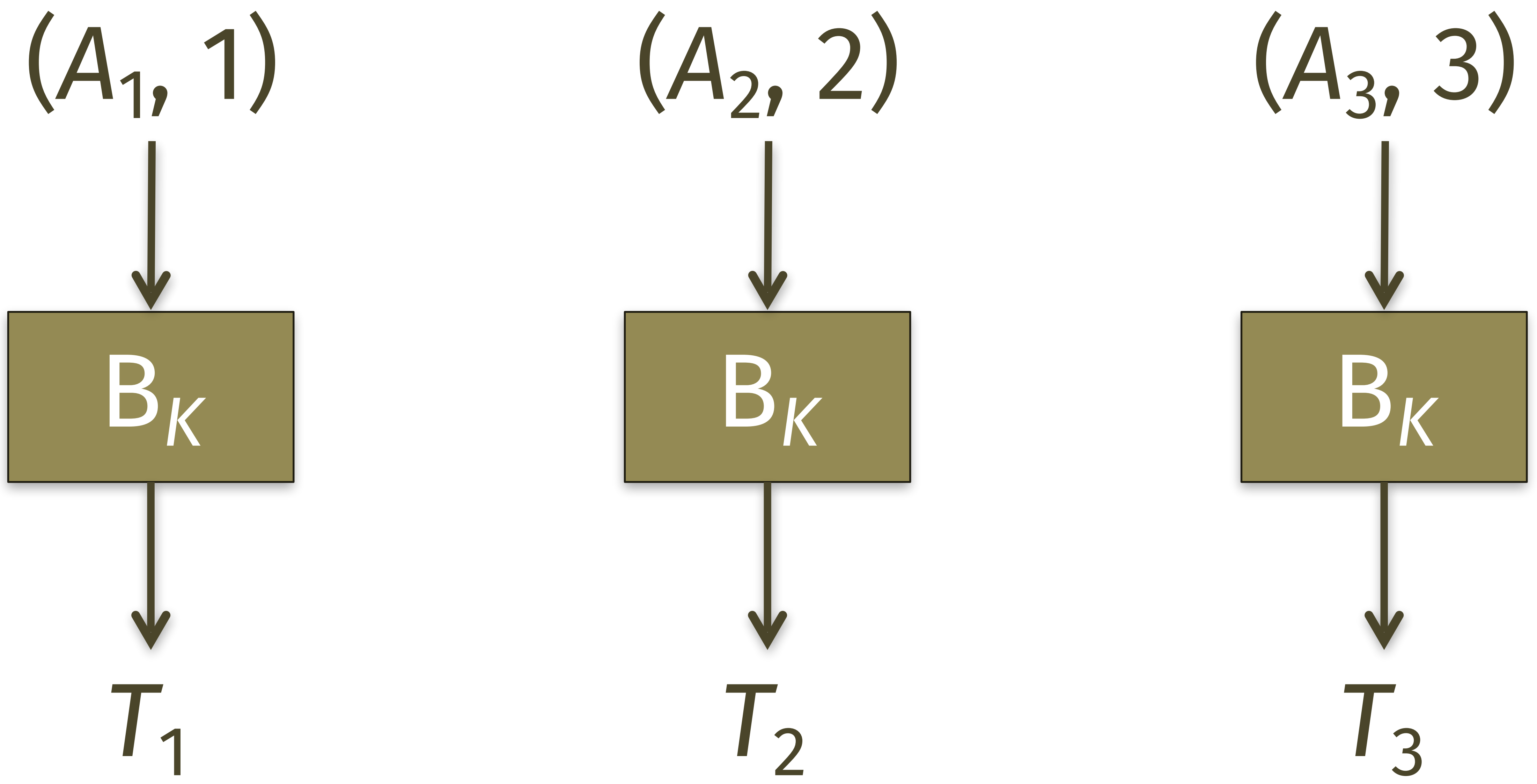
Variable length MACs?

