MATH/CMSC 456 :: UPDATED COURSE INFO

Instructor: Gorjan Alagic (galagic@umd.edu); ATL 3102, office hours: by appointment

Textbook: *Introduction to Modern Cryptography*, Katz and Lindell;

Webpage: alagic.org/cmsc-456-cryptography-spring-2020/ (slides, **reading posted here**);

Piazza: piazza.com/umd/spring2020/cmsc456

ELMS: active, slides and reading posted there, **homework 2 due midnight Tonight.**

Gradescope: active, access through ELMS.

TAs (Our spot: shared open area across from **AVW 4166**)

- Elijah Grubb (egrubb@cs.umd.edu) 11am-12pm TuTh (AVW);
- Justin Hontz (jhontz@terpmail.umd.edu) 1pm-2pm MW (AVW);

You can use AVW 4172 as a waiting room.

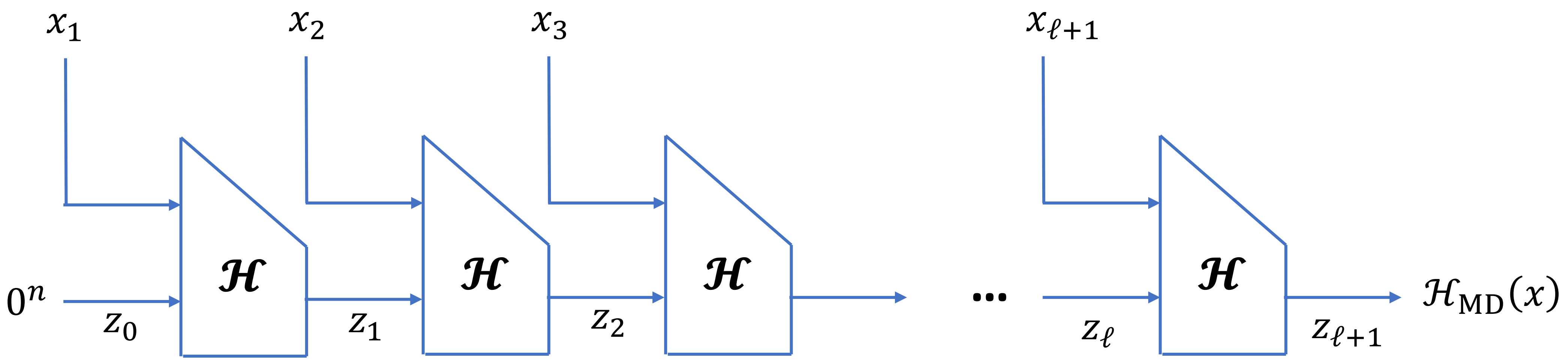
Additional help:

- Chen Bai (cbai1@terpmail.umd.edu) 3:30-5:30pm Tu (2115 ATL – inside **JQI**)
- Bibhusa Rawal (bibhusa@terpmail.umd.edu) 3:30-5:30pm Th (2115 ATL – inside **JQI**)

RECAP: Merkle-Damgård transform

Construction (Merkle-Damgård).
Let $(\mathbf{KeyGen}_{\mathrm{H}}, \mathcal{H})$ be a hash function, and suppose $\mathcal{H}: \{0,1\}^{2n} \to \{0,1\}^n$. Define a new hash function:

- $\mathbf{KeyGen}_{\mathrm{H_{MD}}}$: same as $\mathbf{KeyGen}_{\mathrm{H}}$;
- $\mathcal{H}_{\mathrm{MD}}: \{0,1\}^* \to \{0,1\}^n$ defined as follows, on input x:
  1. assume length $|x|$ of x is divisible by n (otherwise pad with 0s);
  2. split x as $x = (x_1, x_2, \ldots, x_\ell)$ and set $x_{\ell+1} \coloneqq |x|$.
  3. set $z_0 = 0^n$; compute $z_i = \mathcal{H}(x_i, z_{i-1})$;
  4. output $z_{\ell+1}$.



RECAP: Merkle–Damgård transform

Remember:

- critical property we needed for integrity checks…
- … and for Hash-and-MAC…
- … was collision-resistance!

What happens when we apply MD?

Theorem. If $\mathcal{H}$ is a collision-free hash function, then so is its Merkle-Damgård transform $\mathcal{H}_{\mathrm{MD}}$.

See book for proof. It's fairly straightforward.

RECAP: RANDOM ORACLES

Interesting:

It seems like hash functions behave like **random oracles!**

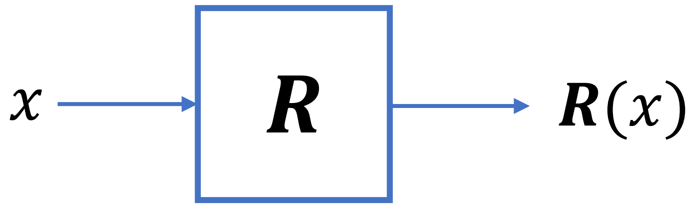
It's **as if** someone sampled a uniformly random function $\boldsymbol{R}$…

… and then put it in an oracle!

A strong hash function (like SHA-3) is developed and standardized.

LOOKS LIKE

1. Define $\boldsymbol{R}: \{0,1\}^{\mathrm{n}} \rightarrow \{0,1\}^n$ by setting $\boldsymbol{R}(x) \leftarrow \{0,1\}^n$ for each x.
2. Put $\boldsymbol{R}$ "into a box" so **everyone** can query it, but only as an oracle.



RECAP: RANDOM ORACLES ⇒ collision-resistant hashing

Random Oracle Model (ROM).

What crypto can we build in this model?

1. Define $\boldsymbol{R}: \{0,1\}^{n} \to \{0,1\}^{n}$
   by setting $\boldsymbol{R}(x) \leftarrow \{0,1\}^{n}$ for each x.

2. Put $\boldsymbol{R}$ "into a box" so **everyone**
   can query it, but only as an oracle.

$$x \longrightarrow \boxed{\boldsymbol{R}} \longrightarrow \boldsymbol{R}(x)$$

Collision-resistant hash:

- recall: random functions are collision-resistant;
- (because preimages are uniformly distributed)
- so $\boldsymbol{R}$ itself serves as a collision-resistant hash;
- if we want small outputs, can discard bits of output.

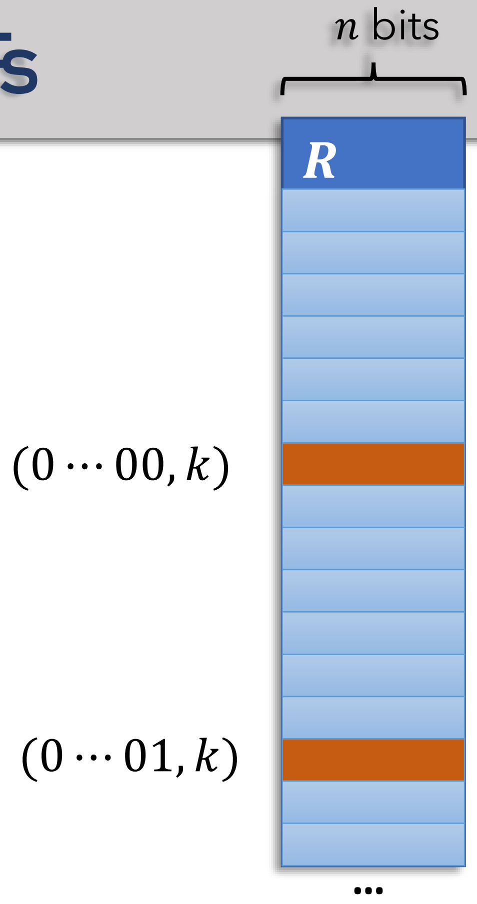
Note:

- this is now *statistical collision-resistance;*
- for normal hash functions, it was *computational* (i.e., against PPT adversaries.)
- remember: *collision-resistant* ⇒ *one-way*. So we also get **one-way functions!**

RECAP: RANDOM ORACLES ⇒ PRFs

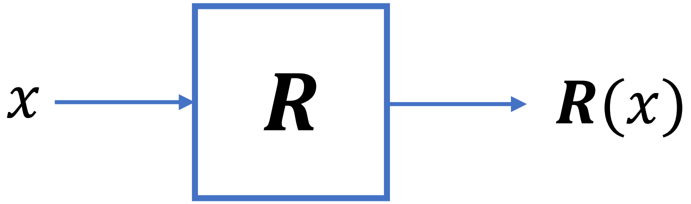
Random Oracle Model (ROM).

