Lecture 4: Message authentication codes

Posted on piazza.com

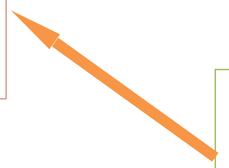
- Lab 2: due on Monday 2/4 at 11pm
- Week 2 reading
- Lab 1 answers

Roadmap for this course

Math Tools

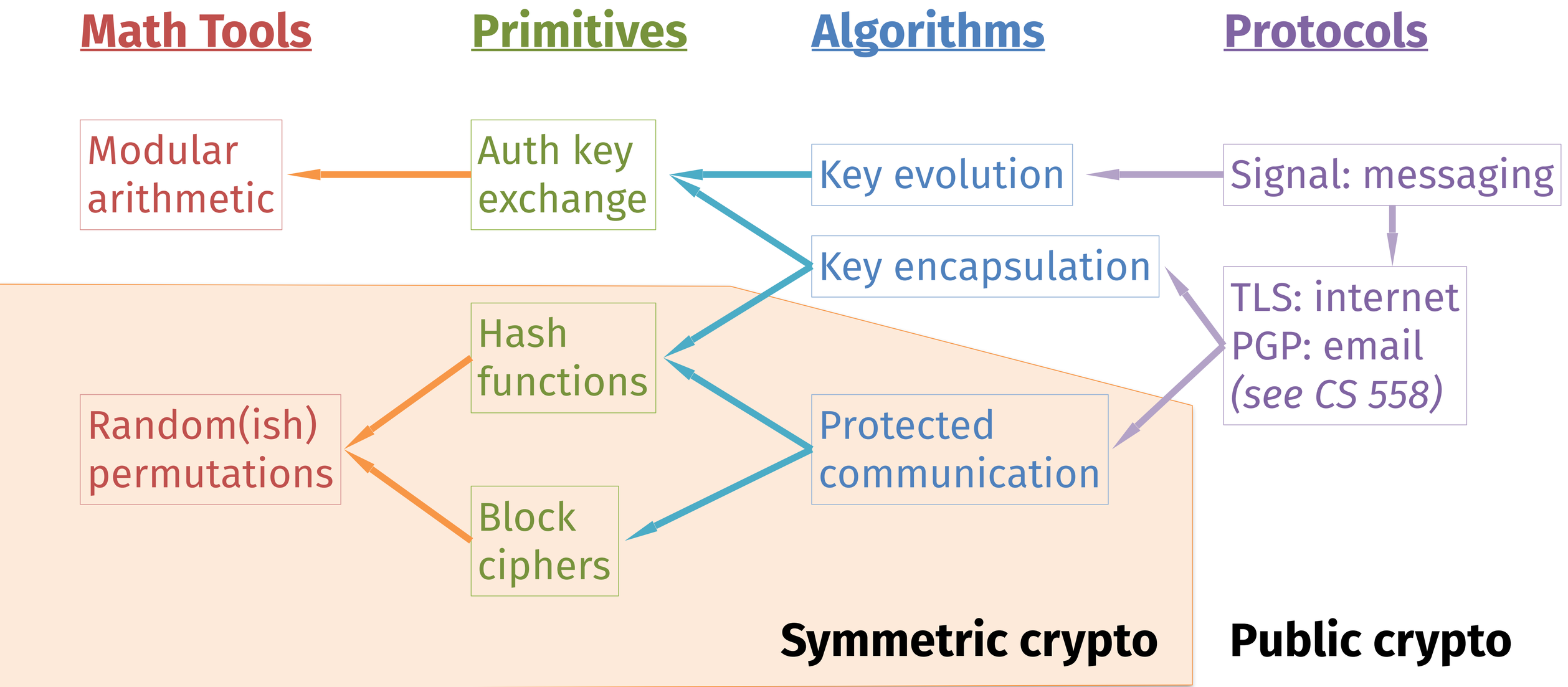
Primitives

Random(ish) permutations

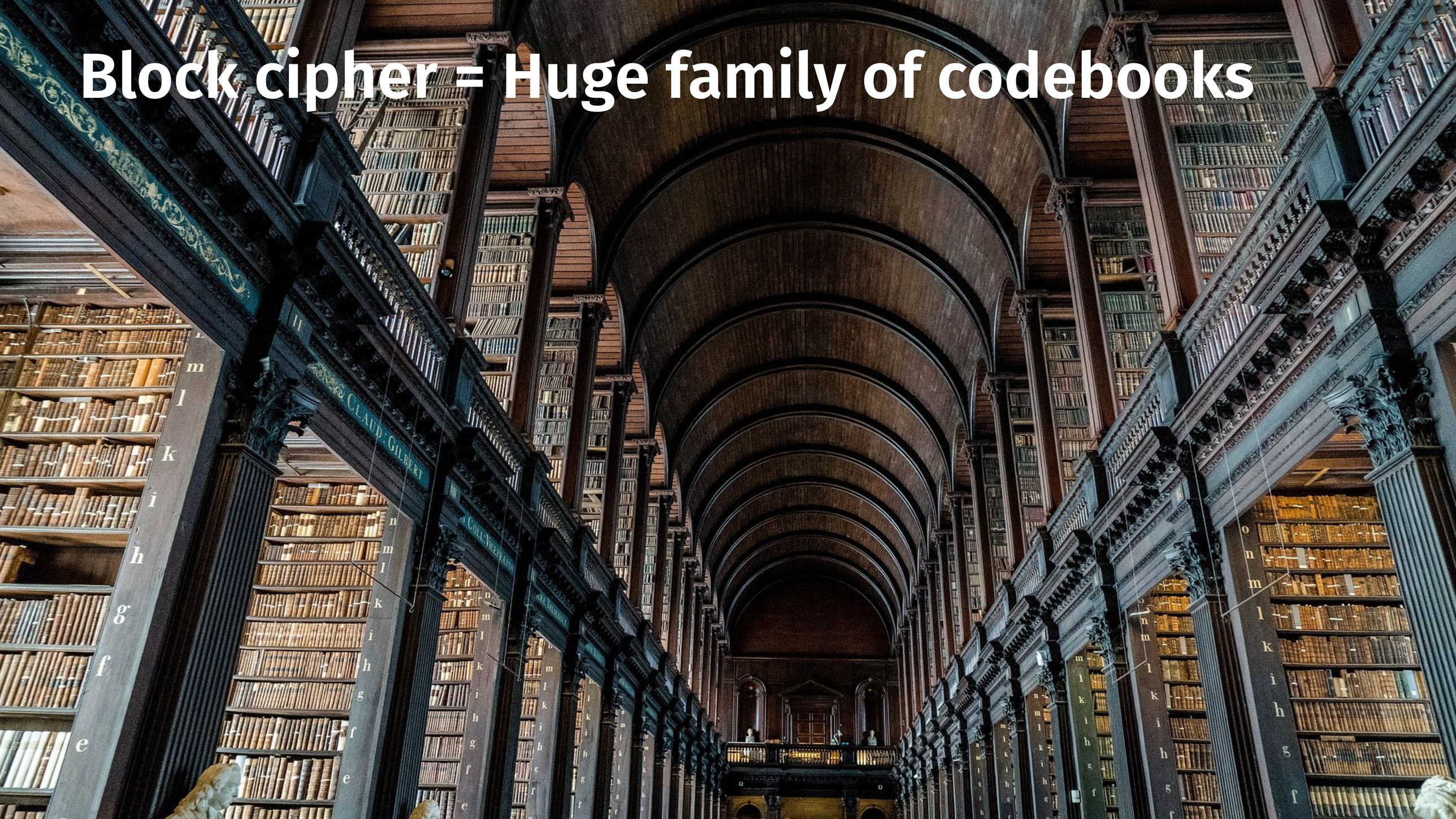


Block ciphers

Roadmap for this course

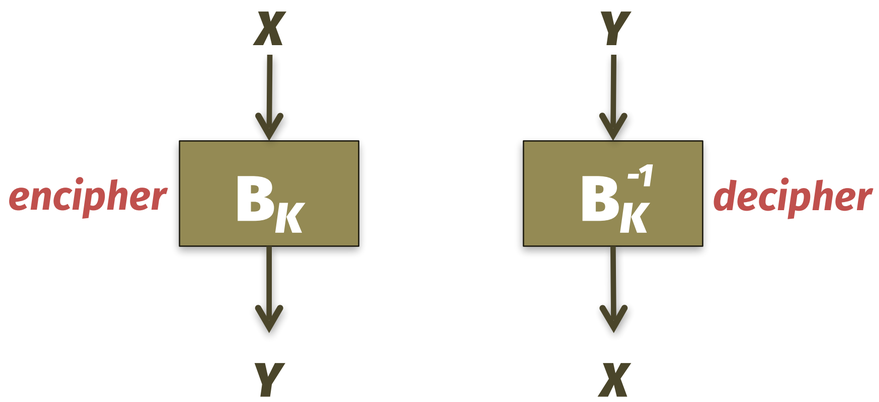


Block cipher = Huge family of codebooks



Block cipher

- Family of permutations, each of the form $B : \{0,1\}^{\mu} \rightarrow \{0,1\}^{\mu}$
- Key $K \in \{0,1\}^{\lambda}$ determines which permutation to use



- B_K is *strongly pseudorandom* if every adversary running in time ≤ t and making ≤ q queries cannot tell it apart from a secret, truly random Π

chosen from 2^{λ} options

chosen from $\approx (2^{\mu})^{(2^{\mu})} > 2^{2^{\mu}}$ options

Two ways crypto primitives can go bad

1. **Broken:** can be attacked much faster than with brute force (e.g., 2DES)

2. **Obsolete:** people have the computing power required to conduct a brute force attack (e.g., DES)

How to think about really large numbers