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



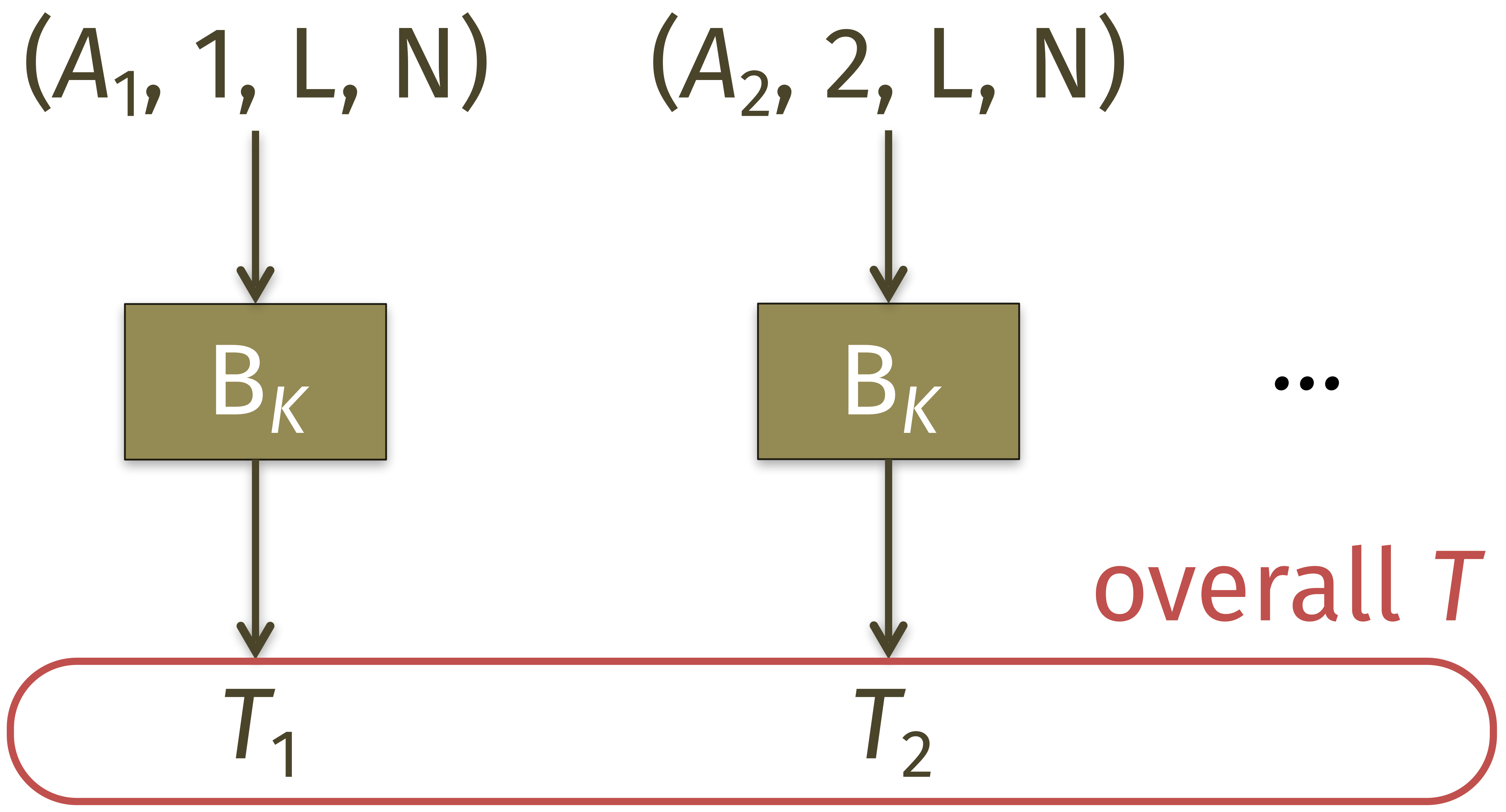
Encode length of message?

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 though…

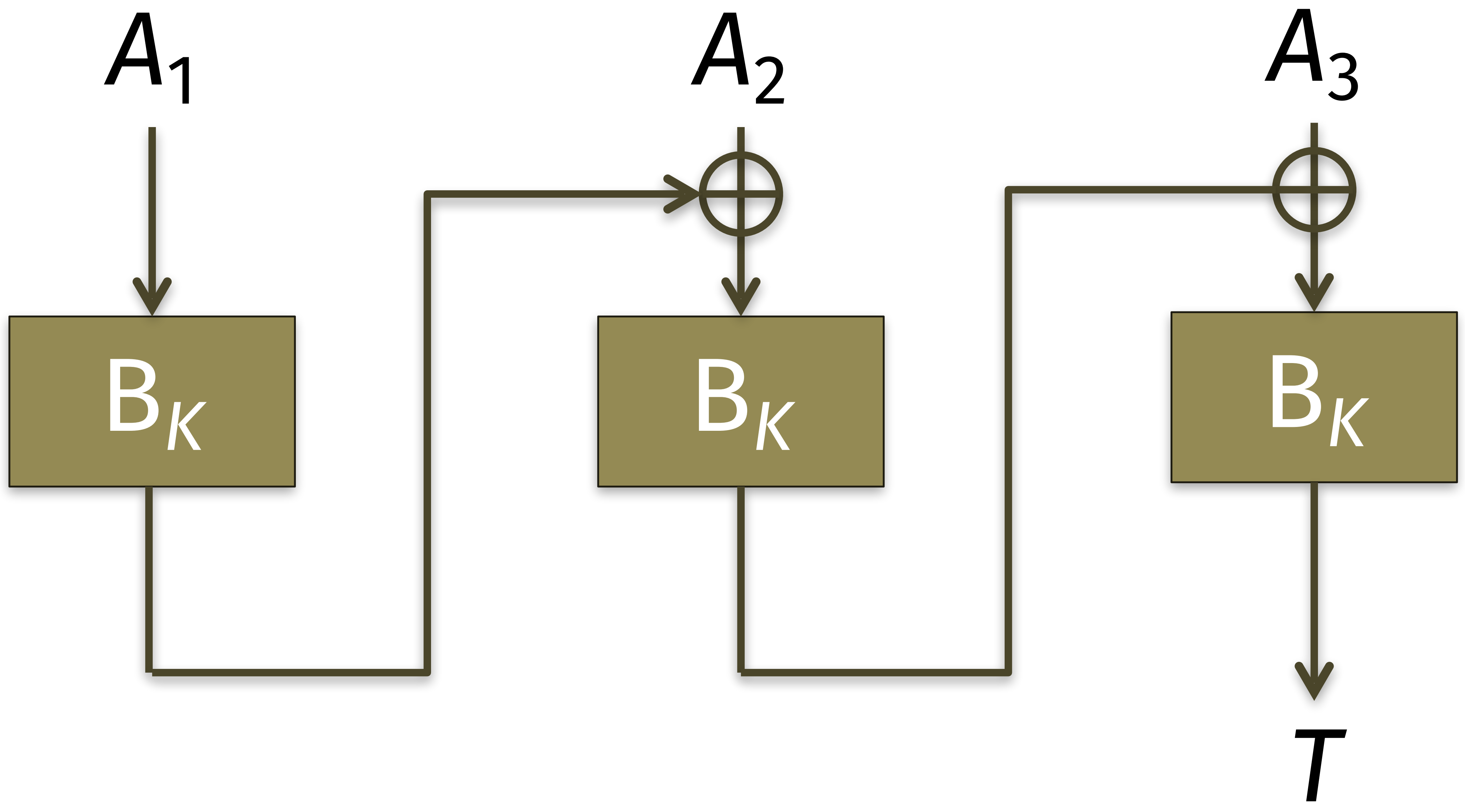
- *Bad throughput:* invoke B_K four times as much as minimally necessary
- *Long tag:* want tag length τ == security parameter λ, indep of msg length α