What crypto can we build in this model?



1. Define $\boldsymbol{R}: \{0,1\}^n \to \{0,1\}^n$
   by setting $\boldsymbol{R}(x) \leftarrow \{0,1\}^n$ for each x.

2. Put $\boldsymbol{R}$ "into a box" so **everyone**
   can query it, but only as an oracle.



Pseudorandom functions:

- sample a key: $k \leftarrow \{0,1\}^{n/2}$;
- define

$$\boldsymbol{F}_k: \{0,1\}^{n/2} \to \{0,1\}^{n/2}$$
$$\boldsymbol{F}_k(x) \coloneqq \boldsymbol{R}(x,k)$$

Why is it pseudorandom? Note: $\boldsymbol{A}$ knows $\boldsymbol{R}$!

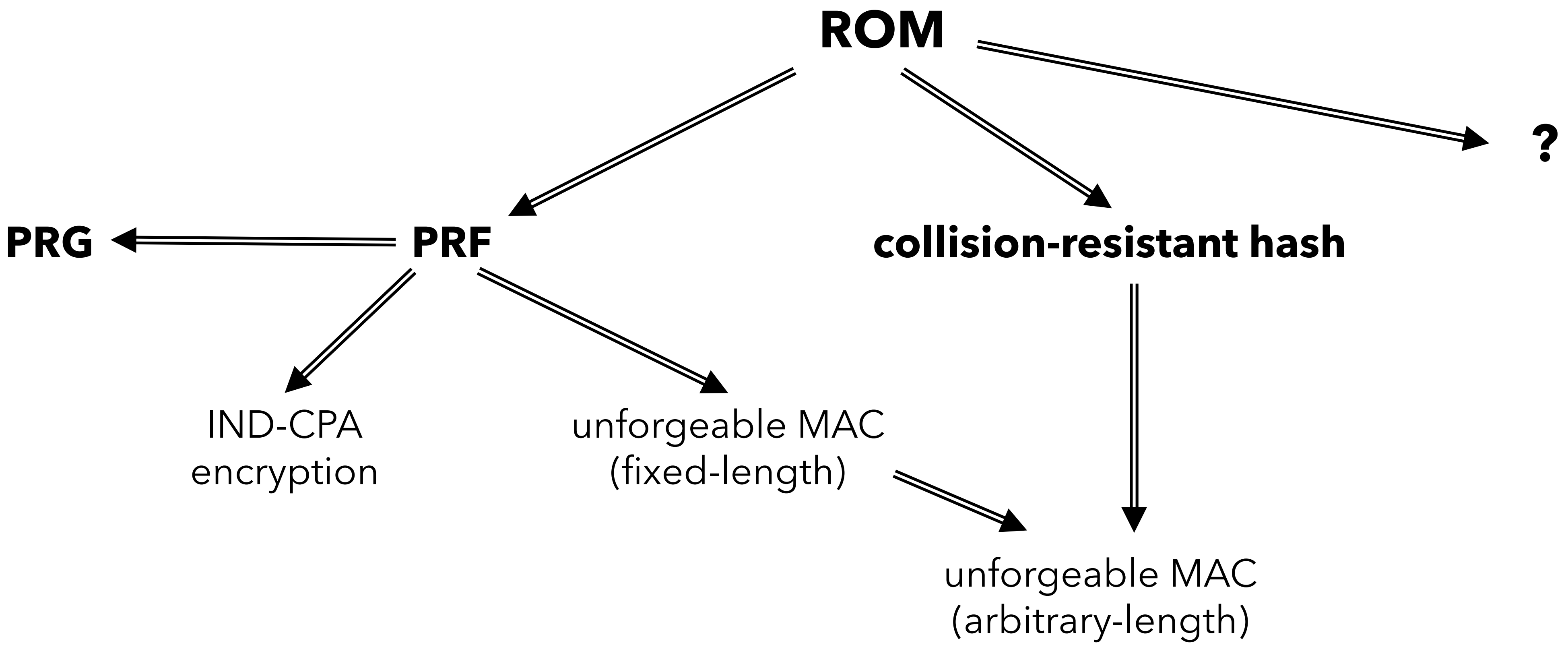
1. take any algorithm $\boldsymbol{A}^{\boldsymbol{F}_k}$. It makes some query x_1;
2. $\Pr[x_1 = (z,k)] = 2^{-n/2}$ for any z; so response is uniformly random in $\{0,1\}^{n/2}$;
3. in particular, $\boldsymbol{A}^{\boldsymbol{F}_k}$ learned nothing with the first query.
4. so we can repeat the argument starting from 1.

so $\boldsymbol{F}_k$ is oracle indistinguishable from a random function!

RECAP: RANDOM ORACLES ⇒ lots of stuff

Random Oracle Model (ROM).

What crypto can we build in this model?



RANDOM ORACLES ⇒ one-time authentication

IMPORTANT! NEW IDEAS!

Lamport scheme. One-time MAC for messages of length ℓ.

Let $\boldsymbol{R}: \{0,1\}^n \to \{0,1\}^n$ be a random oracle.

KeyGen:

I. Sample 2ℓ random inputs to $\boldsymbol{R}$:

- $x_1^0,\ x_2^0,\ x_3^0, \dots, x_\ell^0$.
- $x_1^1,\ x_2^1,\ x_3^1, \dots, x_\ell^1$.

Note each $x_j^b \in \{0,1\}^n$.

II. Now compute, for each j, b:

$$y_j^b \coloneqq \boldsymbol{R}(x_j^b);$$

III. Output key consisting of two parts:

1. $x_1^0,\ x_2^0,\ x_3^0, \dots, x_\ell^0$ and $x_1^1,\ x_2^1,\ x_3^1, \dots, x_\ell^1$;
2. $y_1^0,\ y_2^0,\ y_3^0, \dots, y_\ell^0$ and $y_1^1,\ y_2^1,\ y_3^1, \dots, y_\ell^1$;

0	x_1^0	x_2^0	x_3^0	…	x_ℓ^0
1	x_1^1	x_2^1	x_3^1	…	x_ℓ^1

$\downarrow$ $\boxed{\boldsymbol{R}}$ $\downarrow$

0	y_1^0	y_2^0	y_3^0	…	y_ℓ^0
1	y_1^1	y_2^1	y_3^1	…	y_ℓ^1

RANDOM ORACLES ⇒ one-time authentication

Lamport scheme. One-time MAC for messages of length ℓ.

Let $\boldsymbol{R}: \{0,1\}^n \to \{0,1\}^n$ be a random oracle.

Mac:

On input a message $m \in \{0,1\}^\ell$:

Output tag $t \in \{0,1\}^{n\ell}$ like this:

For each bit position $j = 1, 2, \dots, \ell$

output $x_j^{m_j}$.

Example:

Suppose $m = 010110$.

x_1^0		x_3^0			x_6^0
	x_2^1		x_4^1	x_5^1	

So tag is $(x_1^0, x_2^1, x_3^0, x_4^1, x_5^1, x_6^0)$.