Converting to base-2

$2^{10} \approx 10^{3}$

$2^{20} \approx 10^{6}$

$2^{30} \approx 10^{9}$

$2^{40} \approx 10^{12}$

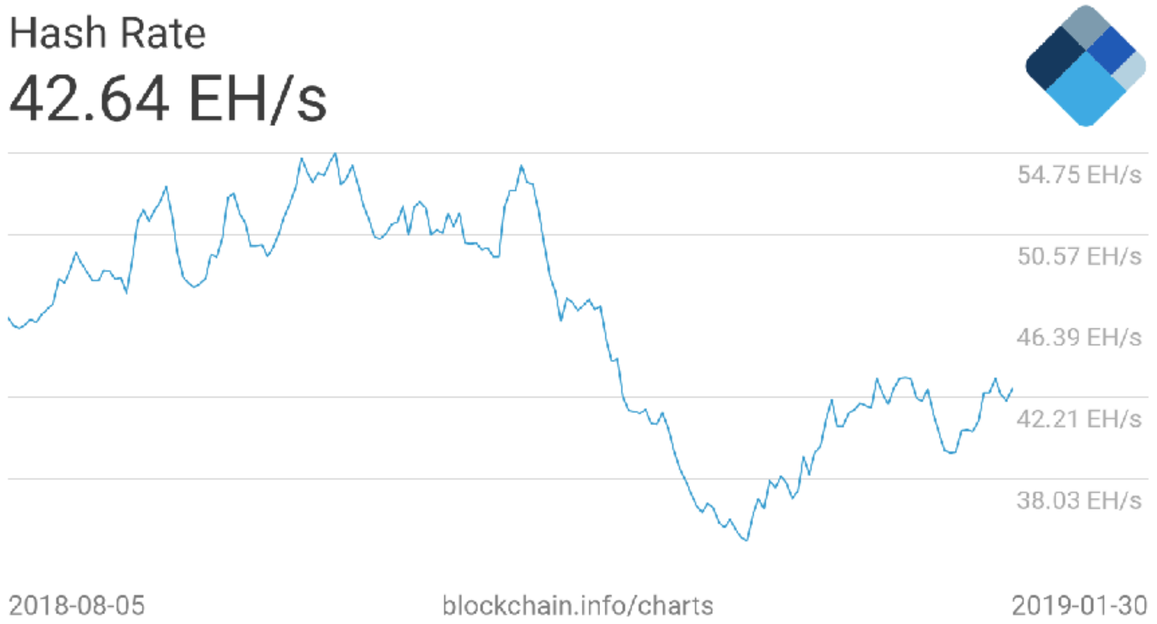
$2^{50} \approx 10^{15}$

$2^{60} \approx 10^{18}$

$\vdots$

Speed of cryptography on modern computers

- Running AES: $(2^{31}\ {}^{\text{cycles}}/_{\text{sec}})$ / $(10\ {}^{\text{cycles}}/_{\text{aes}}) \approx 2^{27}\ {}^{\text{aes}}/_{\text{sec}}$
  - So about $2^{52}\ {}^{\text{ops}}/_{\text{year}}$, given that 1 year ≈ 2^{25} sec
- Entire bitcoin network: about $2^{65}\ {}^{\text{ops}}/_{\text{sec}}$



Difficulty of attacking crypto

Eve's search space	Time with laptop (2^{52} ops/yr)	Time with bitcoin network (2^{65} ops/sec)
2^{20} (your lab)	0.01 second	~instantaneous
2^{56} (DES brute force)	2^{4} = 16 CPU core-years	much less than 1 second
2^{80}	2^{28} = 256 million CPU core-yr	2^{15} seconds ≈ 9 hours
2^{128} (AES brute force)	2^{76} = 64 sextillion CPU core-yr = 2^{33} x 2^{43} = 8 trillion CPU core-yr for each person on earth	2^{63} sec ≈ 2^{38} year ≈ 100 billion yr
2^{256} (largest AES)	(about the energy of the sun)	