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

CBC-MAC: cipher block chaining, revisited

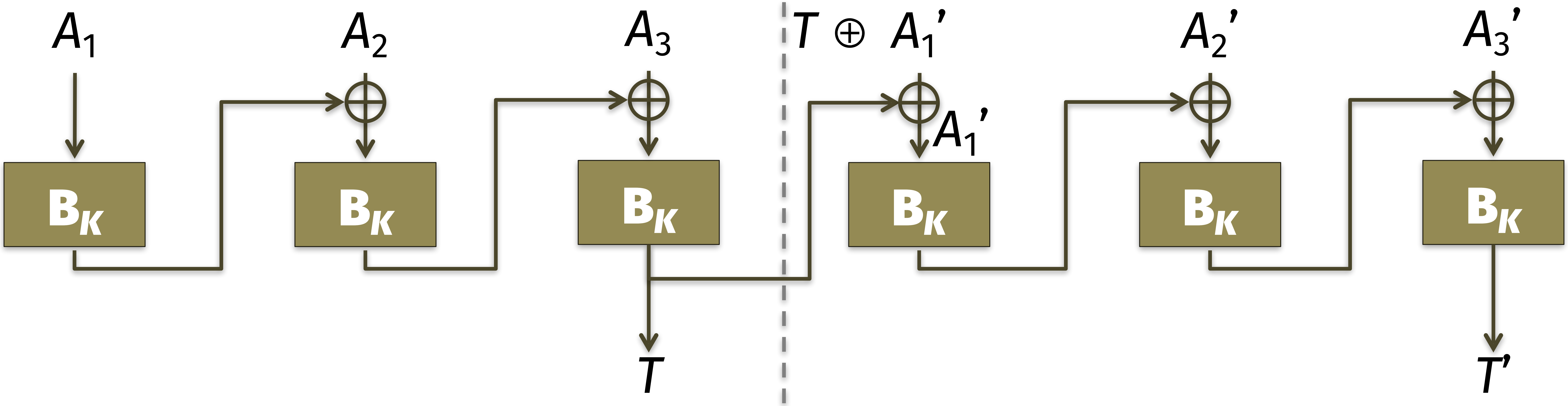
- 1st block simply runs the underlying block cipher (no more nonce/IV!)
- Subsequent inputs to the block cipher depend on both new input + prior output
- Only the final block tag is revealed ⇒ important for performance *and* security