Key

0	x_1^0	x_2^0	x_3^0	…	x_ℓ^0
1	x_1^1	x_2^1	x_3^1	…	x_ℓ^1

$\downarrow \boxed{\boldsymbol{R}} \downarrow$

0	y_1^0	y_2^0	y_3^0	…	y_ℓ^0
1	y_1^1	y_2^1	y_3^1	…	y_ℓ^1

RANDOM ORACLES ⇒ one-time authentication

Lamport scheme. One-time MAC for messages of length ℓ.

Let $\boldsymbol{R}: \{0,1\}^n \to \{0,1\}^n$ be a random oracle.

Ver:

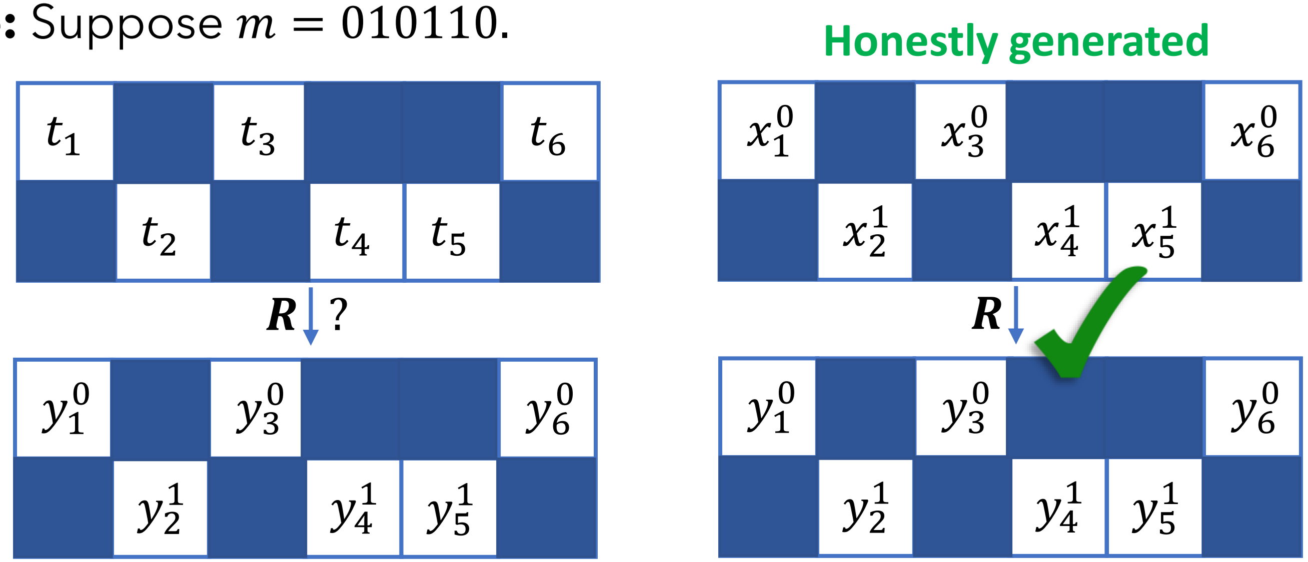
On input $m \in \{0,1\}^\ell$ and tag $(t_1, t_2, \ldots, t_\ell)$:

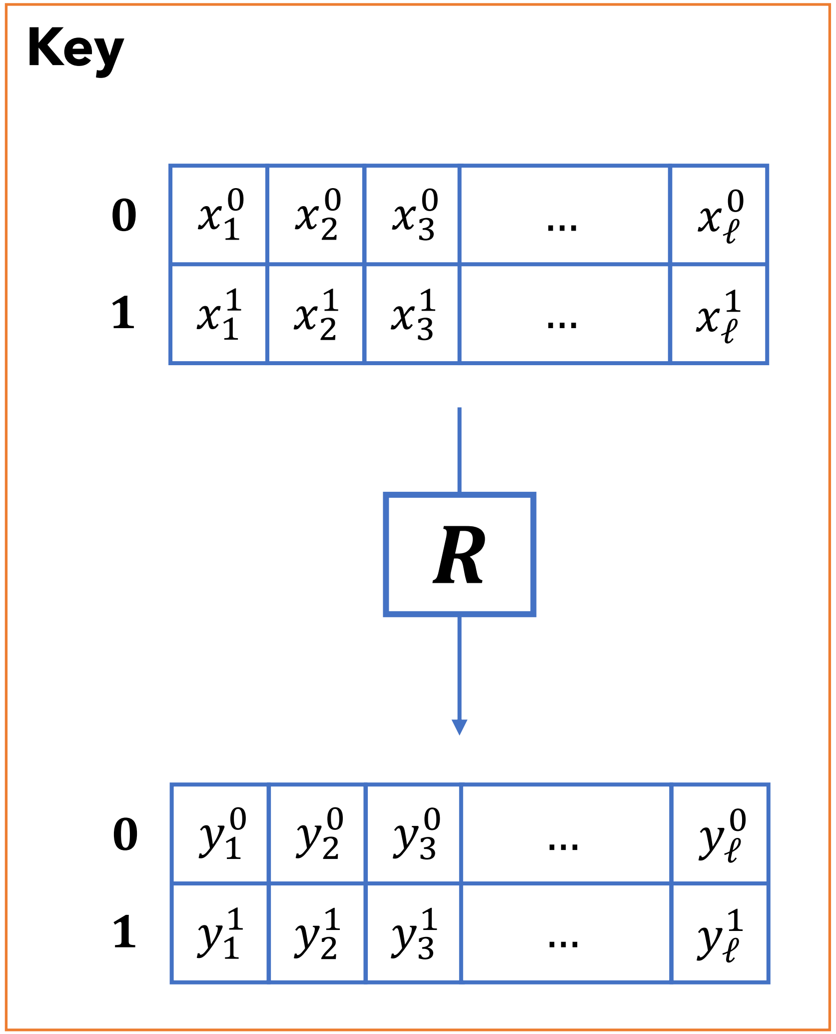
For each bit position $j = 1, 2, \ldots, \ell$:

If $\left(R(t_j) \neq y_j^{m_j}\right)$ output **reject;**

output **accept.**

Example: Suppose $m = 010110$.





RANDOM ORACLES ⇒ one-time authentication

Lamport scheme. One-time MAC for messages of length ℓ.

Let $\boldsymbol{R}: \{0,1\}^n \rightarrow \{0,1\}^n$ be a random oracle.

Check correctness:

- for message $m \in \{0,1\}^\ell$ ...
- … tag is $\left(x_1^{m_1}, x_2^{m_2}, x_3^{m_3}, \dots, x_\ell^{m_\ell}\right)$;
- at the verification stage, we do this check for each j:

$$\boldsymbol{R}\left(x_j^{m_j}\right) = y_j^{m_j}$$

- but in **KeyGen** this is exactly how we defined y_j^b for $b \in \{0,1\}$.
- so verification succeeds.

So scheme is correct. Is it unforgeable?

Key

0	x_1^0	x_2^0	x_3^0	…	x_ℓ^0
1	x_1^1	x_2^1	x_3^1	…	x_ℓ^1

$$\downarrow \boxed{\boldsymbol{R}} \downarrow$$

0	y_1^0	y_2^0	y_3^0	…	y_ℓ^0
1	y_1^1	y_2^1	y_3^1	…	y_ℓ^1

RANDOM ORACLES ⇒ one-time authentication

Lamport scheme. One-time MAC for messages of length ℓ.

Let $\boldsymbol{R}: \{0,1\}^n \to \{0,1\}^n$ be a random oracle.

So scheme is correct. Is it unforgeable?

Let's look at the adversary's view. It has two things:

$$m = m_0 m_1 m_2 \ldots m_\ell$$

$$t = \begin{array}{|c|c|c|c|c|c|} \hline x_1^0 & & x_3^0 & & & x_6^0 \\ \hline & x_2^1 & & x_4^1 & x_5^1 & \\ \hline \end{array}$$

Now adversary tries to forge on $m^* \neq m$.