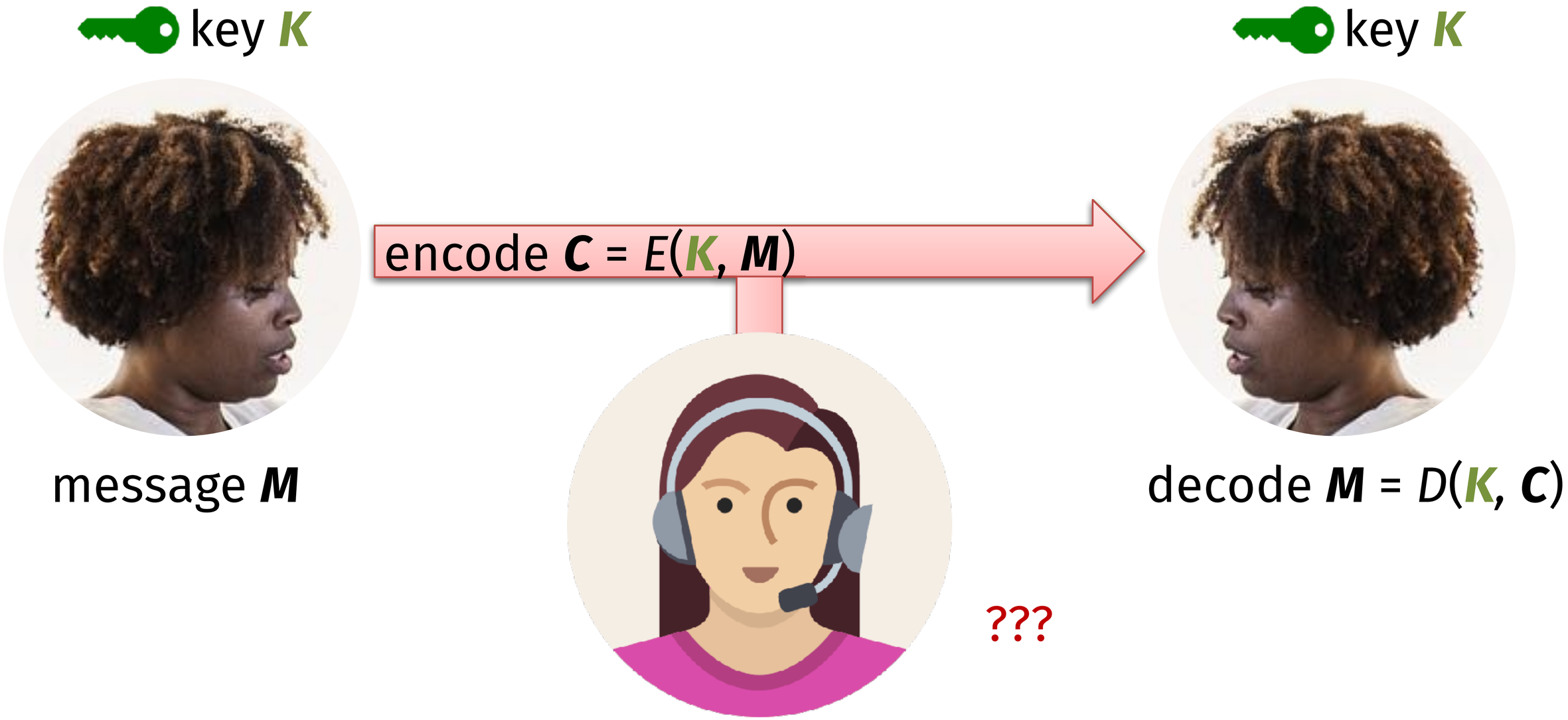
Random functions of different lengths

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

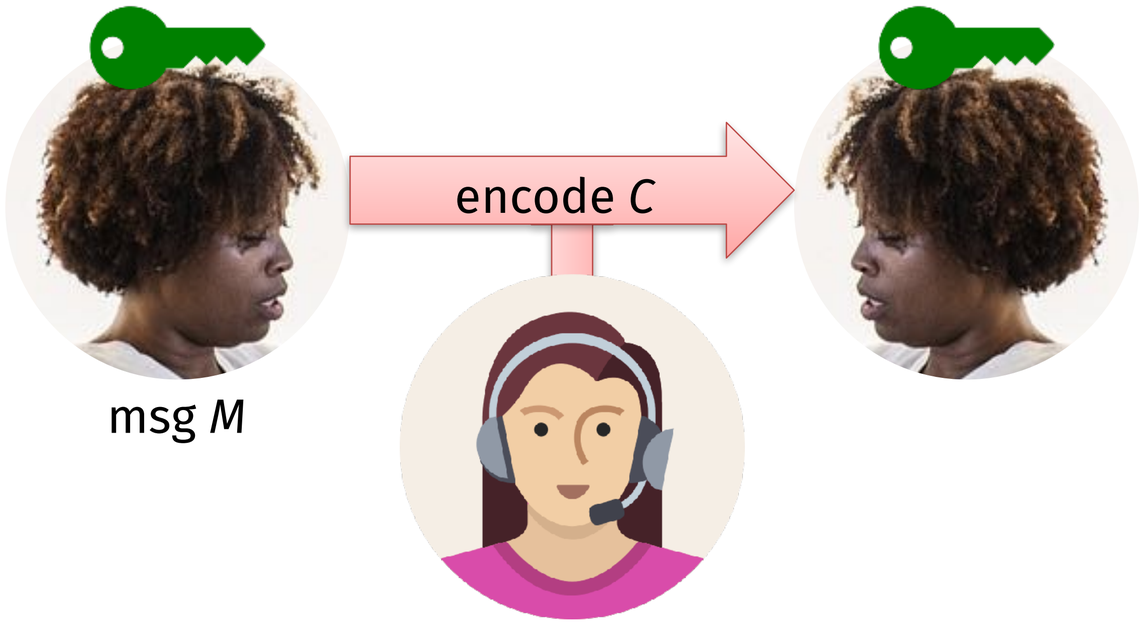
Case:	in < out	in = out (permutation)	in > out	in = ∞
One *public* function	Stream cipher ←	Codebook	→ Compression function	→ Hash function / random oracle
Family of functions indexed by *private* key		↓ Block cipher	→ Message auth code	↗