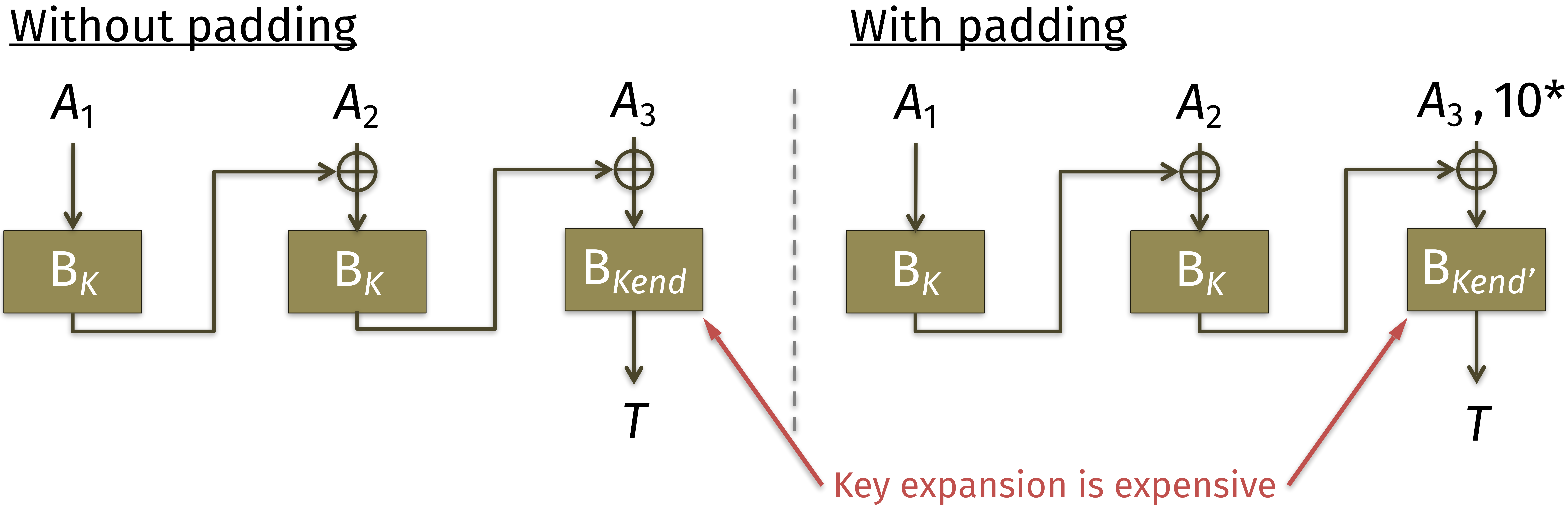
CBC-MAC

Theorem. 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

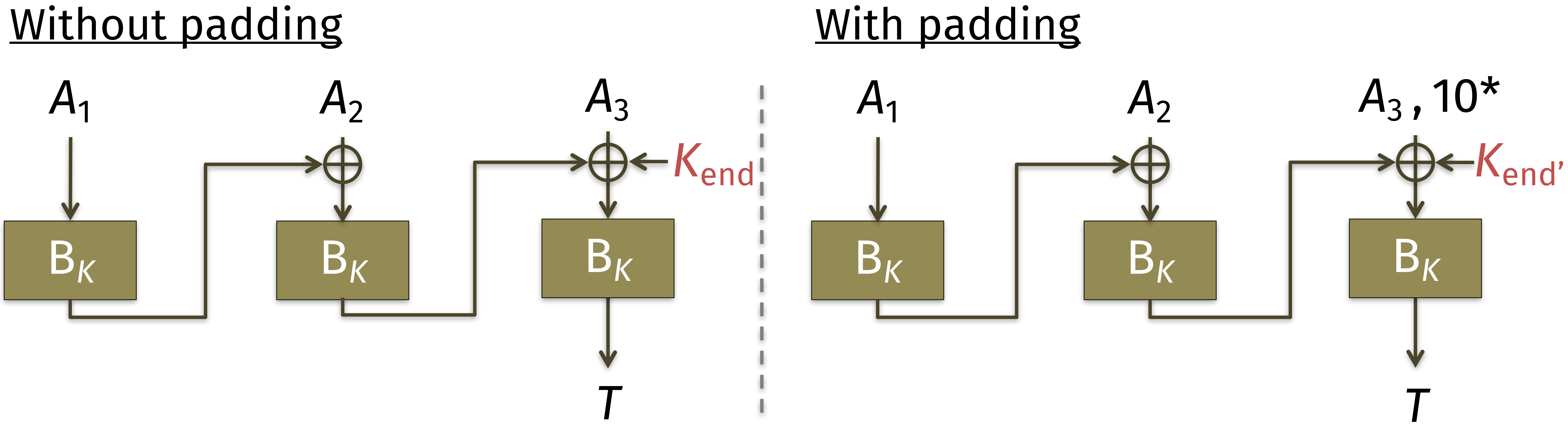
Theorem. CBC-MAC **insecure** if recipient doesn't know length in advance, or if length is not a multiple of the block length (i.e., padding won't work)



FMAC: Use a different key for the final step



Cipher-based MAC (CMAC)

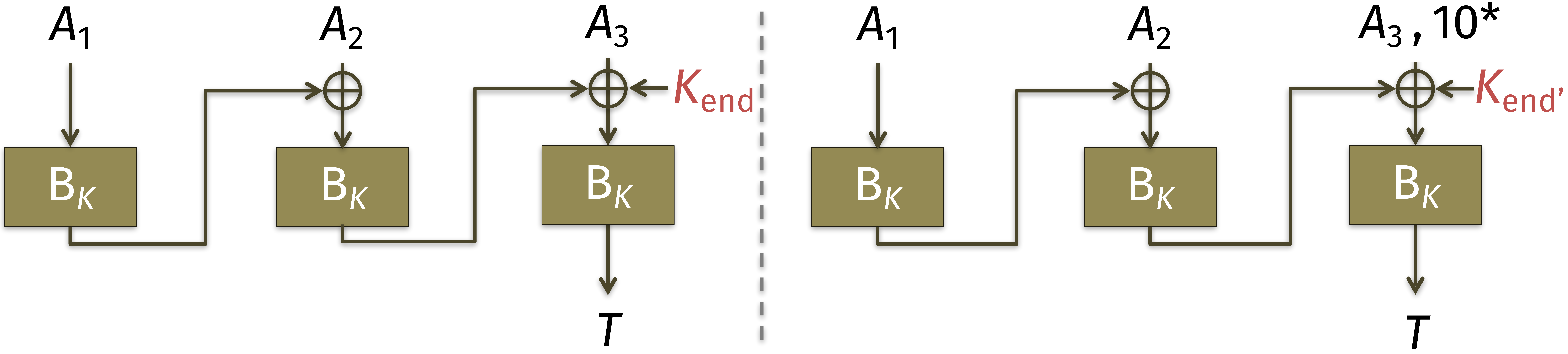


- Designed by Black and Rogaway, 2000
- Don't use extra keys to encrypt. Use them to influence the final block.