There's a bit j where m^* differs from m. Say $j = 2$. Then…

$$m^* = m_0 m_1^* m_2 \ldots m_\ell$$

$$t^* = \begin{array}{|c|c|c|c|c|c|} \hline x_1^0 & \boxed{x_2^0} & & & & \\ \hline & & & x_4^1 & x_5^1 & \\ \hline \end{array}$$

But x_2^0 is random and unknown.

Key

0	x_1^0	x_2^0	x_3^0	…	x_ℓ^0
1	x_1^1	x_2^1	x_3^1	…	x_ℓ^1

$$\downarrow \boxed{\boldsymbol{R}} \downarrow$$

0	y_1^0	y_2^0	y_3^0	…	y_ℓ^0
1	y_1^1	y_2^1	y_3^1	…	y_ℓ^1

RANDOM ORACLES ⇒ one-time authentication

Lamport scheme. One-time MAC for messages of length ℓ.

Let $\boldsymbol{R}: \{0,1\}^n \rightarrow \{0,1\}^n$ be a random oracle.

So Lamport is a one-time MAC. So what?

Look at verification again:

Ver:

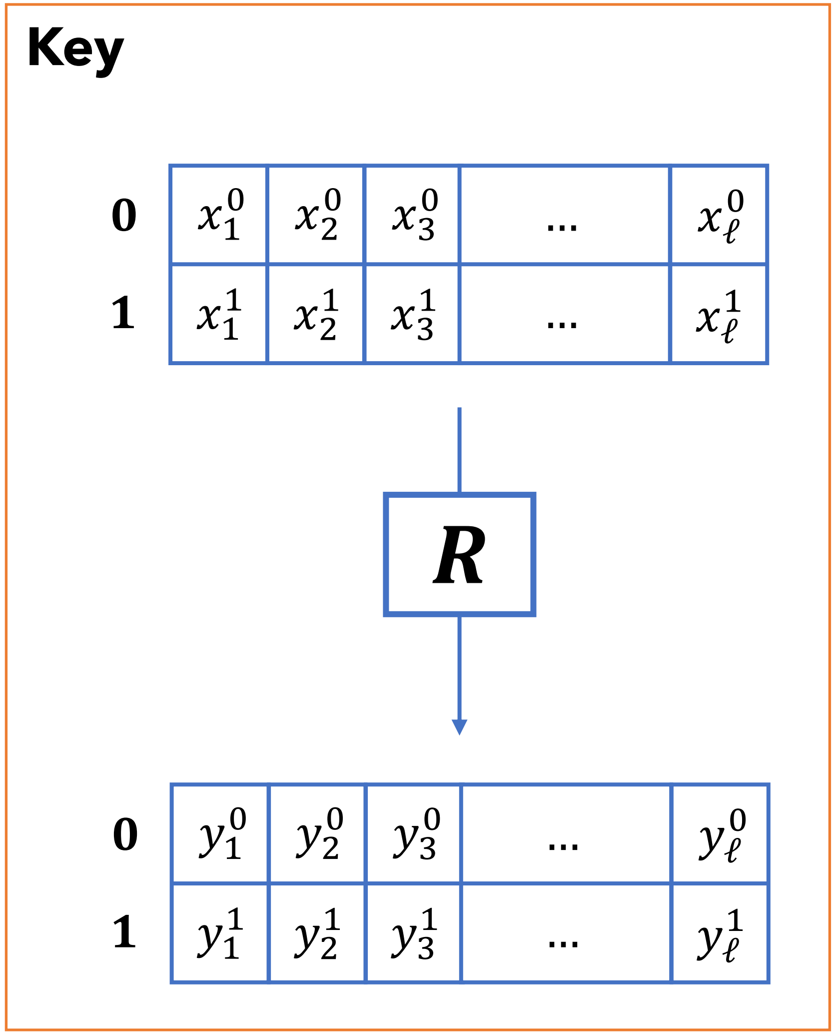
On input $m \in \{0,1\}^\ell$ and tag $(t_1, t_2, \ldots, t_\ell)$:

For each bit position $j = 1, 2, \ldots, \ell$:

If $\left(R(t_j) \neq y_j^{m_j}\right)$ output **reject;**

output **accept.**

It only requires knowledge of the y_j^b!



RANDOM ORACLES ⇒ one-time authentication

Lamport scheme. One-time MAC for messages of length ℓ.

Let $\boldsymbol{R}: \{0,1\}^n \to \{0,1\}^n$ be a random oracle.

So Lamport is a one-time MAC. So what?

Look at verification again:

Ver:

On input $m \in \{0,1\}^\ell$ and tag $(t_1, t_2, \ldots, t\_\ell)$:

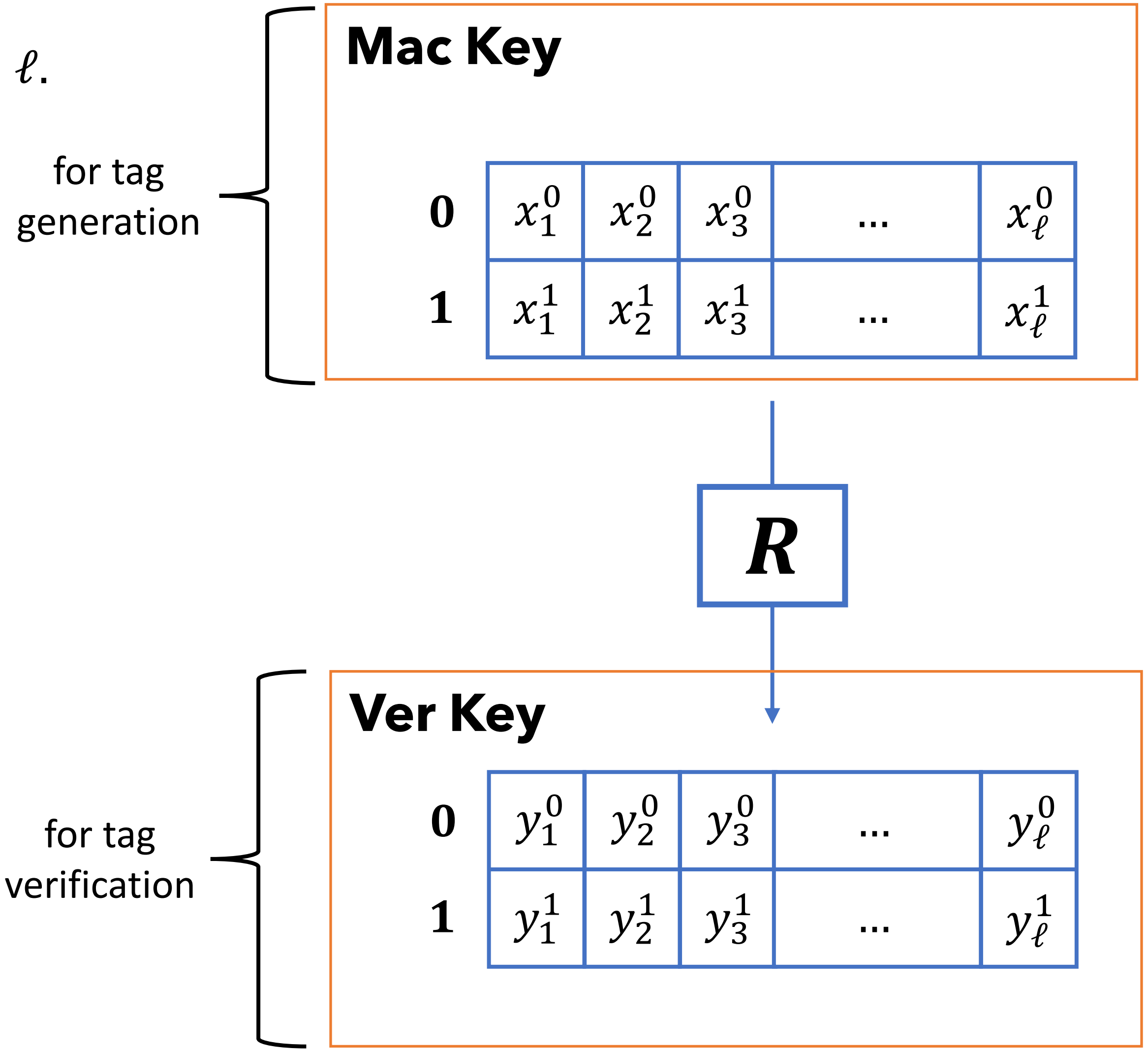
For each bit position $j = 1, 2, \ldots, \ell$:

If $\left(R(t_j) \neq y_j^{m_j}\right)$ output **reject;**

output **accept.**

It only requires knowledge of the y_j^b!