"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*

Part 1: Protecting data at rest

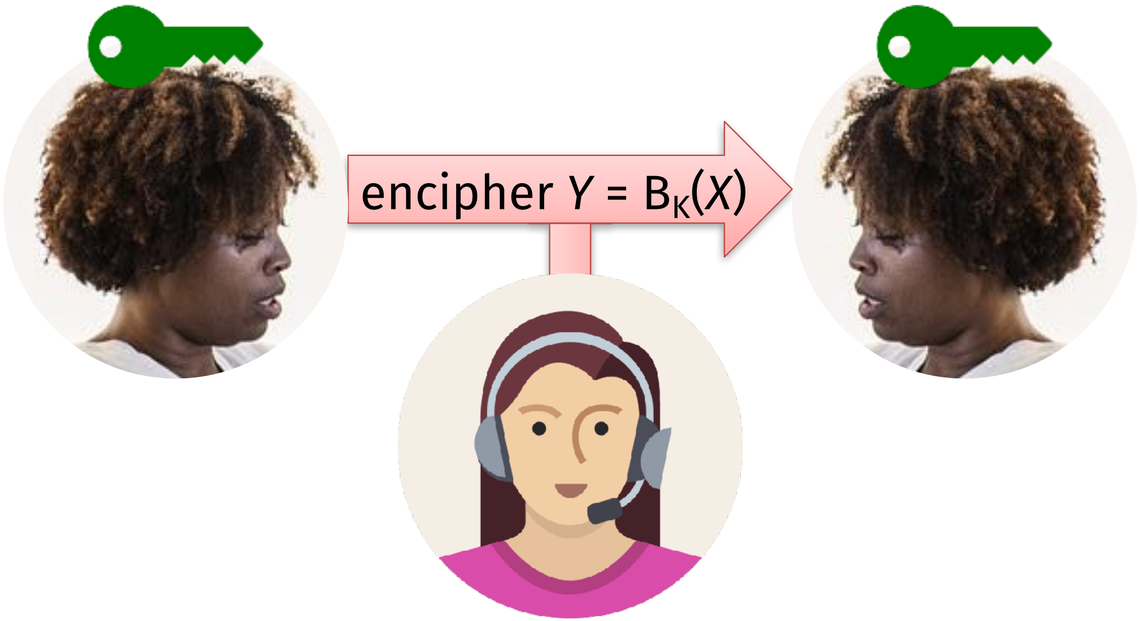


Alice's integrity + confidentiality goals



- *Data authenticity*:
  if Eve tampers with *C*, then Alice can detect the change
- *Entity authenticity*:
  future Alice knows that she previously created *C*
- *Privacy*: Eve cannot learn *M*

Does a cipher give us protected communication?



Yes, we get authenticity + privacy!

But only if

- Message length is exactly one block
- (For privacy) Alice sends just 1 message

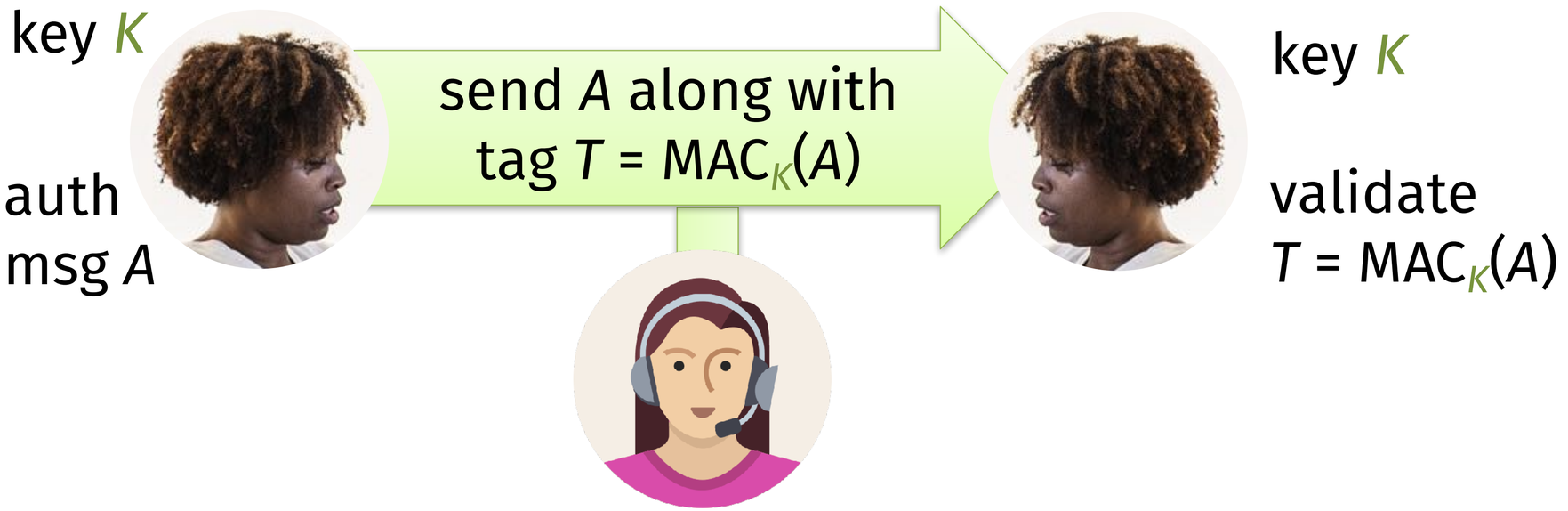
In general: no. Ciphers are a building block toward protected comms but do not provide it on their own.

Auguste Kerckhoffs' principles to protect communication

1. The system must be practically, if not mathematically, ***indecipherable***
2. It should ***not require secrecy***, and it should not be a problem if it falls into enemy hands
3. It must be possible to communicate and ***remember the key*** without using written notes, and correspondents must be able to ***change or modify it at will***
4. It must be applicable to telegraph communications
5. It must be portable, and should not require several persons to handle or operate
6. Lastly, given the circumstances in which it is to be used, the system must be easy to use and should ***not be stressful to use*** or require its users to know and comply with a long list of rules