One-key CBC-MAC (OMAC)

Without padding

With padding



- Designed by Iwata & Kurosawa 2003
- Derive the finalization keys K_{end}, $K_{end'}$ from the original key K (saves on key length)

Crypto in practice uses… none of these MACs?

bu.edu homepage (2017)

Obsolete connection settings

The connection to this site uses TLS 1.0 (an obsolete protocol), RSA (an obsolete key exchange), and AES_256_CBC with HMAC-SHA1 (an obsolete cipher).

www.amazon.com

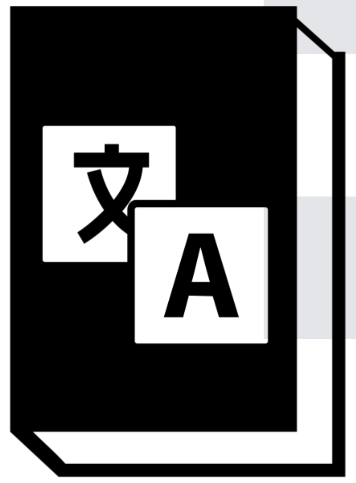
Secure connection

The connection to this site is encrypted and authenticated using TLS 1.2 (a strong protocol), ECDHE_RSA with P-256 (a strong key exchange), and AES_128_GCM (a strong cipher).

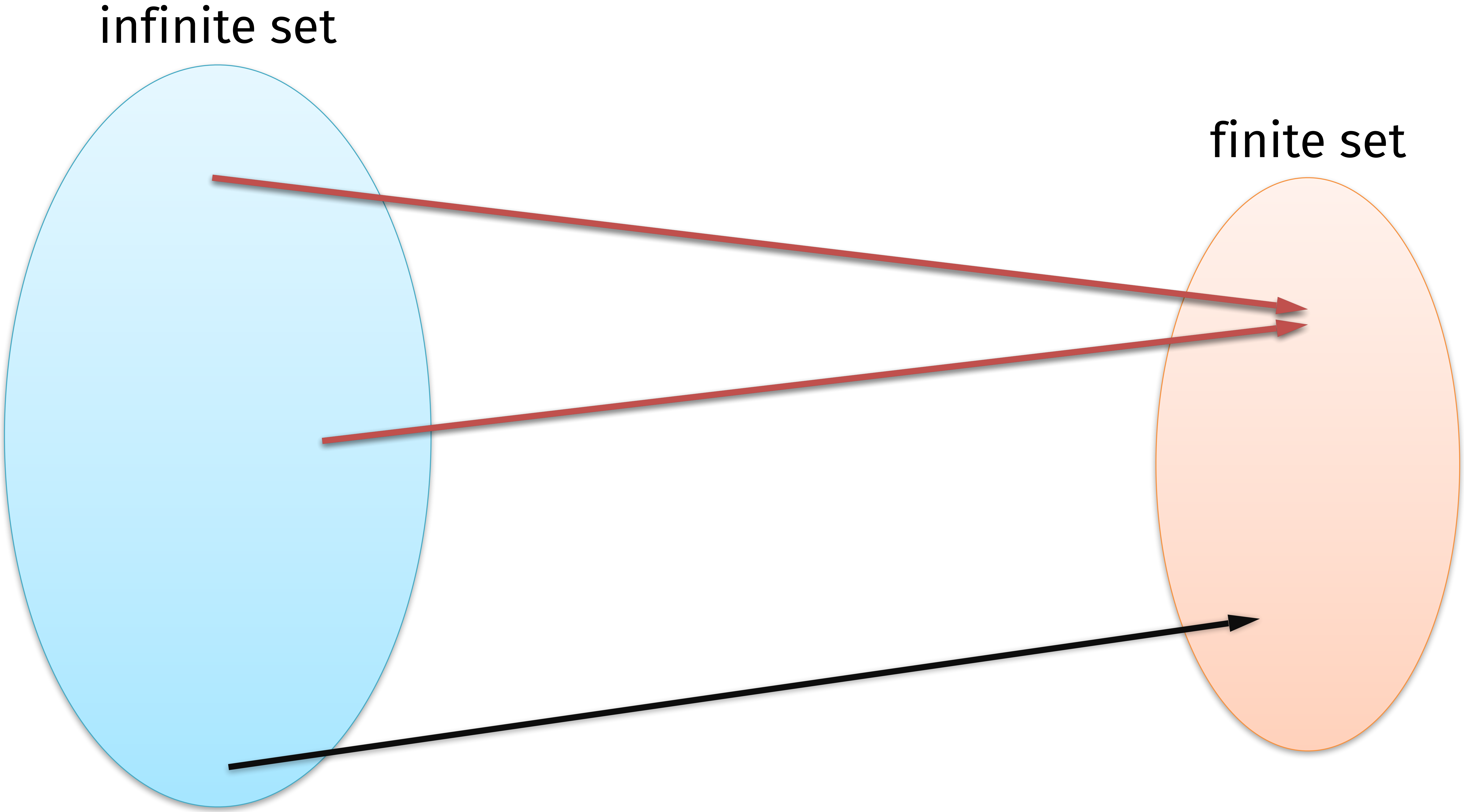
Hash function = 1 public, infinite-size codebook

- Hash function $H : \{0,1\}^\infty \to \{0,1\}^{out}$
- Compresses long messages into short digests
  - No longer possible to invert!
- We have already seen one in the homeworks: 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