So can split key into a **Mac key**, and a **Ver key**.



RANDOM ORACLES ⇒ one-time authentication

Lamport scheme. One-time MAC for messages of length ℓ.

Let $\boldsymbol{R}: \{0,1\}^n \to \{0,1\}^n$ be a random oracle.

So Lamport is a one-time MAC…

With a separate **Mac key**, and a **Ver key**.

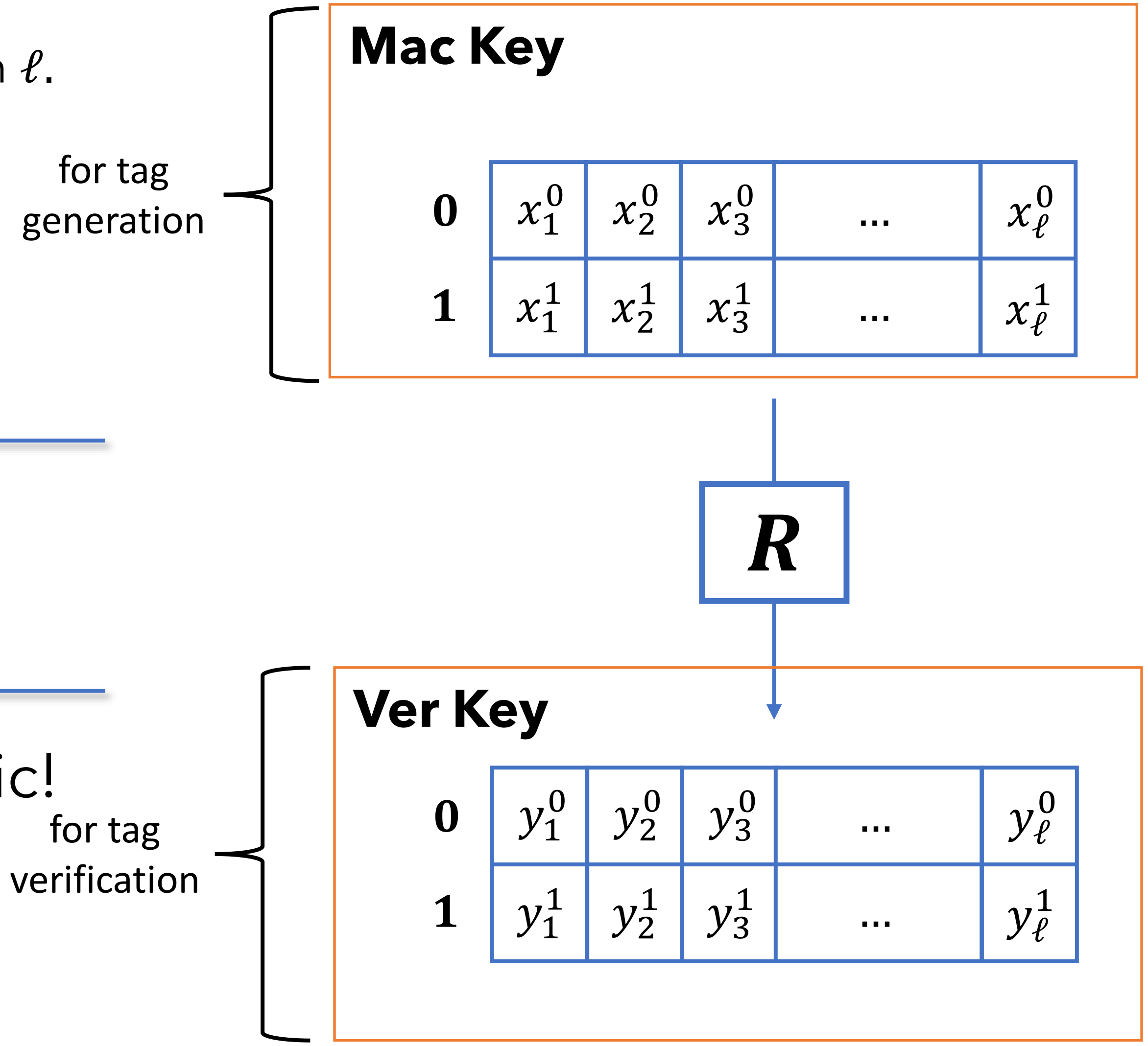
So what?

Security **only** rested on **unknowability** of the x_j^b.

So **only** the **Mac Key** needs to be kept secret!

In other words, we can make **Ver Key** public!

(**Mac Key** stays secure because $\boldsymbol{R}$ is one-way.)



DIGITAL SIGNATURES

Old setting: one key, shared privately.

Alice
k

(m, t)

Bob
k

New setting: private *signing* key, public *verification* key.

Alice
sk

(m, t)

vk

PUBLIC-KEY CRYPTOGRAPHY!

- **only Alice** can sign
- **anyone** can verify

LAMPORT SIGNATURE SCHEME

New setting: private *signing* key, public *verification* key.

Alice

x_1^0	x_2^0	x_3^0	…	x_ℓ^0
x_1^1	x_2^1	x_3^1	…	x_ℓ^1

$$\downarrow \boxed{\boldsymbol{R}} \downarrow$$

y_1^0	y_2^0	y_3^0	…	y_ℓ^0
y_1^1	y_2^1	y_3^1	…	y_ℓ^1

PUBLIC

$\boxed{\boldsymbol{R}}$

LAMPORT SIGNATURE SCHEME

New setting: private *signing* key, public *verification* key.

Alice

x_1^0	x_2^0	x_3^0	…	x_ℓ^0
x_1^1	x_2^1	x_3^1	…	x_ℓ^1

by one-way property of $\boldsymbol{R}$

y_1^0	y_2^0	y_3^0	…	y_ℓ^0
y_1^1	y_2^1	y_3^1	…	y_ℓ^1

PUBLIC

$\boxed{\boldsymbol{R}}$

LAMPORT SIGNATURE SCHEME

New setting: private *signing* key, public *verification* key.

Alice

x_1^0	x_2^0	x_3^0	…	x_ℓ^0
x_1^1	x_2^1	x_3^1	…	x_ℓ^1

$m = 0 \quad 1 \quad 0 \quad \ldots \quad 1 \in \{0,1\}^\ell$

x_1^0		x_3^0	…	
	x_2^1		…	x_ℓ^1

PUBLIC

y_1^0	y_2^0	y_3^0	…	y_ℓ^0
y_1^1	y_2^1	y_3^1	…	y_ℓ^1

$\boxed{\boldsymbol{R}}$

LAMPORT SIGNATURE SCHEME

New setting: private *signing* key, public *verification* key.