Source: *La Militaire*, 1883

Message authenticity

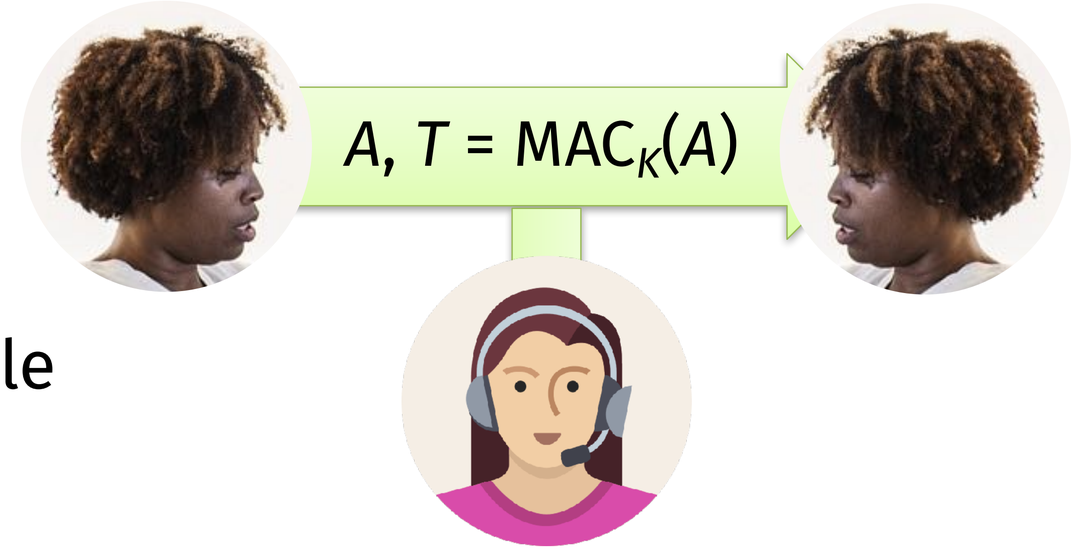


Objective of actively malicious Mallory:
inject a new message and tag (A, T)
or tamper with an existing one

What cryptographic authenticity will *not* do

- *Hide message contents*:
  Need encryption for that

- *Thwart replay attacks*:
  A higher-level protocol needs to handle this, say via nonces or timestamps



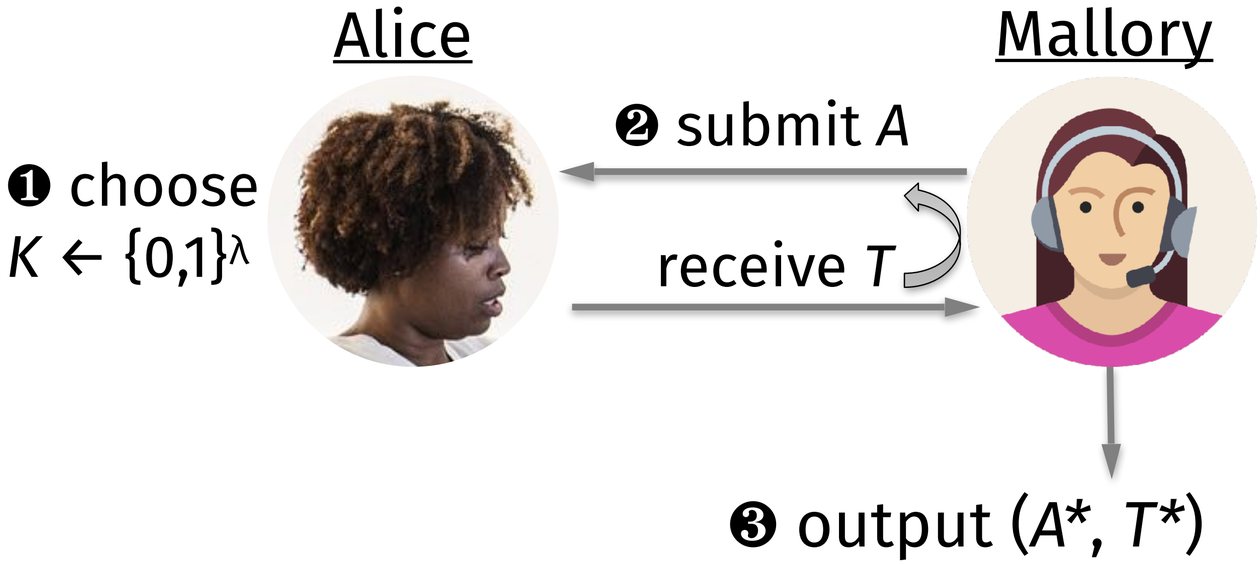
Definition: Message authentication code

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

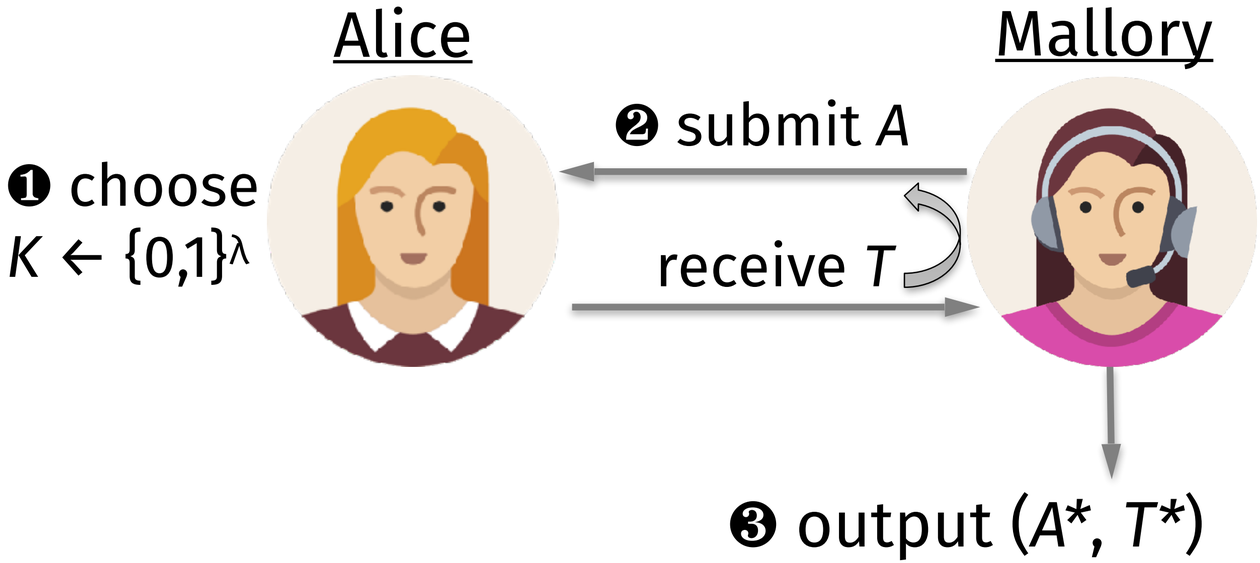
Even after viewing many (A, T) pairs,
Mallory cannot ***forge*** a new one



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

Existential unforgeability

We say that a MAC has ***(q, t, ε)-existential unforgeability against a chosen message attack*** if all adversaries that make ≤ q queries and run in time ≤ t can forge a message with probability < ε