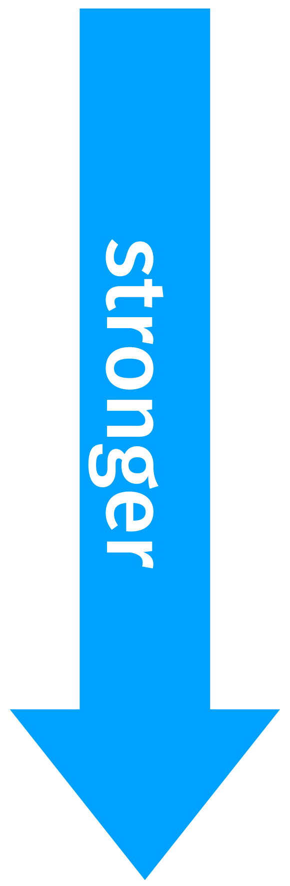


Hash function strength

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

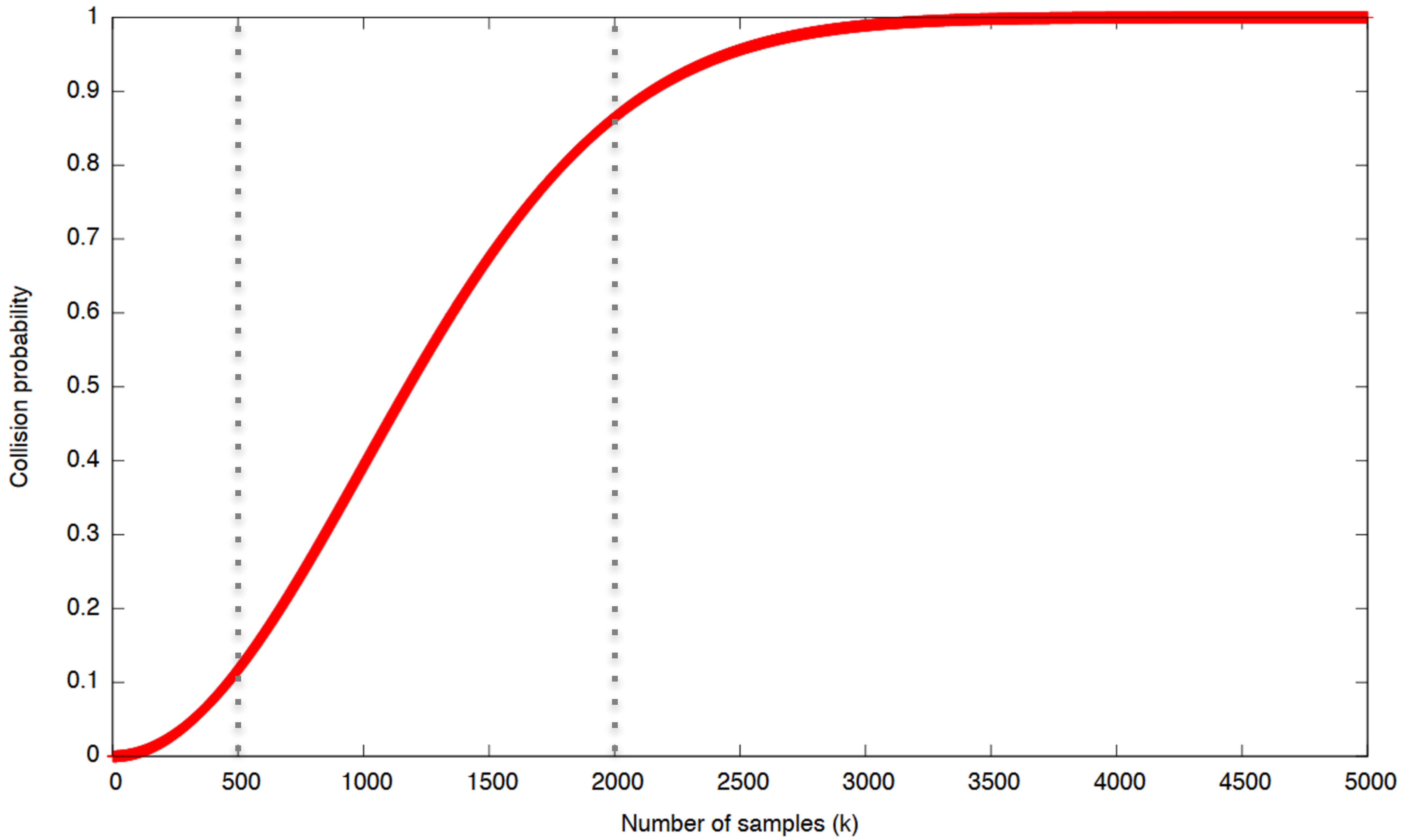
Security notions against adversaries who *possess the code* of *H*

- Preimage resistance: given **y** = **H**(**x**), tough to find any preimage **x'**
- 2^{nd} preimage resistance: given **x**, tough to find new **x'** s.t. **H**(**x'**) = **H**(**x'**)
- Collision resistance: given only **H**, difficult to find two different inputs **x** and **x'** s.t. **H**(**x'**) = **H**(**x'**) faster than a birthday bound search



Giving Mallory the code of **H** >> her power in any other game in this class

Birthday bound (reminder)