Alice

x_1^0	x_2^0	x_3^0	…	x_ℓ^0
x_1^1	x_2^1	x_3^1	…	x_ℓ^1

$m = \quad 0 \quad 1 \quad 0 \quad \ldots \quad 1 \in \{0,1\}^\ell$

x_1^0		x_3^0	…	
	x_2^1		…	x_ℓ^1

PUBLIC

y_1^0	y_2^0	y_3^0	…	y_ℓ^0
y_1^1	y_2^1	y_3^1	…	y_ℓ^1

$\boxed{\boldsymbol{R}}$

LAMPORT SIGNATURE SCHEME

New setting: private *signing* key, public *verification* key.

Alice

x_1^0	x_2^0	x_3^0	…	x_ℓ^0
x_1^1	x_2^1	x_3^1	…	x_ℓ^1

PUBLIC

y_1^0	y_2^0	y_3^0	…	y_ℓ^0
y_1^1	y_2^1	y_3^1	…	y_ℓ^1

$\boxed{\boldsymbol{R}}$

verification: check that $\boxed{\boldsymbol{R}}$

$m = 0 \quad 1 \quad 0 \quad \dots \quad 1 \in \{0,1\}^\ell$

x_1^0		x_3^0	…	
	x_2^1		…	x_ℓ^1

digital signature of document m

LAMPORT SIGNATURE SCHEME

Is it unforgeable?

Let's look at the adversary's view. It has three things:

$$m = m_0 m_1 m_2 \ldots m_\ell$$

$t =$

x_1^0		x_3^0			x_6^0
	x_2^1		x_4^1	x_5^1	

y_1^0	y_2^0	y_3^0	…	y_ℓ^0
y_1^1	y_2^1	y_3^1	…	y_ℓ^1

Now adversary tries to forge on $m^* \neq m$.

There's a bit j where m^* differs from m. Say $j = 2$. Then…

an inversion occurred here!

$$m^* = m_0 m_1^* m_2 \ldots m_\ell$$

$t^* =$

x_1^0	$\boxed{x_2^0}$	x_3^0			x_6^0
			x_4^1	x_5^1	

To do a formal proof:

- build an algorithm for inverting $\boldsymbol{R}$**…**
- … which internally simulates 1-EUF-CMA
- … with the Lamport scheme.

The inversion will be at any bit where the query and the forgery differ.

DIGITAL SIGNATURES

Definition. A digital signature scheme is a triple of PPT algorithms:

- (key generation) $\mathbf{KeyGen}$: on input 1^n, outputs a secret key sk and a public key vk;
- (tag generation) $\mathbf{Sign}$: on input a secret key sk and message $m \in \{0,1\}^*$, outputs signature $\mathbf{Sign}_{sk}(m)$;
- (verification) $\mathbf{Ver}$: on input a public key vk and a message-signature pair (m, s), outputs $\mathbf{1}$ or $\mathbf{0}$.

satisfying *correctness*: for all $(sk, vk) \leftarrow \mathbf{KeyGen}$ and all m, $\mathbf{Ver}_{vk}(m, \mathbf{Sign}_{sk}(m)) = \mathbf{1}$.

Compare to MACs:

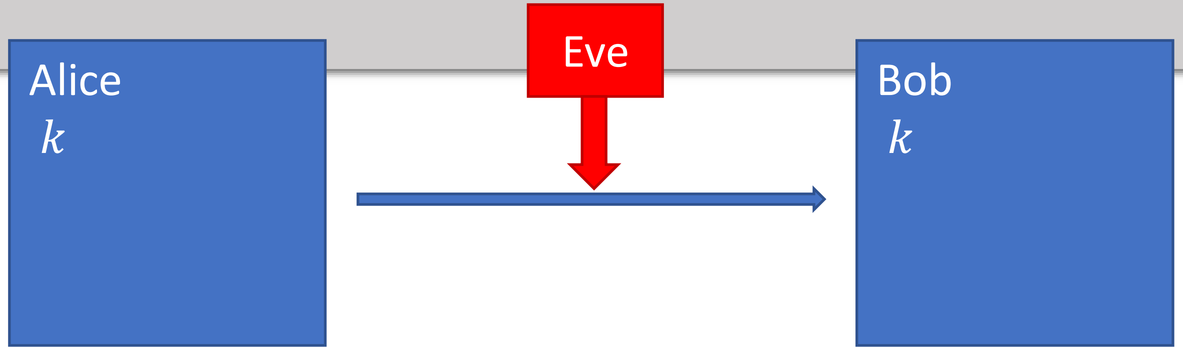
Definition. A message authentication code (MAC) is a triple of PPT algorithms:

- (key generation) $\mathbf{KeyGen}$: on input 1^n, outputs a key $k \in \{0,1\}^n$;
- (tag generation) $\mathbf{Mac}$: on input a key k and message $m \in \{0,1\}^*$, outputs a tag $\mathbf{Mac}_k(m)$;
- (verification) $\mathbf{Ver}$: on input a key k and a message-tag pair (m, t), outputs $\mathbf{1}$ or $\mathbf{0}$;

satisfying *correctness*: for all k and all m, $\mathbf{Ver}_k\big(m, \mathbf{Mac}_k(m)\big) = 1$.

PUBLIC-KEY CRYPTOGRAPHY
(a teaser)

SO FAR…

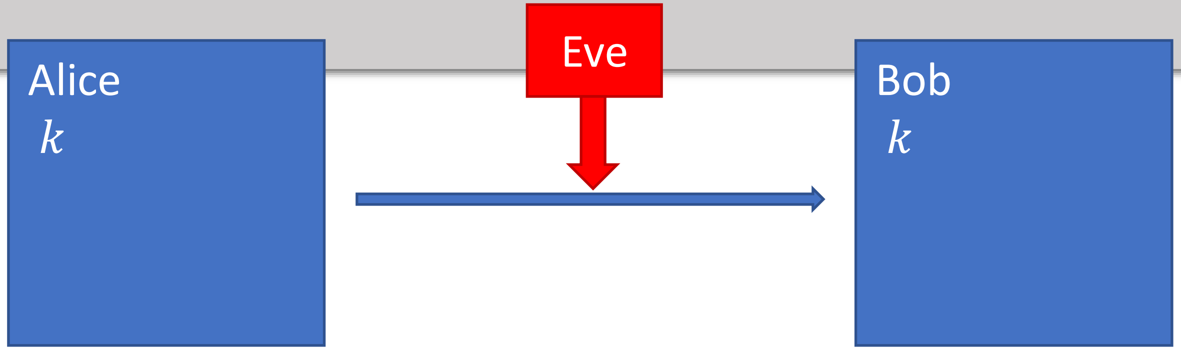


Until we saw Lamport…

- all schemes used shared secret keys;
- this comes with a problem: **how do you share the secret safely?**
- you can't use crypto…

- … so you're left with some physical method
- … and if that's intercepted (or searched without you knowing), all your crypto is pointless.

SO FAR…



There's other problems with secret keys…

Like **symmetry!**

Consider authentication:

- with a MAC, generating tags and verifying tags are coupled together;
- if Alice shares a key with Bob…
- … she doesn't **just** grant Bob the ability to verify the authenticity of her messages;
- … she **also** grants him the ability to authenticate them with her key!

So, to a third party, Bob could pretend to be Alice!

How do we solve that problem?

(By the way, this is also why secret-key crypto is sometimes called **symmetric-key crypto.**)

COMMUNICATION OVER PUBLIC CHANNELS?