Block cipher → MAC

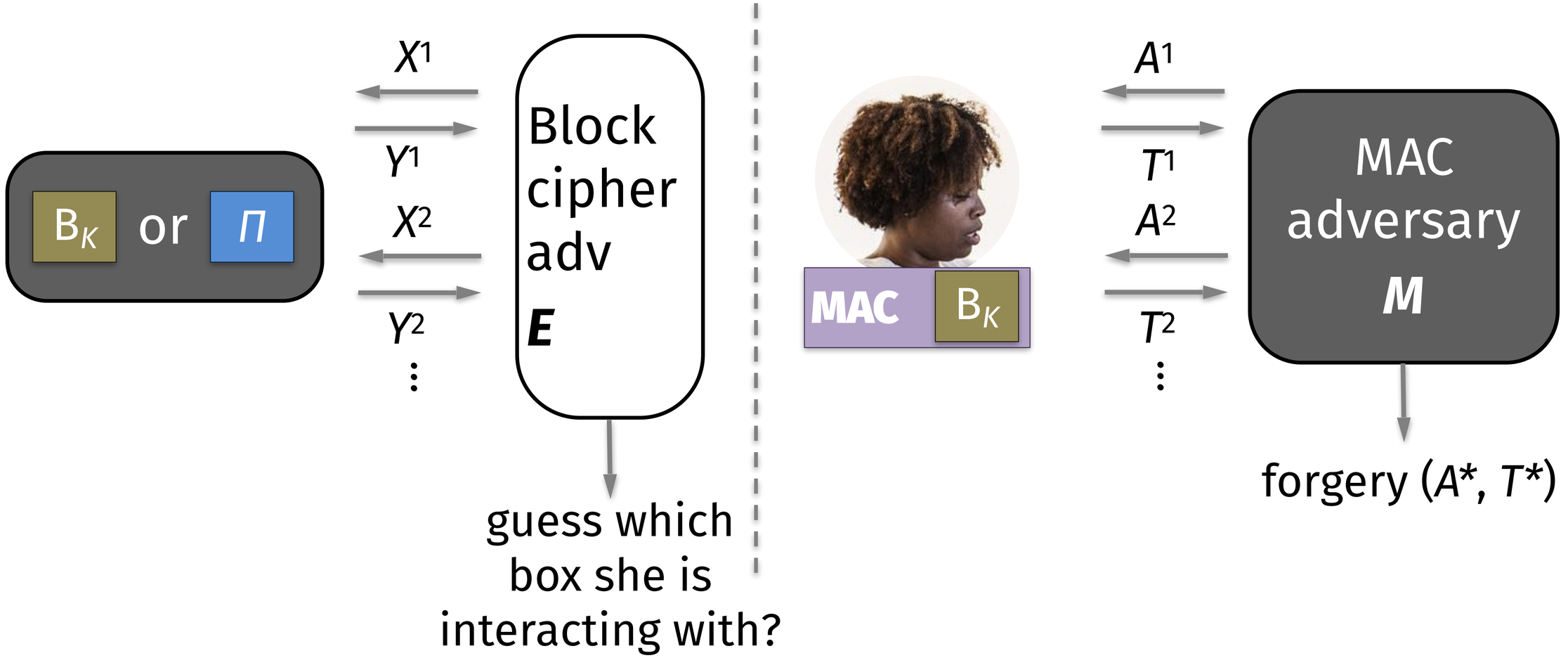
- For our first MAC, let's restrict $|A| = |T|$ = block length of a block cipher
- In this case, simply applying the block cipher suffices!

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

- How do we prove this claim?
  - B_K is pseudorandom, meaning Mallory cannot distinguish it from Π
  - The EU-CMA game is about forgery; it doesn't have an indistinguishability style
- What if we made the MAC from Π rather than B_K?
  - Remember, the output of $\Pi(X)$ doesn't depend on $\Pi(X')$ for any $X \neq X'$

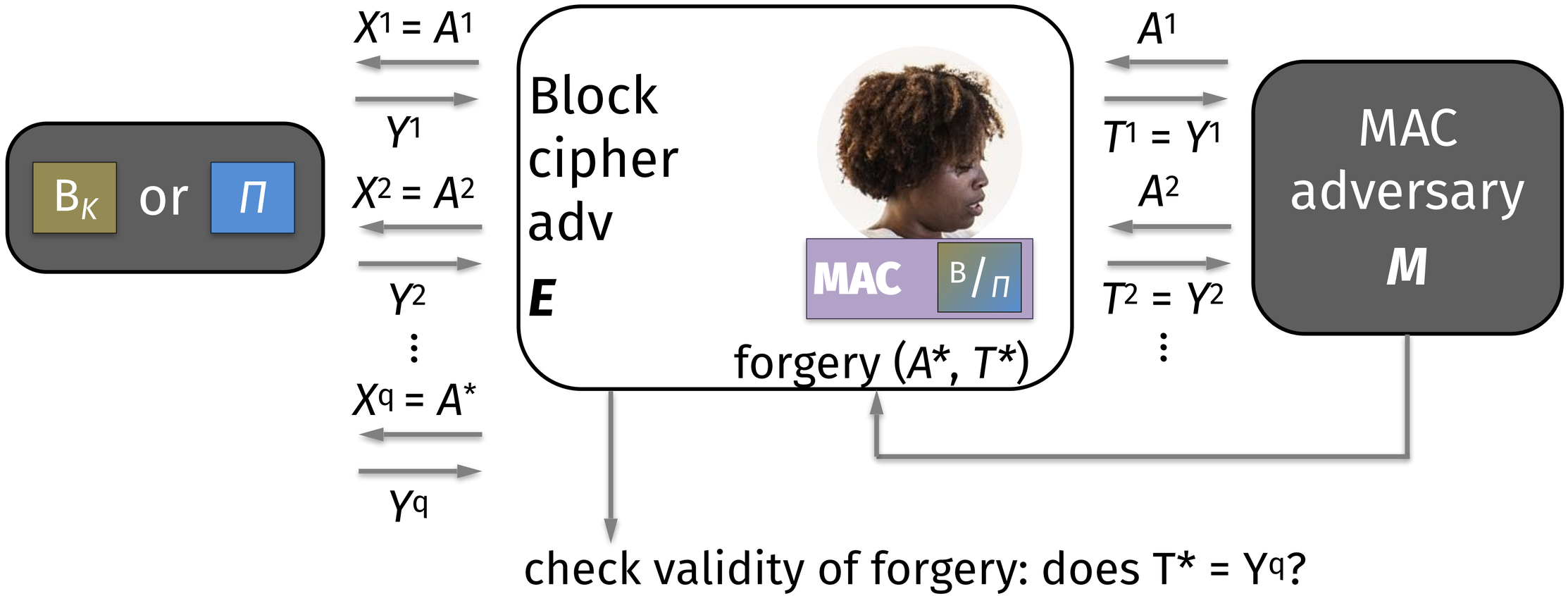
Thm: B_K is pseudorandom → MAC B_K is EU-CMA