- 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

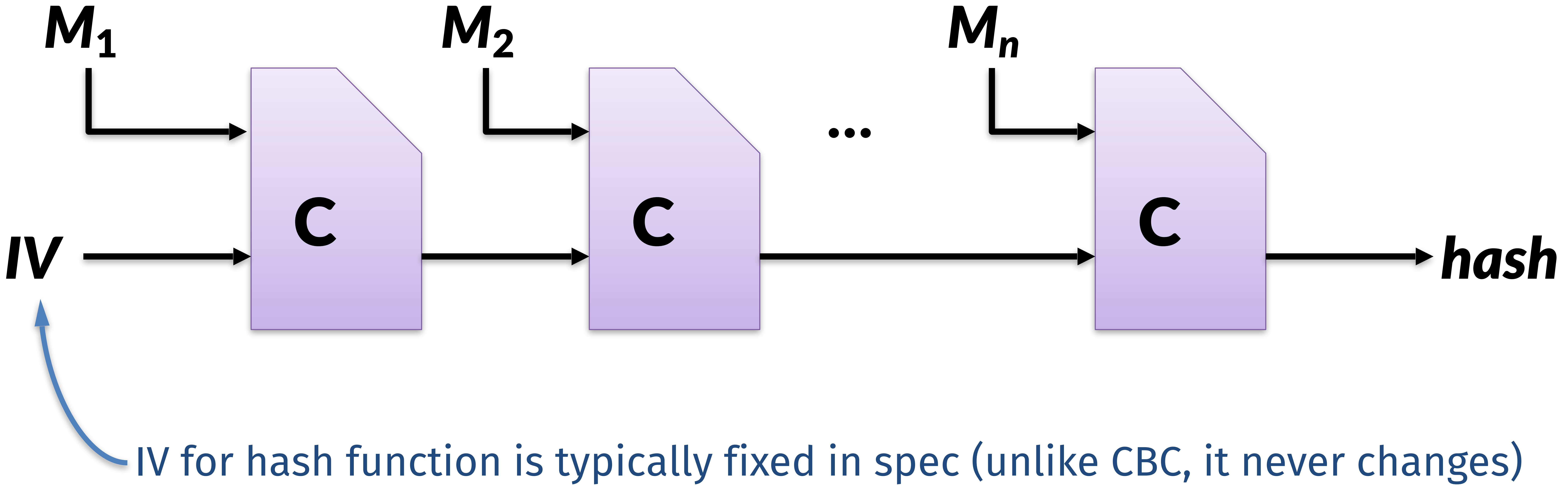
Secure Hash Algorithm (SHA) family

- SHA-1 and SHA-2 are NSA-designed, NIST-approved
  - 1995: SHA-1 (160 bits) is now broken, though still occasionally used today
    - 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: New SHA-3 standard with different design + designers (will discuss later)

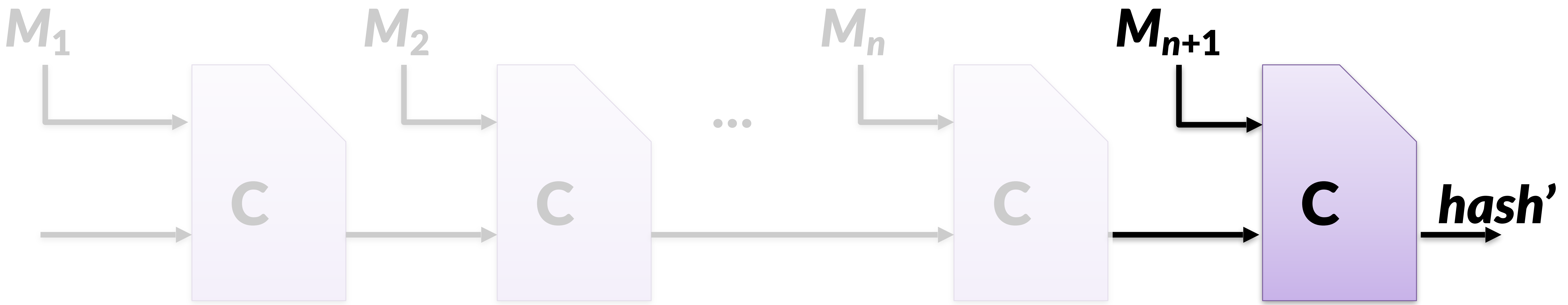
Merkle-Damgård paradigm

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

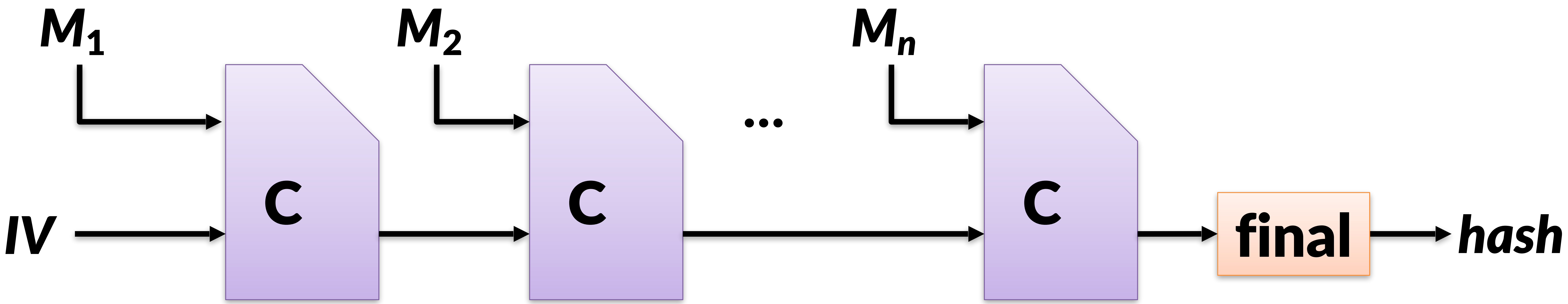
1. A fixed-length, *compressing* random-looking function
2. A mode of operation that iterates this function multiple times in a smart manner