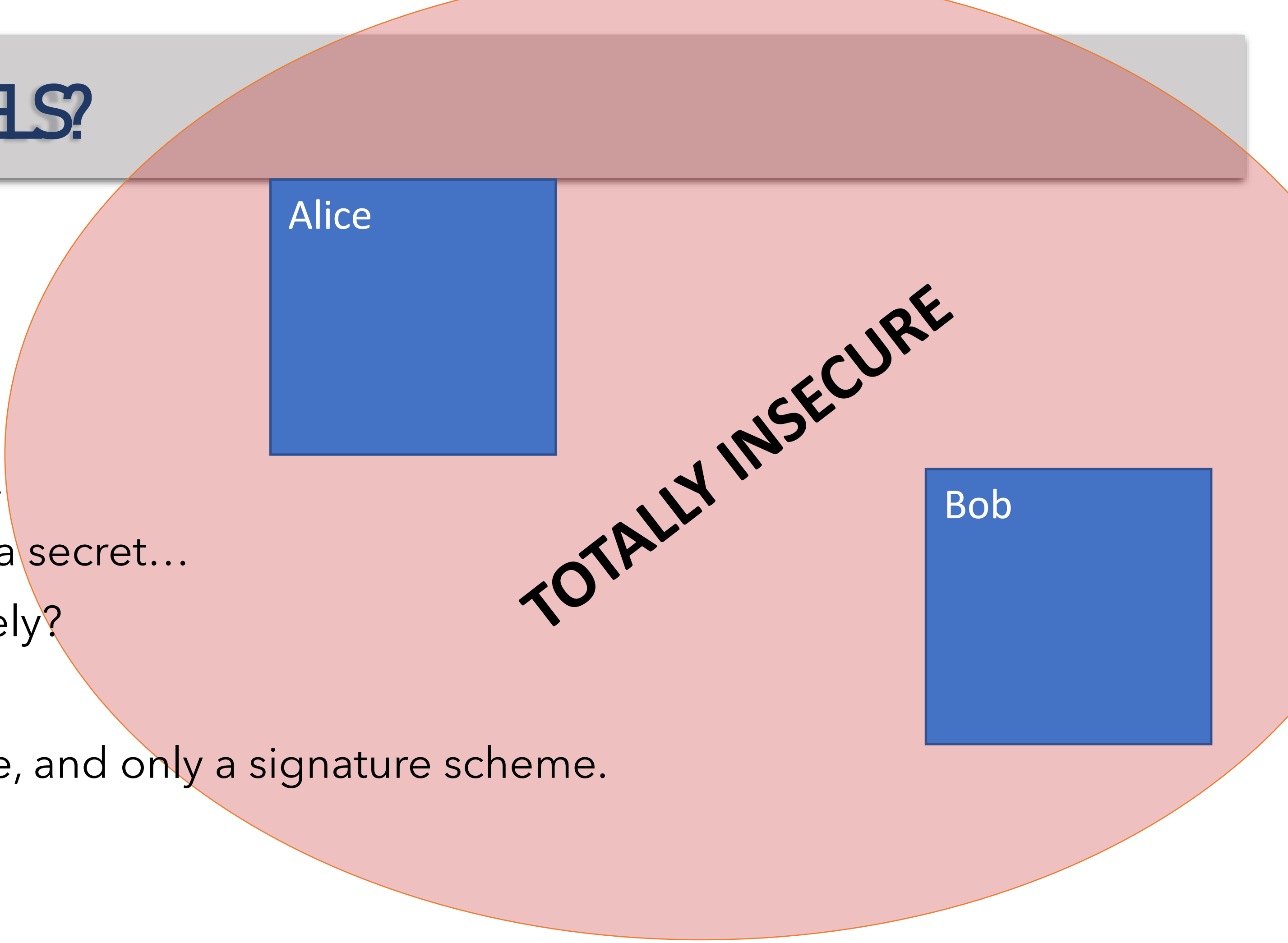
Maybe the problem is just unavoidable?

At first…

- it might seem like this is just how things are…
- after all, if you and another party don't share a secret…
- … how can you possibly communicate securely?

Lamport is certainly interesting, but it's one time, and only a signature scheme.

- what about encryption?
- and what about the key sharing problem?
- are these problems even solvable?



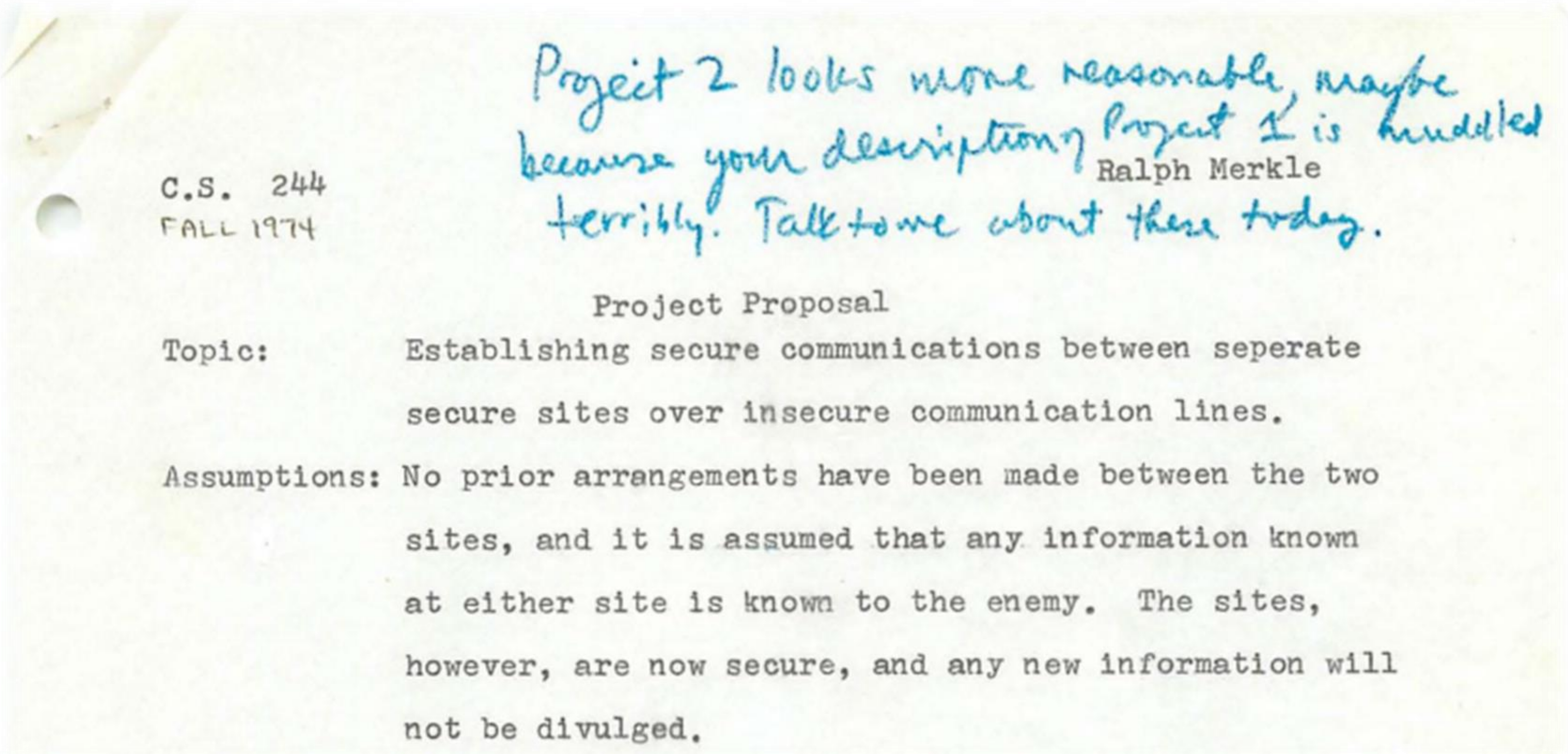
COMMUNICATION OVER PUBLIC CHANNELS?

If you think this sounds impossible…

… you're in good company! *(you could be a professor at Berkeley!)*

In 1974, an undergrad named Ralph Merkle took a security course at UC Berkeley.

For the course project, he proposed the following:



Project 2 looks more reasonable, maybe because your description of Project 1 is muddled terribly. Talk to me about these today.

C.S. 244
FALL 1974

Ralph Merkle

Project Proposal

Topic: Establishing secure communications between seperate secure sites over insecure communication lines.

Assumptions: No prior arrangements have been made between the two sites, and it is assumed that any information known at either site is known to the enemy. The sites, however, are now secure, and any new information will not be divulged.

COMMUNICATION OVER PUBLIC CHANNELS?

If you think this sounds impossible…

… you're in good company! *(you could be a professor at Berkeley!)*

In 1974, an undergrad named Ralph Merkle took a security course at UC Berkeley.

- for the course project, he proposed this exact problem;
- the prof told him to pick a different project…
- … so Merkle dropped the course, but continued working on his idea.

He eventually came up with something called "Merkle puzzles."

- it allowed two **honest** parties to share a secret over public channels in n timesteps…
- but any **adversary** who wanted to find the secret had to spend n^2 timesteps.

He wrote a paper, but it was rejected. The expert review said: *"Experience shows that it is extremely dangerous to transmit key information in the clear."*

COMMUNICATION OVER PUBLIC CHANNELS?

A few years earlier…

In 1969, cryptographers in GCHQ (British NSA) discovered a protocol for public-key crypto!
(This was not known to the public until 1997.)

In the public world…

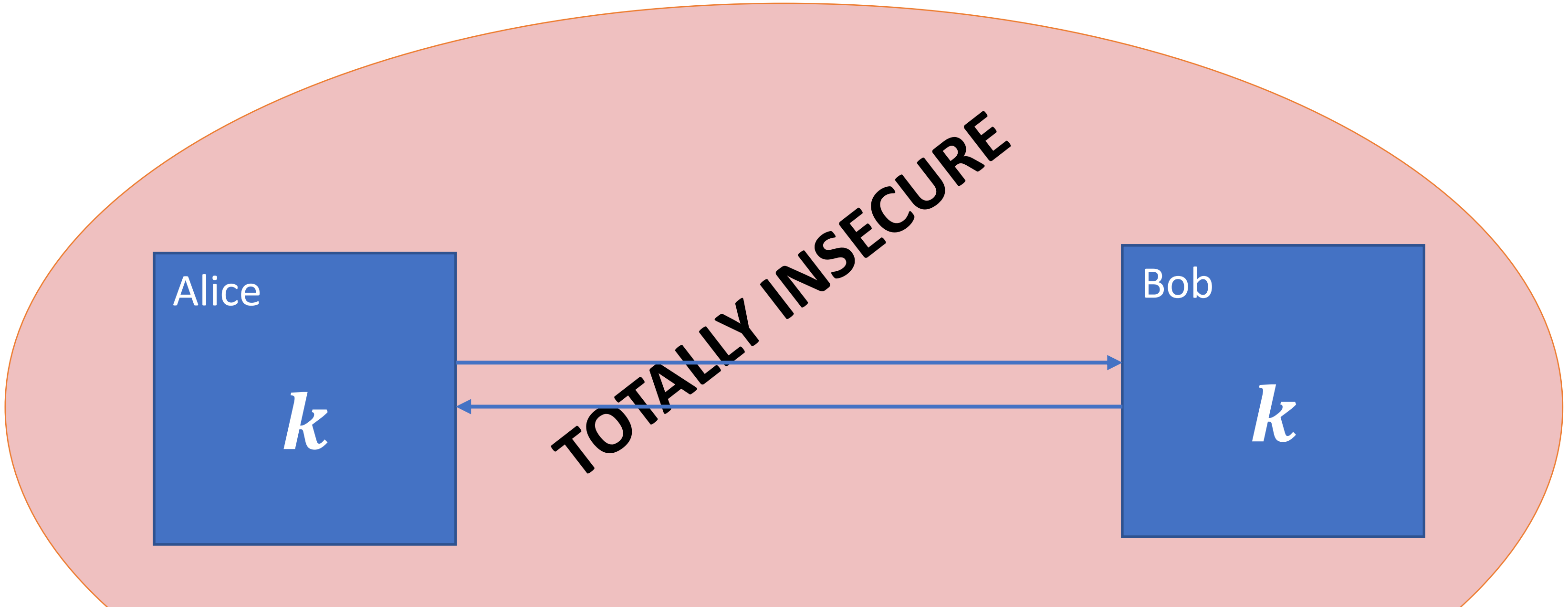
- two years after Merkle's discovery…
- in 1976, *Whitfield Diffie and Martin Hellman* discovered a **key-exchange protocol.**
- What does Diffie-Hellman key exchange do? Something magical!

KEY EXCHANGE

Diffie-Hellman key-exchange.

What does Diffie-Hellman key exchange do? Something magical!

- two parties are separated by an insecure channel (just like in encryption);
- *but they do not share any secret information!*
- and yet, after sending a few (insecure) messages in the open…
- they suddenly share a secret key!
- ***WHAT?***



KEY EXCHANGE → SECRET-KEY CRYPTO

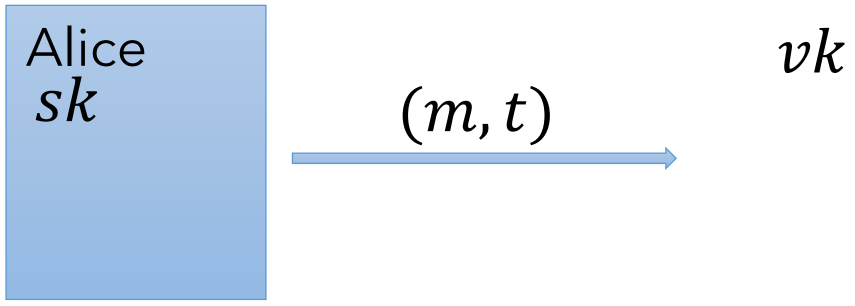
What then?

- after key exchange is performed…
- … we are now in the setting we assumed for:

1. secret-key encryption
2. message authentication

So: we can then use all the tools we learned about so far!

ASYMMETRIC CRYPTO



Is there another way?

Recall how Lamport worked.

- no key exchange required;
- one private key, one public key;
- asymmetric roles: private key enables signing, public key enables verification.

This can be extended to digital signatures for **any** number of messages!

What about encryption?

- can there be an asymmetric encryption scheme too?
- what's the right asymmetry?
- public key encrypts; private key decrypts.

ASYMMETRIC CRYPTO

Public-key encryption

Alice

dk ek

ASYMMETRIC CRYPTO

Public-key encryption

Public-key encryption

For two-way communication, Bob can do the same thing Alice did.

PUBLIC-KEY CRYPTOGRAPHY

Clearly…

… public-key crypto is awesome!

How do we get there?

- we don’t know how to build it out of “generic things”…
- … like PRGs or PRFs.
- we need specific computational assumptions…
- … and these assumptions require some math.

Specifically, some number theory.

So some work will be involved, but it will be very worthwhile!

THE PLAN

Clearly…

… public-key crypto is awesome!

How do we get there?

- we don't know how to build it out of "generic things"…
- … like PRGs or PRFs.
- we need specific computational assumptions…
- … and these assumptions require some math.

Specifically, some elementary number theory!

So some work will be involved, but it will be very worthwhile!

THE PLAN

Next 6 weeks:

Topic	Dates
Intro and symmetric-key cryptography (8 lectures)	January 28 – February 20
RSA and Diffie-Hellman (4 lectures **+ 1 hwk**) ***Carl Miller***	February 25 – March 5
Secret sharing (2 lectures **+ 1 hwk**) ***Bill Gasarch***	March 10 - 12
Fun guest lecture	March 24
Midterm review	March 26
Midterm	**March 31**
Fun guest lecture (blockchain, likely)	April 2
I’m back.	April 7 - end