Prove the contrapositive: given adversary *Mallory* that *forges* a MAC, we will construct an adversary *Eve* that distinguishes a block cipher from *Π*



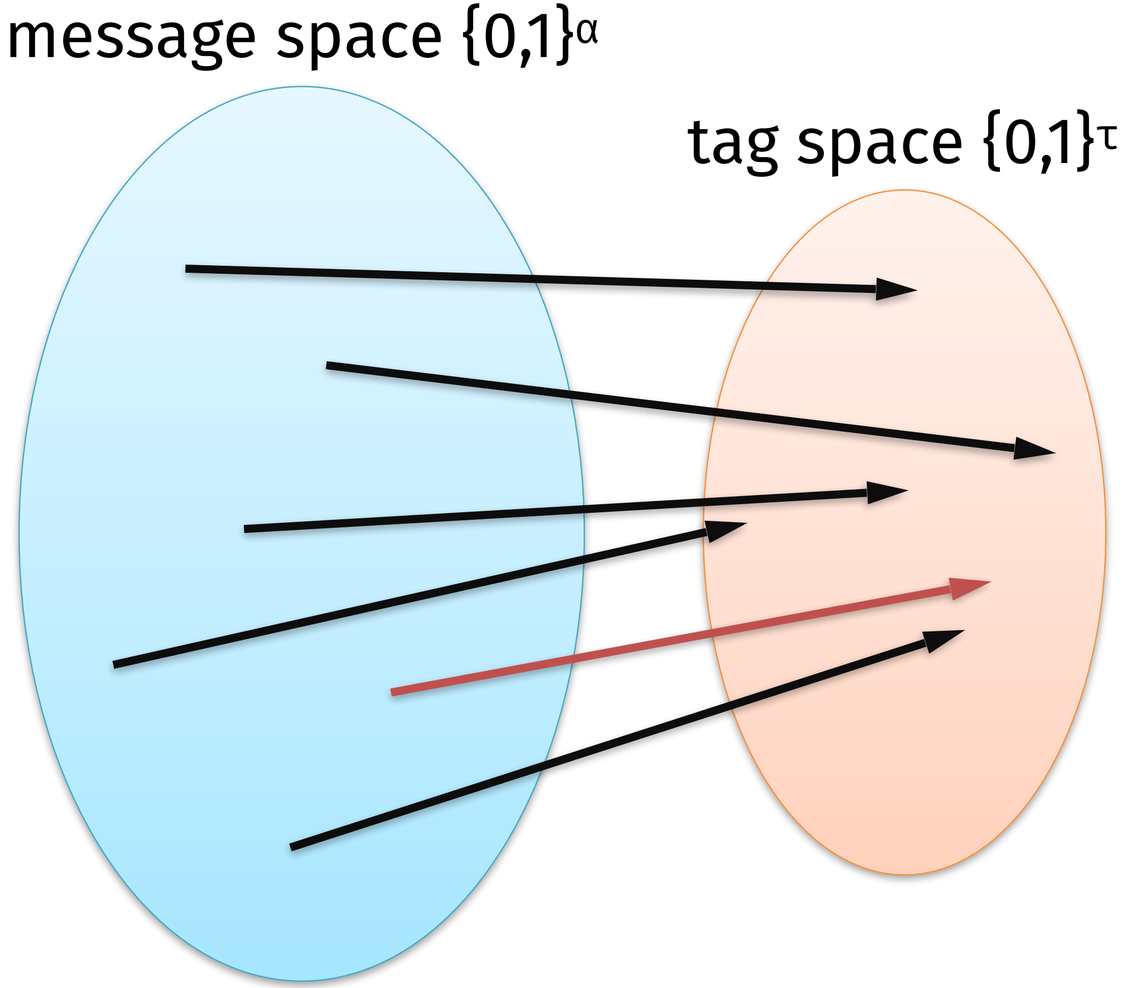
Thm: B_K is pseudorandom → MAC B_K is EU-CMA

Why this works: If *E* had access to B_K then *M* can forge. If *E* had access to *Π* then Pr[*M* forges] ≤ $2^{-\tau}$ because *Π*(A*) is independent of other queries



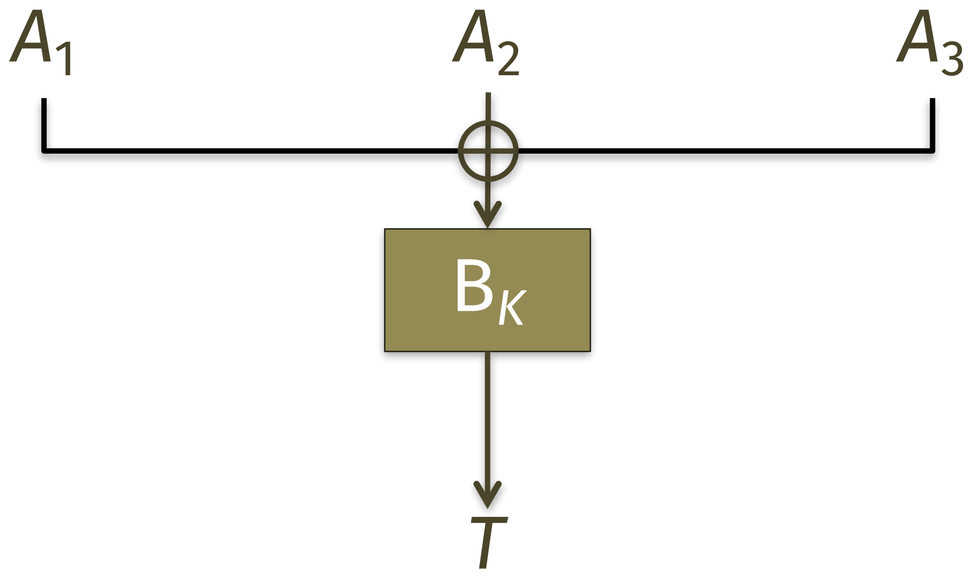
MACs for longer messages?

- *Performance goal:* minimize space required for the MAC tag
- *Security goal:* ensure that MAC remains existentially unforgeable



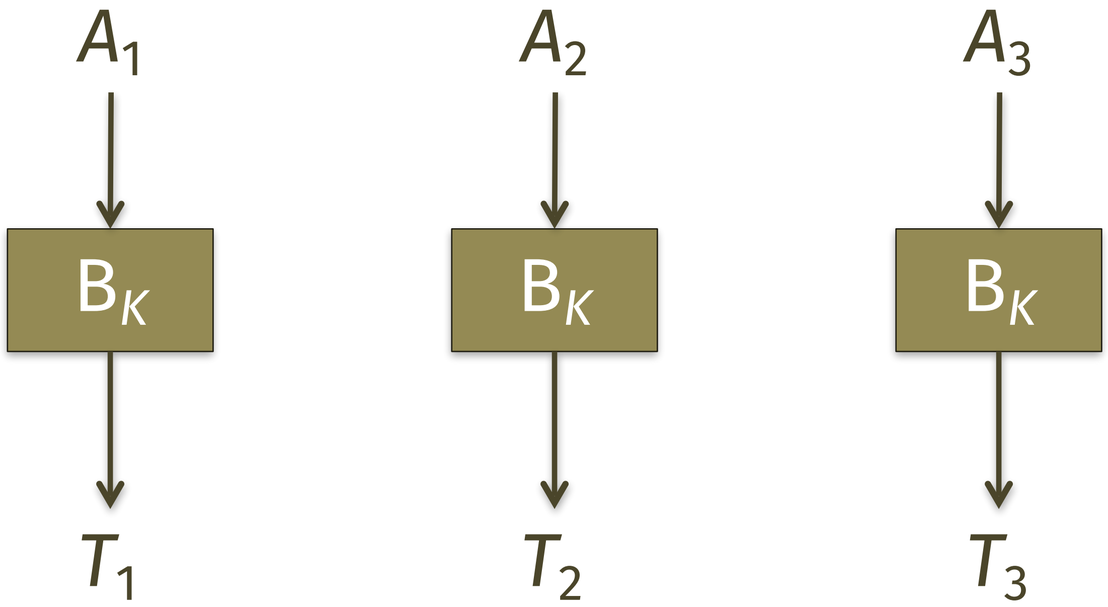
Variable length MACs?

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



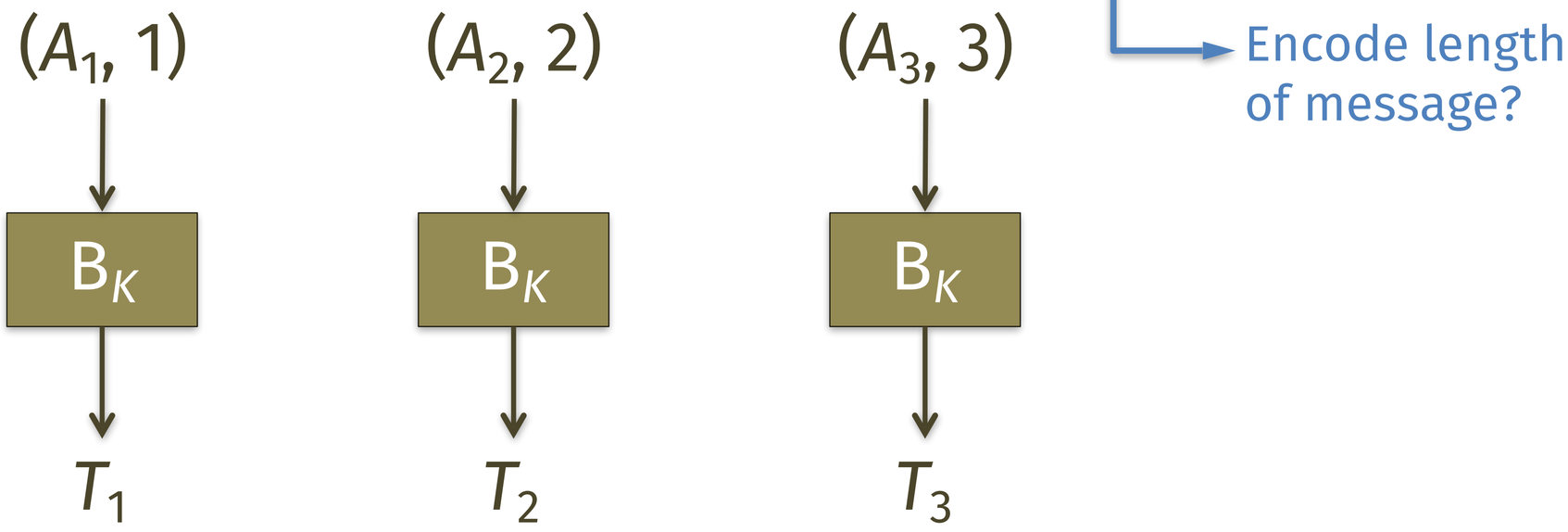
Variable length MACs?

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

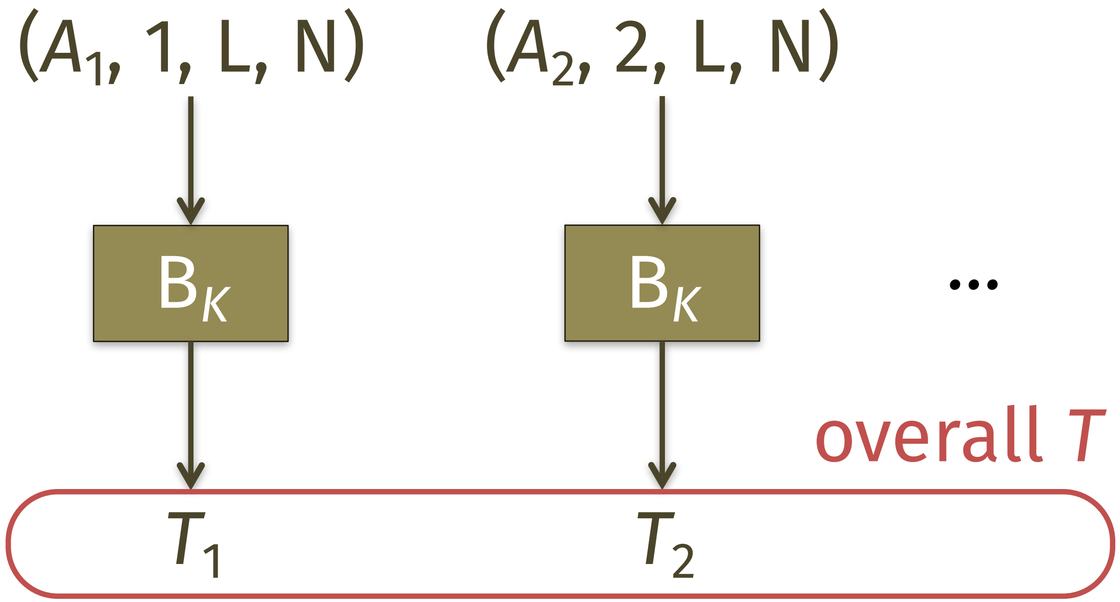


A construction that works

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