Problem: Length extension attack

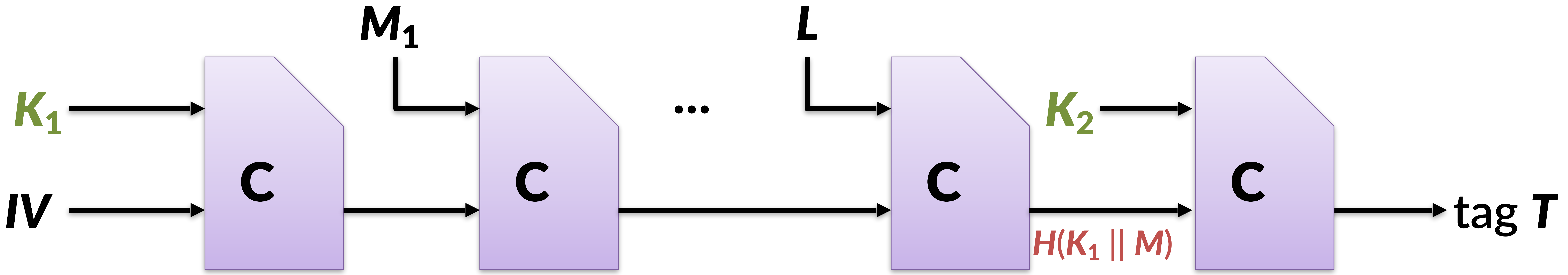


Countermeasure: finalization



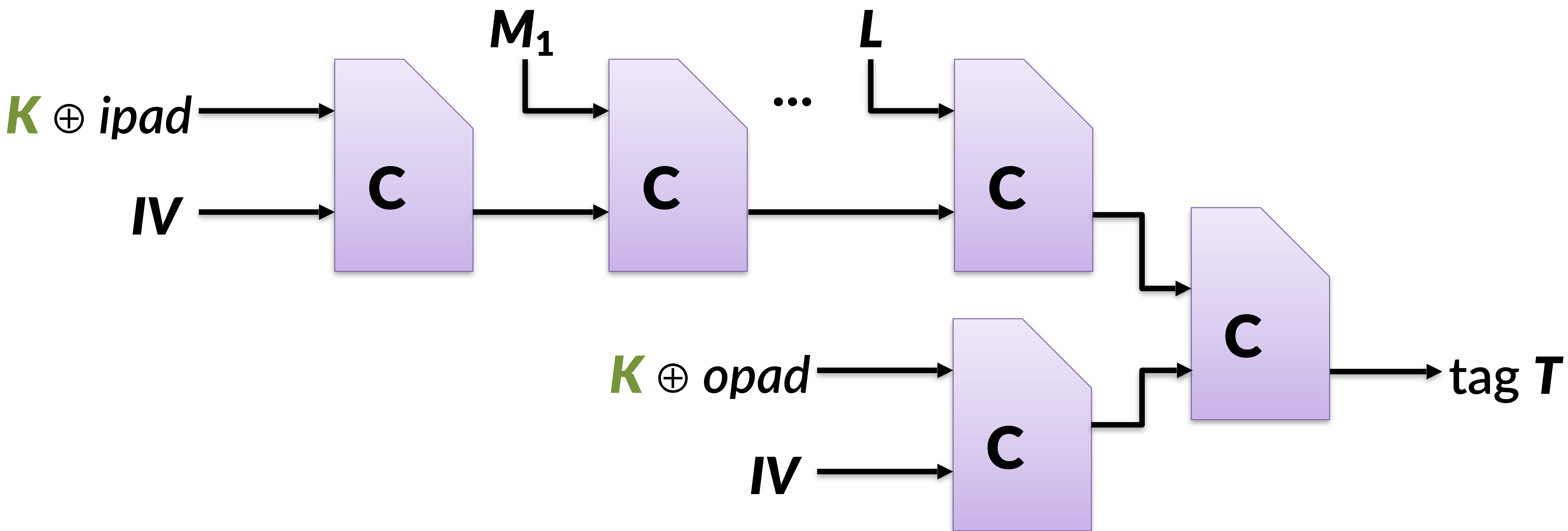
Hash function → MAC

- *NMAC*: finalize the hash function by calling *C* one more time
- There are two keys, and the final step depends on the second key



HMAC [Bellare Canetti Krawczyk 97]

- *HMAC*: Like before, but derive two “independent” keys from one key
  - Recall from CMAC → OMAC story: nobody wants to carry multiple keys
  - Fixed constants ipad = 0x5C, opad = 0x36 repeated to equal the key length



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 in 2017: