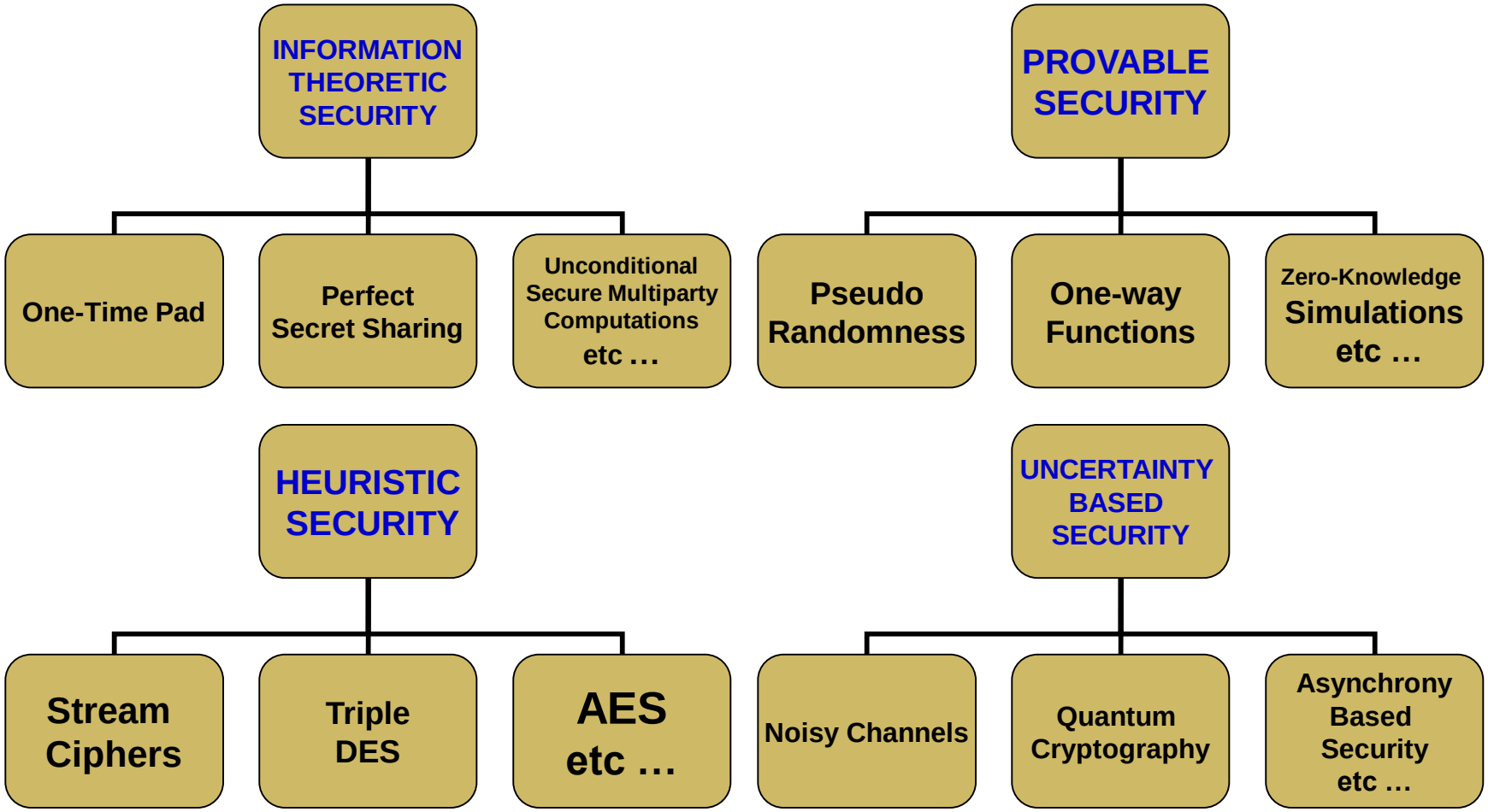
PRINCIPLES OF INFORMATION SECURITY

The Subject Matter, Basic Pre-requisites and Design Challenges

THE SUBJECT MATTER

Today's Multi-verses of Security

The Four Silos of Contemporary Security



BUT WHY DIFFERENT TYPES OF SECURITY?

- ***P = NP*** : A world where it is *impossible* to pose questions (with easily demonstrable solutions) that are hard to answer.
- ***P \neq NP (but only in worst-case)*** : A world where it is *infeasible* to pose questions (with easily demonstrable solutions) that are hard to answer.
- ***Average-case hard problems, but no one-way functions*** : A world where it is *feasible* to pose questions (with easily demonstrable solutions) that are hard to answer (but infeasible for the poser himself/herself to solve it!).
- ***One-way Functions but no PKC*** : A world where it is *easy* to pose questions (with easily demonstrable solutions) that are hard to answer.
- ***Public-Key Cryptography*** : It is *easy* to pose questions (with easily demonstrable solutions) that are hard to answer *except* for a selected group.

WHY IS HEAVY-DUTY MATHEMATICS NEEDED FOR SECURITY?

Kerckhoff's Principle

Advantages of publishing

Kerckhoff's Principle

Dr. Auguste Kerckhoff
1883

"a cryptosystem should be secure even if everything about the system, except the key, is public knowledge"

Why should it be made public?

- Reverse Engineering
- Costly Storage
- Ease of replacement
- Multiple Users

- Ethical Hacking
- Standards

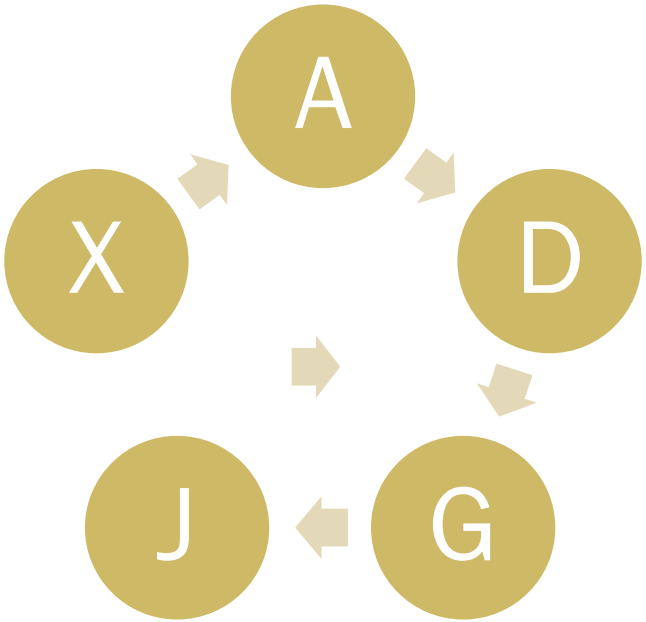
CRYPTANALYSIS OF HISTORICAL CIPHERS

Shift Cipher, Mono-alphabetic Substitution Cipher and Vigenere Cipher

The Shift Cipher and the Sufficient Key Space Principle

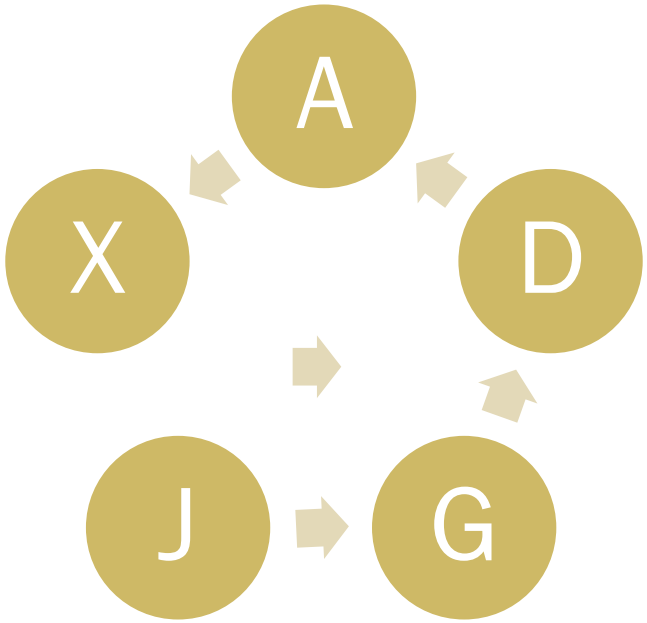
SHIFT CIPHER

- Rotate each letter by the key k
- For example, if k is 3 then:



Encryption

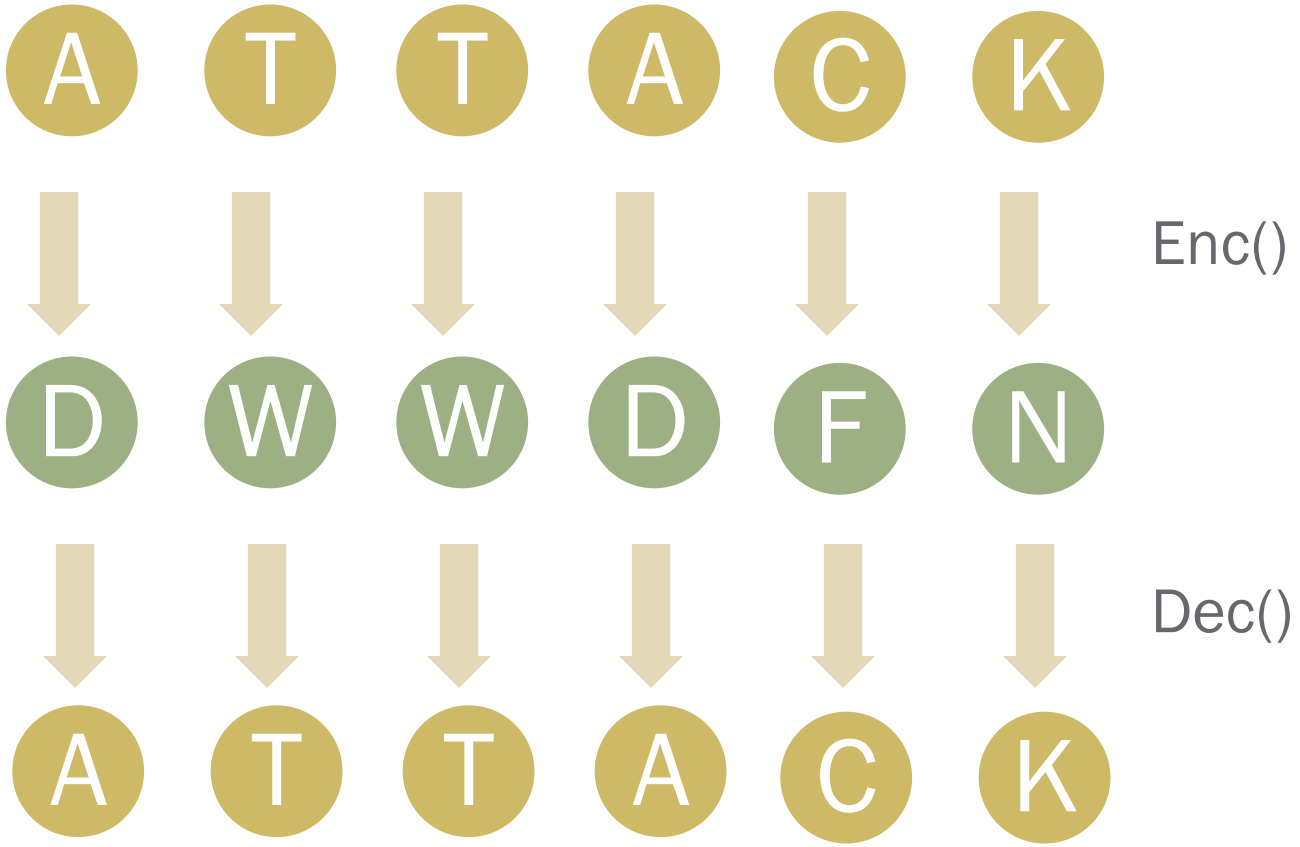
Enc(x) = (x + k) mod 26.



Decryption

Dec(x) = (x - k) mod 26

Example: Key = 3 and Plaintext = “ATTACK”



Problem with Shift ciphers

- Not enough keys!
- If we shift a letter 26 times, we get the same letter back.
  - A shift of 27 is the same as a shift of 1, etc.
  - So we only have 25 keys (1 to 25).
- Therefore, easy to attack via brute force.

Example: Cryptanalysis of shift ciphers

- Cipher text : OVDTHUFWVZZPISLRLFZHYLAOLYL

Key Value	Possible Plain Text
1	NUCSGTEVUYYOHRKQKEYGXKZNKXK
2	MTBRFSDUTXXNGQJPJDXFWJYMJWJ
3	LSAQERCTSWWMFPIOICWEVIXLIVI
4	KRZPDQBSRVVLEOHNHBVDUHWKHUH
5	JQYOCPARQUUKDNGMGAUCTGVJGTG
6	IPXNBOZQPTTJCMFLFZTBSFUIFSF
7	HOWMANYPOSSIBLEKEYSARETHERE
8	GNVLZMXONRRHAKDJDXRZQDSGDQD
9	FMUKYLWNMQQGZJCICWQYPCRFCPC
10	ELTJXKVMLPPFYIBHBVPXOBQEBOB
11	DKSIWJULKOOEXHAGAUOWNAPDANA
12	CJRHVITKJNNDWGZFZTNVMZOCZMZ
13	BIQGUHSJIMMCVFYEYSMULYNBYLY

Mono-Alphabetic Substitution Cipher: Just Large Key Space Isn't Enough!

Mono Alphabetic Substitution Cipher

Each character in the plain text

Mapped to →

Unique character in the Cipher Text

→ A total of 26! Keys

Plain Text : “Tell him about me”

Assume the following mapping:

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
F	A	Y	E	Z	C	S	G	V	H	W	Q	B	D	J	U	I	L	K	M	X	R	T	N	O	P

The plaintext would be encrypted as follows: (ignore the spaces)

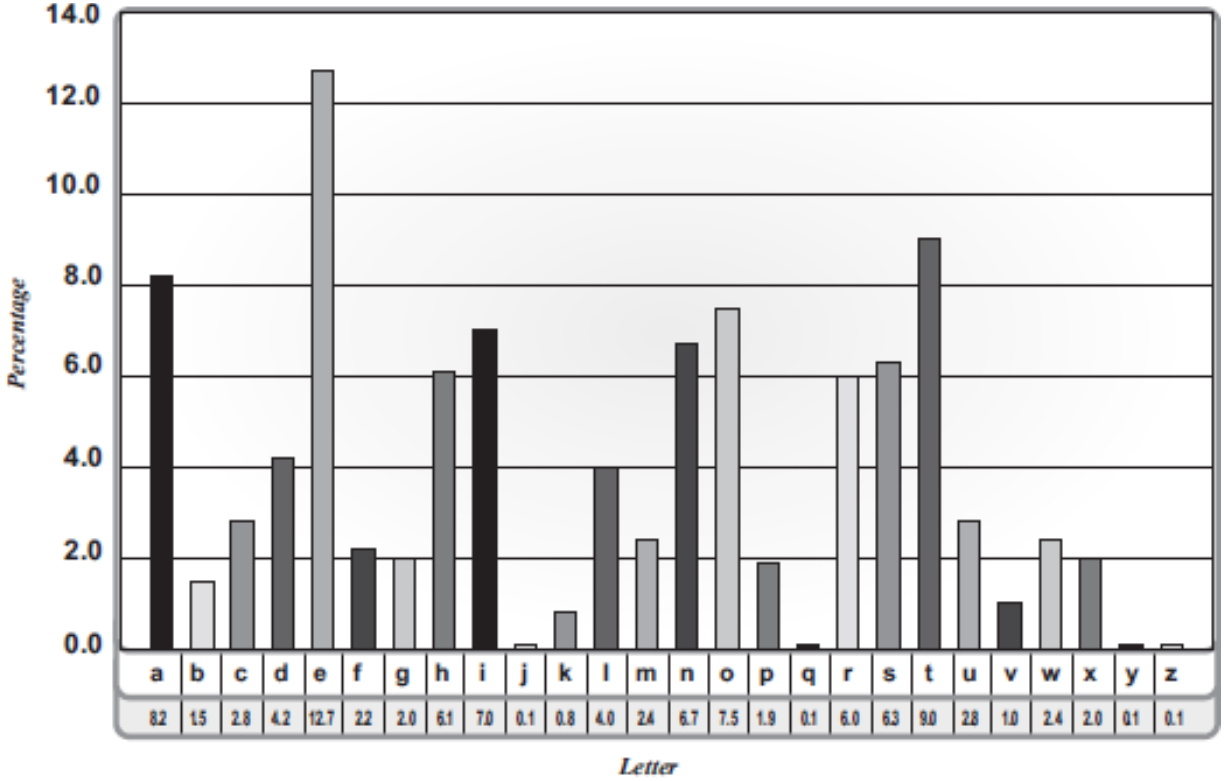
t	e	l	l		h	i	m		a	b	o	u	t		m	e
M	Z	Q	Q		G	V	B		F	A	J	X	M		B	Z

Frequency Analysis

Consider the cipher:

UZQSOVUOHXMOPVGPOZPEVSGZWSZOPFPESXUDBMETSXAIZVUEPHZHMDZSHZOWSF
PAPPDTSVPQUZWYMXUZUHSXEPYEPOPDZSZUFPOMBZWPFUPZHMDJUDTMOHMQ

Standard Frequency Distribution of English Alphabets:



Frequency of alphabets in the cipher text:

P 13.33	H 5.83	F 3.33	B 1.67	C 0.00
Z 11.67	D 5.00	W 3.33	G 1.67	K 0.00
S 8.33	E 5.00	Q 2.50	Y 1.67	L 0.00
U 8.33	V 4.17	T 2.50	I 0.83	N 0.00
O 7.50	X 4.17	A 1.67	J 0.83	R 0.00
M 6.67				

{P,Z} Є {e,t}

{S,U,O,M,H} Є {a,h,I,n,o,r,s}

{A,B,G,Y,I,J} Є {b,j,k,q,v,x,z}

What is the actual mapping?

The Plaintext (after breaking it)

it was disclosed yesterday that several informal but direct contacts have been made with political representatives of the viet cong in moscow

A Statistical Approach to Break Shift Cipher

p_i : Probability of i^{th} character in plaintext

$$\sum_{i=0}^{25} p_i^2 \approx 0.065$$

q_i : Probability of i^{th} character in ciphertext

$$I_j \stackrel{\text{def}}{=} \sum_{i=0}^{25} p_i \cdot q_{i+j}$$

Find k such that I_k is nearly 0.065

Vigenere Cipher: Code Complexity Does Not Guarantee Security!

Vigenere Cipher

Poly alphabetic substitution

Each character in the plain text → Mapped to → Any character in the Cipher Text

Blaise de Vigenere

Try breaking my cipher !

Encryption

Periodic key: deceptivedeceptivedeceptive

Plaintext: wearediscoveredsaveyourself

Ciphertext: ZICVTWQNGRZGVTWAVZHCQYGLMGJ

d	e	c	e	p	t	i	v	e	d	e	c	e	p	t	i	v	e	d	e	c	e	p	t	i	v	e
w	e	a	r	e	d	i	s	c	o	v	e	r	e	d	s	a	v	e	y	o	u	r	s	e	l	f
Z	I	C	V	T	W	Q	N	G	R	Z	G	V	T	W	A	V	Z	H	C	Q	Y	G	L	M	G	J

Cryptanalysis of Vigenere Cipher

Brute force → Infeasible !

Frequency Analysis → Information is Obscured !

Unbreakable ?

NO !

Still there is scope for Frequency Analysis !

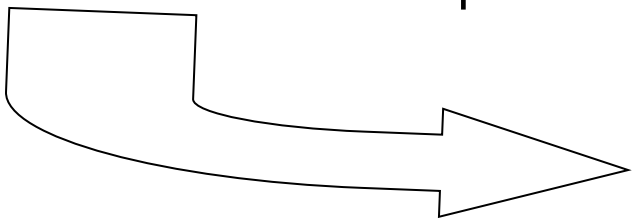
Not all information is lost !

Attacker

Cryptanalysis (Contd.)

1. Find the Key Length

Two Steps:



2. Find the Key

A Statistical Approach to Vigenere Cipher

p_i : Probability of i^{th} character in plaintext

$$\sum_{i=0}^{25} p_i^2 \approx 0.065$$

q_i : Probability of i^{th} character in the cipher sub-sequence

$$c_1, c_{1+\tau}, c_{1+2\tau}, \ldots$$

$$S_\tau \stackrel{\text{def}}{=} \sum_{i=0}^{25} q_i^2$$

Find key length t such that S_t is nearly 0.065

Once key length t is found, Vigenere cipher is nothing but a collection of t shift ciphers!

DESIGNING SECURE SYSTEMS IS HARD

The entry of SHANNON's Genius (next class!)

THANK YOU

Any Questions?

SHANNON’s THEORY

PERFECT SECRECY

ENCRYPTION SCHEME (Generic Definition)

- Three Algorithms
  - Key Generation (***Gen***)
  - Encryption (***Enc***)
  - Decryption (***Dec***)
- Message Space ***M***

Perfect Secrecy

- Adversary must not obtain any *additional* information about the message, due to the ciphertext

For any message m and any valid ciphertext c,

$$\Pr[\text{Message} = m \mid \text{Ciphertext} = c] = \Pr[\text{Message} = m]$$

PERFECTLY SECRET ENCRYPTION (textbook definition)

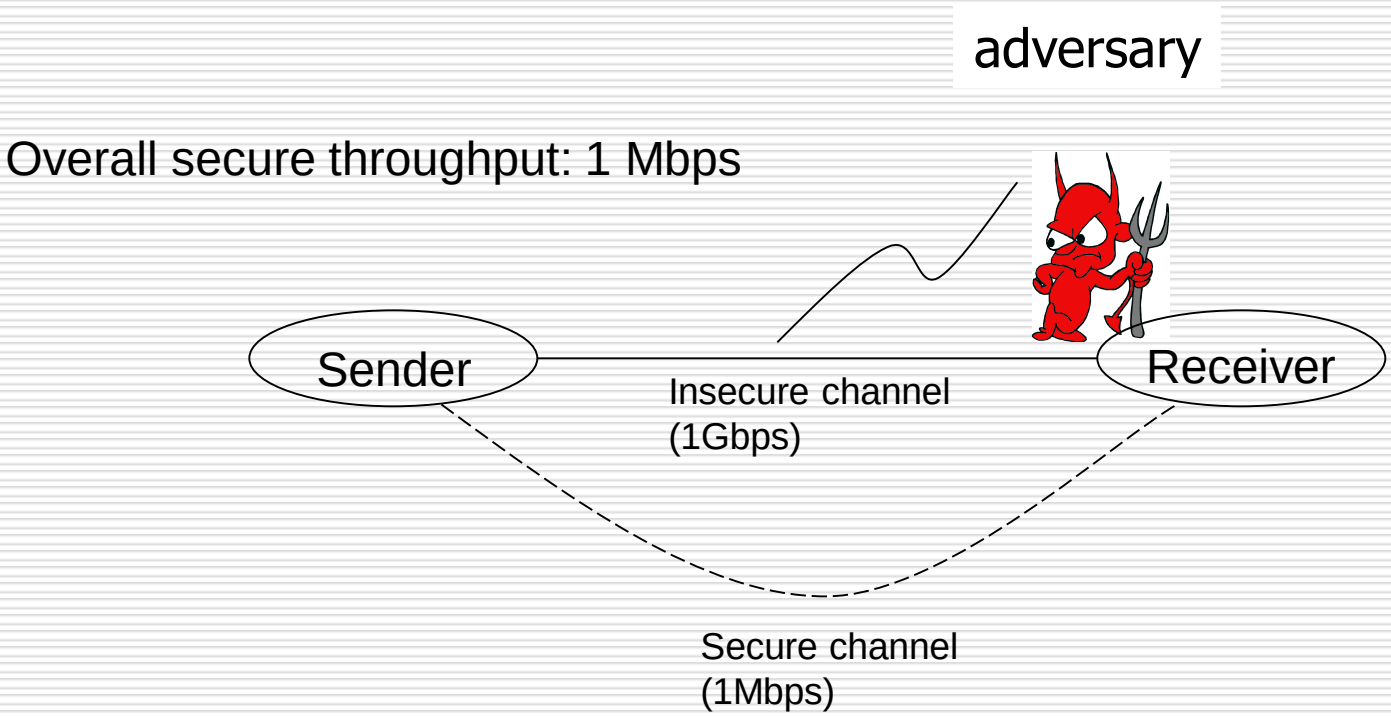
DEFINITION 2.1 *An encryption scheme* (Gen, Enc, Dec) *over a message space* $\mathcal{M}$ *is* perfectly secret *if for every probability distribution over* $\mathcal{M}$*, every message* $m \in \mathcal{M}$*, and every ciphertext* $c \in \mathcal{C}$ *for which* $\Pr[C = c] > 0$*:*

$$\Pr[M = m \mid C = c] = \Pr[M = m].$$

Achieving Perfect Secrecy is Impractical

Shannon proved that for perfect secrecy the size of the (and the entropy of) key space be at least as much as the size of (and the entropy of) the message space

Why is it Impractical?



- The secure communication throughput of the entire system is same as the cumulative bandwidth of truly secure channels in the system

The Issue

- Fast secure channels are required for efficient secure communication

"Chicken-and-Egg" problem!

EQUIVALENT STATEMENTS of PERFECT SECRECY

LEMMA 2.2 *An encryption scheme* (Gen, Enc, Dec) *over a message space* $\mathcal{M}$ *is perfectly secret if and only if for every probability distribution over* $\mathcal{M}$, *every message* $m \in \mathcal{M}$, *and every ciphertext* $c \in \mathcal{C}$:

$$\Pr[C = c \mid M = m] = \Pr[C = c].$$

PROOF

If:

$$\Pr[C = c \mid M = m] = \Pr[C = c]$$

Then, multiplying both sides by: $\Pr[M = m] / \Pr[C = c]$

$$\frac{\Pr[C = c \mid M = m] \cdot \Pr[M = m]}{\Pr[C = c]} = \Pr[M = m]$$

Using Bayes' Theorem, L.H.S. is: $\Pr[M = m \mid C = c]$.

Similarly, the other direction.

PERFECT INDISTINGUISHABILITY

LEMMA 2.3 *An encryption scheme* (Gen, Enc, Dec) *over a message space* $\mathcal{M}$ *is perfectly secret if and only if for every probability distribution over* $\mathcal{M}$, *every* $m_0, m_1 \in \mathcal{M}$, *and every* $c \in \mathcal{C}$:

$$\Pr[C = c \mid M = m_0] = \Pr[C = c \mid M = m_1].$$

PROOF

One direction follows from previous Lemma.

For the other direction, fix p = $\Pr[C = c \mid M = m_0]$

$$\Pr[C = c] = \sum_{m \in \mathcal{M}} \Pr[C = c \mid M = m] \cdot \Pr[M = m]$$

$$= \sum_{m \in \mathcal{M}} p \cdot \Pr[M = m]$$

$$= p \cdot \sum_{m \in \mathcal{M}} \Pr[M = m] \qquad = p$$

ONE-TIME PAD (VERNAM CIPHER)

- Message Space = Key Space = Cipher space = $\{0,1\}^n$
- ***Gen:*** Choose an n-bit key uniformly at random.
- ***Enc:*** c = m xor key
- ***Dec:*** m = c xor key

Vernam Cipher is perfect

$$\Pr[C = c \mid M = m] = \Pr[M \oplus K = c \mid M = m]$$

$$= \Pr[m \oplus K = c] = \Pr[K = m \oplus c]$$

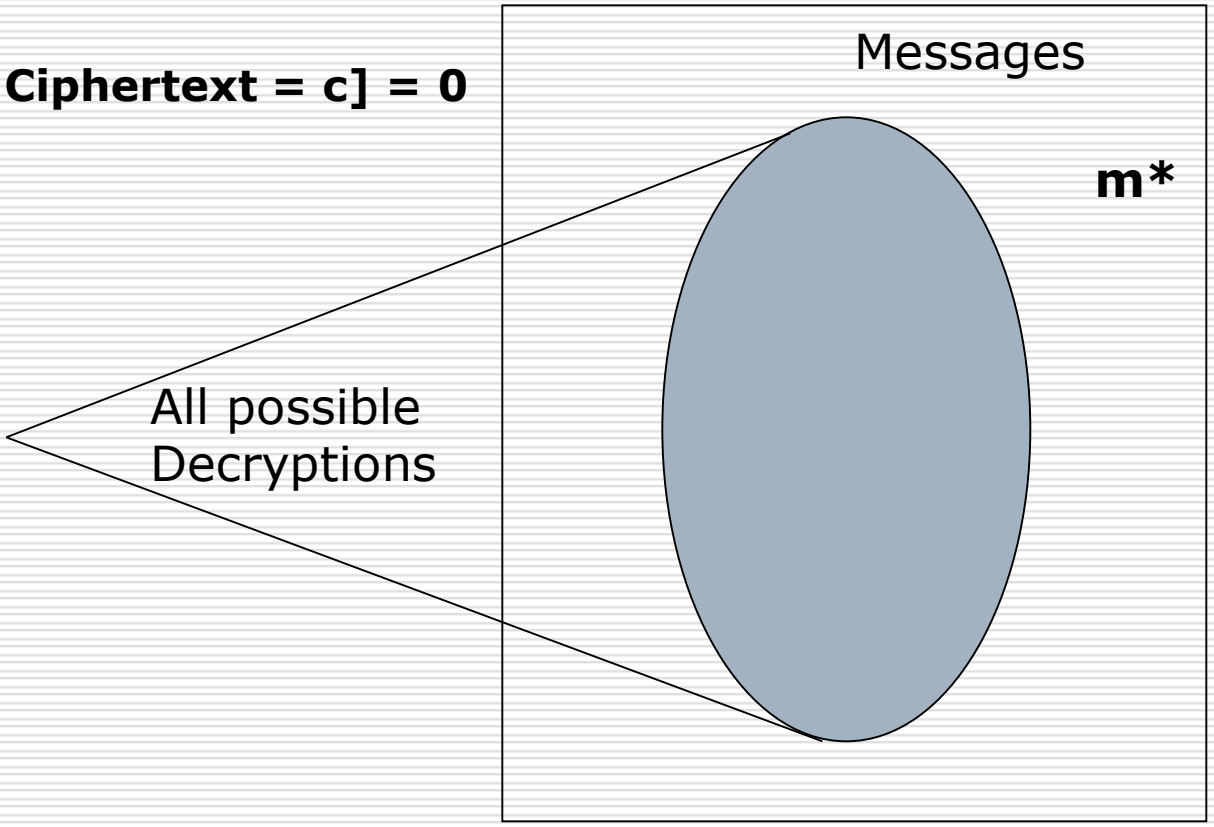
$$= 1/2^n$$

LIMITATIONS OF PERFECT SECRECY

Key space is at least as large as the message space

If More Messages than Keys then it CANNOT be PERFECT!

P[Message = m* | Ciphertext = c] = 0



(Next Class)
Two Relaxations to Perfect Secrecy

- Security is only preserved against efficient adversaries that run in a feasible amount of time

- Adversaries can potentially succeed with some very small probability

These two relaxations are necessary and (if certain interesting mathematical objects exist) are also sufficient

THANK YOU

Any Questions?

PSEUDORANDOMNESS

And One-way Functions

(RECALL)
Two Relaxations to Perfect Secrecy

- Security is only preserved against efficient adversaries that run in a feasible amount of time

- Adversaries can potentially succeed with some very small probability

These two relaxations are necessary and (if certain interesting mathematical objects exist) are also sufficient

DEFINING "EFFICIENT" ADVERSARY

- Borrowing from Tenets of Theoretical Computer Science:
  - Efficiency = **PROBABILISTIC POLYNOMIAL TIME STRATEGIES (PPT)**

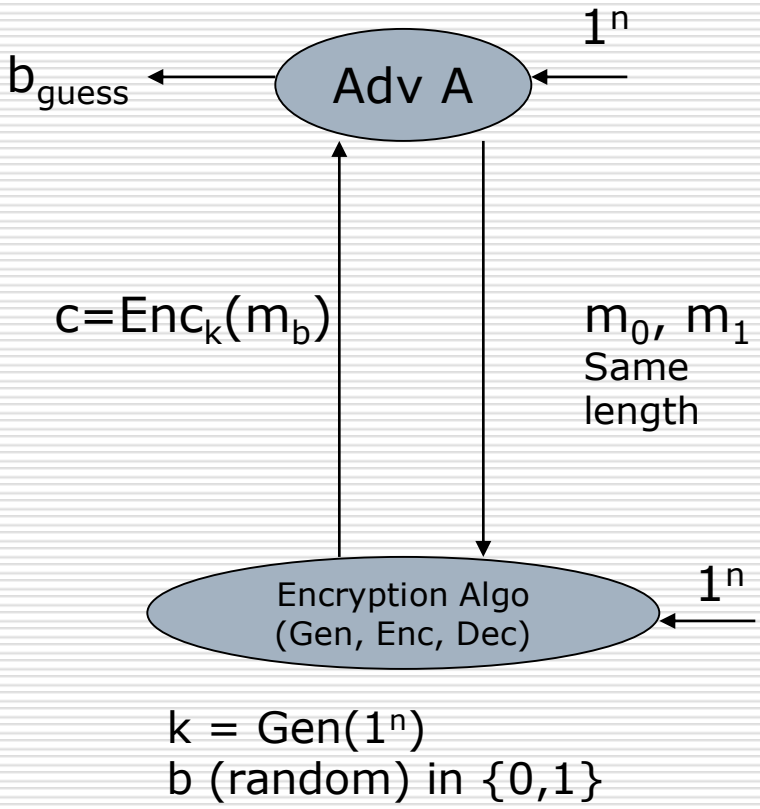
DEFINING "NEGLIGIBLE" PROBABILITY

- **Negligible Functions (asymptotically) grow slower than the inverse of any polynomial.**

Textbook Definition

DEFINITION 3.4 *A function f is* negligible *if for every polynomial $p(\cdot)$ there exists an N such that for all integers $n > N$ it holds that $f(n) < \frac{1}{p(n)}$.*

Defining Computational Security Against Eavesdroppers



Cryptosystem is secure against eavesdroppers if for all PPT adversaries A:

$P[b_{guess} = b] <= ½ + negl(n)$

PSEUDORANDOM GENERATORS

And computational
indistinguishability

Pseudo Random Generator (PRG)

A *deterministic polynomial time algorithm* G, inputs n bits and outputs l(n) bits where:

(a) $l(n) > n$

And

(b) Output of G is computationally indistinguishable from uniform distribution

PRG
(Textbook Definition)

DEFINITION 3.14 *Let $\ell(\cdot)$ be a polynomial and let G be a deterministic polynomial-time algorithm such that for any input $s \in \{0,1\}^n$, algorithm G outputs a string of length $\ell(n)$. We say that G is a* pseudorandom generator *if the following two conditions hold:*

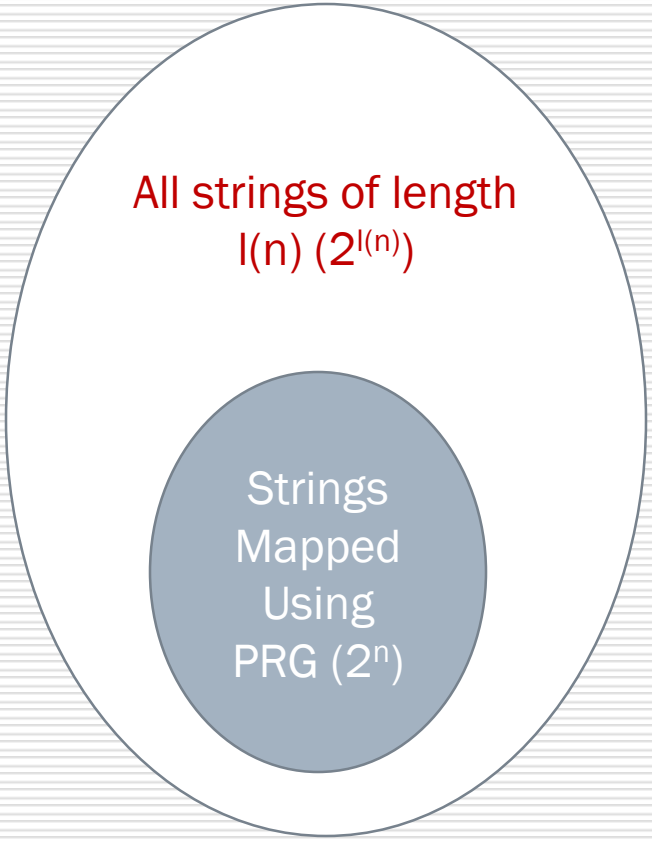
1. (Expansion:) *For every n it holds that $\ell(n) > n$.*
2. (Pseudorandomness:) *For all probabilistic polynomial-time distinguishers D, there exists a negligible function* negl *such that:*

$$\left|\Pr[D(r)=1] - \Pr[D(G(s))=1]\right| \leq \mathsf{negl}(n),$$

where r is chosen uniformly at random from $\{0,1\}^{\ell(n)}$, the seed *s is chosen uniformly at random from $\{0,1\}^n$, and the probabilities are taken over the random coins used by D and the choice of r and s.*

The function $\ell(\cdot)$ is called the expansion factor *of G.*

Discussion



- With *exponential* samples it is easy to distinguish pseudorandomness from randomness!

DESIGNING a SECURE ENCRYPTION SCHEME USING PRGs
(just "pseurorandomize the one-time pad!)

- Gen: on input 1^n, choose $k \leftarrow \{0,1\}^n$ uniformly at random and output it as the key.
- Enc: on input a key $k \in \{0,1\}^n$ and a message $m \in \{0,1\}^{\ell(n)}$, output the ciphertext
$$c := G(k) \oplus m.$$
- Dec: on input a key $k \in \{0,1\}^n$ and a ciphertext $c \in \{0,1\}^{\ell(n)}$, output the plaintext message
$$m := G(k) \oplus c.$$

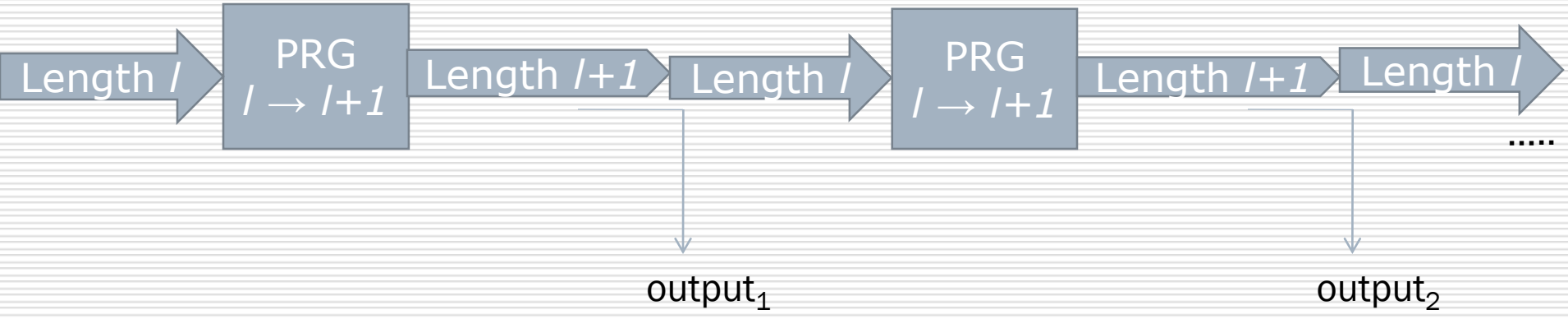
Designing PRGs from Computational Hardness

- Single bit expansion PRG to arbitrary expansion PRG
- From One-way functions to single-bit expansion PRG
- Candidate PRG from Discrete Logarithm

Expanding the Expansion in PRG

THEOREM 6.8 *Assume that there exists a pseudorandom generator with expansion factor $\ell(n) = n + 1$. Then for any polynomial $p(\cdot)$, there exists a pseudorandom generator with expansion factor $\ell(n) = p(n)$.*

THEOREM 6.8 *Assume that there exists a pseudorandom generator with expansion factor $\ell(n) = n + 1$. Then for any polynomial $p(\cdot)$, there exists a pseudorandom generator with expansion factor $\ell(n) = p(n)$.*



1. Take the last bit from *l* + *1* length string for output
2. Apply *l′* times to get output of string *l′*

Designing single-bit expansion PRGs from Computational Hardness

- One-way Functions
  - Easy to compute, Hard to Invert
  - Textbook definition:

DEFINITION 6.1 *A function* $f : \{0,1\}^* \to \{0,1\}^*$ *is* one-way *if the following two conditions hold:*

1. (Easy to compute:) *There exists a polynomial-time algorithm* M_f *computing* f*; that is,* $M_f(x) = f(x)$ *for all* x.
2. (Hard to invert:) *For every probabilistic polynomial-time algorithm* $\mathcal{A}$*, there exists a negligible function* negl *such that*

$$\Pr[\mathsf{Invert}_{\mathcal{A},f}(n) = 1] \leq \mathsf{negl}(n).$$

A Candidate One-way Function

- Discrete Logarithm Problem (in Z_p^*)

$$f_{p,g}(x) = g^x \bmod p$$

HARDCORE PREDICATES

- Hardest bit of information about x to obtain from f(x)
- Textbook definition:

DEFINITION 6.5 *A function* hc : $\{0,1\}^* \to \{0,1\}$ *is a* hard-core predicate of a function f *if* **(1)** hc *can be computed in polynomial time, and* **(2)** *for every probabilistic polynomial-time algorithm* $\mathcal{A}$ *there exists a negligible function* negl *such that*

$$\Pr_{x \leftarrow \{0,1\}^n}[\mathcal{A}(f(x)) = \mathsf{hc}(x)] \leq \frac{1}{2} + \mathsf{negl}(n),$$

where the probability is taken over the uniform choice of x *in* $\{0,1\}^n$ *and the random coin tosses of* $\mathcal{A}$.

MSB(x) is a Hardcore predicate of Discrete Logarithm Problem

From One-way Functions to PRGs

THEOREM 6.7 *Let f be a one-way permutation and let* hc *be a hard-core predicate of f. Then, $G(s) \stackrel{\text{def}}{=} (f(s), \mathsf{hc}(s))$ constitutes a pseudorandom generator with expansion factor $\ell(n) = n + 1$.*

In case of discrete logarithms:

$$G(s) = (g^s \bmod p, \mathrm{msb}(s))$$

TASK: Implement your own provably secure pseudorandom generator assuming DLP is a one-way function.

Discrete Logarithm based PRG

THANK YOU

Any Questions?

CPA-SECURITY

And

PSEUDORANDOM FUNCTIONS

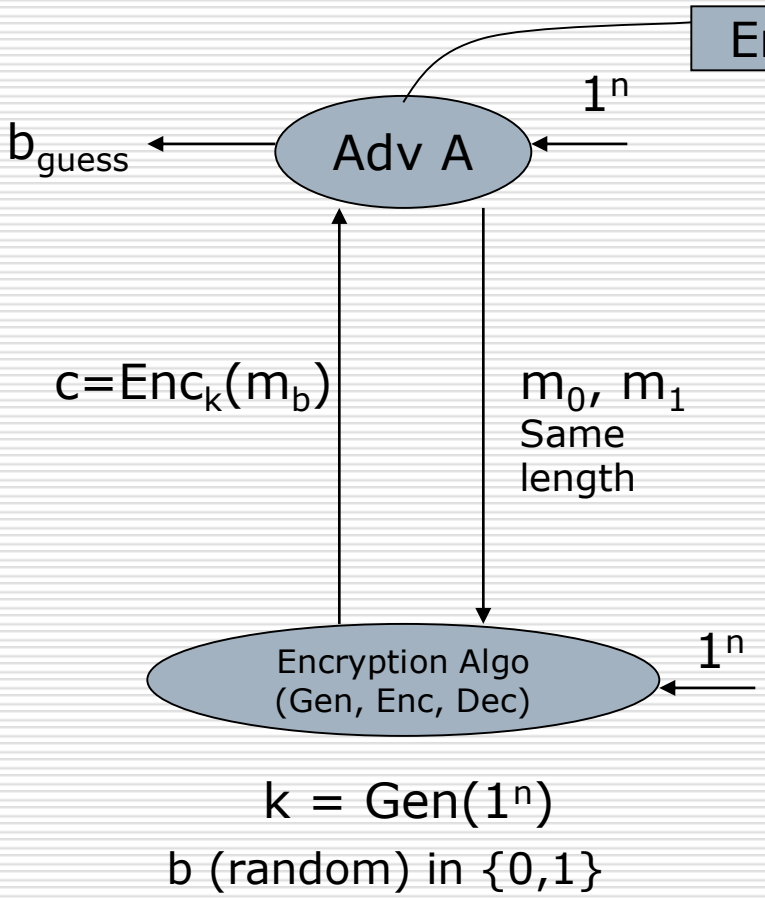
RECALL: IT'S STILL ONE-TIME USAGE!

- Gen: on input 1^n, choose $k \leftarrow \{0,1\}^n$ uniformly at random and output it as the key.
- Enc: on input a key $k \in \{0,1\}^n$ and a message $m \in \{0,1\}^{\ell(n)}$, output the ciphertext
$$c := G(k) \oplus m.$$
- Dec: on input a key $k \in \{0,1\}^n$ and a ciphertext $c \in \{0,1\}^{\ell(n)}$, output the plaintext message
$$m := G(k) \oplus c.$$

CHOSEN PLAINTEXT ATTACK (CPA)

Adversary can obtain the ciphertexts of any message of his/her choice!

Defining Computational Security Against CHOSEN PLAINTEXT ATTACK



Cryptosystem is CPA-secure if for all PPT adversaries A:

$P[b_{guess} = b] <= ½ + negl(n)$

THEOREM

- **No deterministic Encryption algortihm can be CPA-secure!**

- Why?

PROBABILISTIC ENCRYPTION

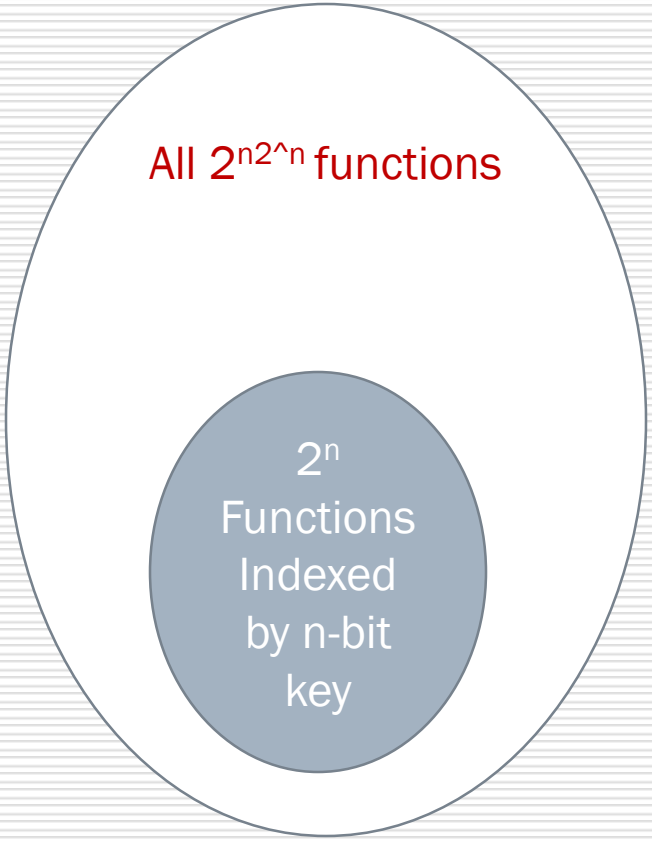
Pseudorandom Functions F_k

Basic idea: $c = (r, F_k(r) + m)$

Pseudorandom Functions (PRF)

- Functions that are easy to compute
- Computationally indistinguishable from a random function, say from domain $\{0,1\}^n$ to co-domain $\{0,1\}^n$
- Recall that there are $2^{n2\hat{}n}$ possible functions

PRFs are even more "pseudo" than PRGs



With *exponential* queries it is easy to distinguish pseudorandom functions from truly random ones!

PRF Definition

DEFINITION 3.23 *Let* $F : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ *be an efficient, length-preserving, keyed function. We say that* F *is a* pseudorandom function *if for all probabilistic polynomial-time distinguishers* D*, there exists a negligible function* negl *such that:*

$$\left| \Pr[D^{F_k(\cdot)}(1^n) = 1] - \Pr[D^{f(\cdot)}(1^n) = 1] \right| \leq \mathrm{negl}(n),$$

where $k \leftarrow \{0,1\}^n$ *is chosen uniformly at random and* f *is chosen uniformly at random from the set of functions mapping* n*-bit strings to* n*-bit strings.*

CPA-SECURE ENCRYPTION

CONSTRUCTION 3.24

Let F be a pseudorandom function. Define a private-key encryption scheme for messages of length n as follows:

- Gen: on input 1^n, choose $k \leftarrow \{0,1\}^n$ uniformly at random and output it as the key.
- Enc: on input a key $k \in \{0,1\}^n$ and a message $m \in \{0,1\}^n$, choose $r \leftarrow \{0,1\}^n$ uniformly at random and output the ciphertext

$$c := \langle r,\ F_k(r) \oplus m \rangle.$$

- Dec: on input a key $k \in \{0,1\}^n$ and a ciphertext $c = \langle r, s \rangle$, output the plaintext message

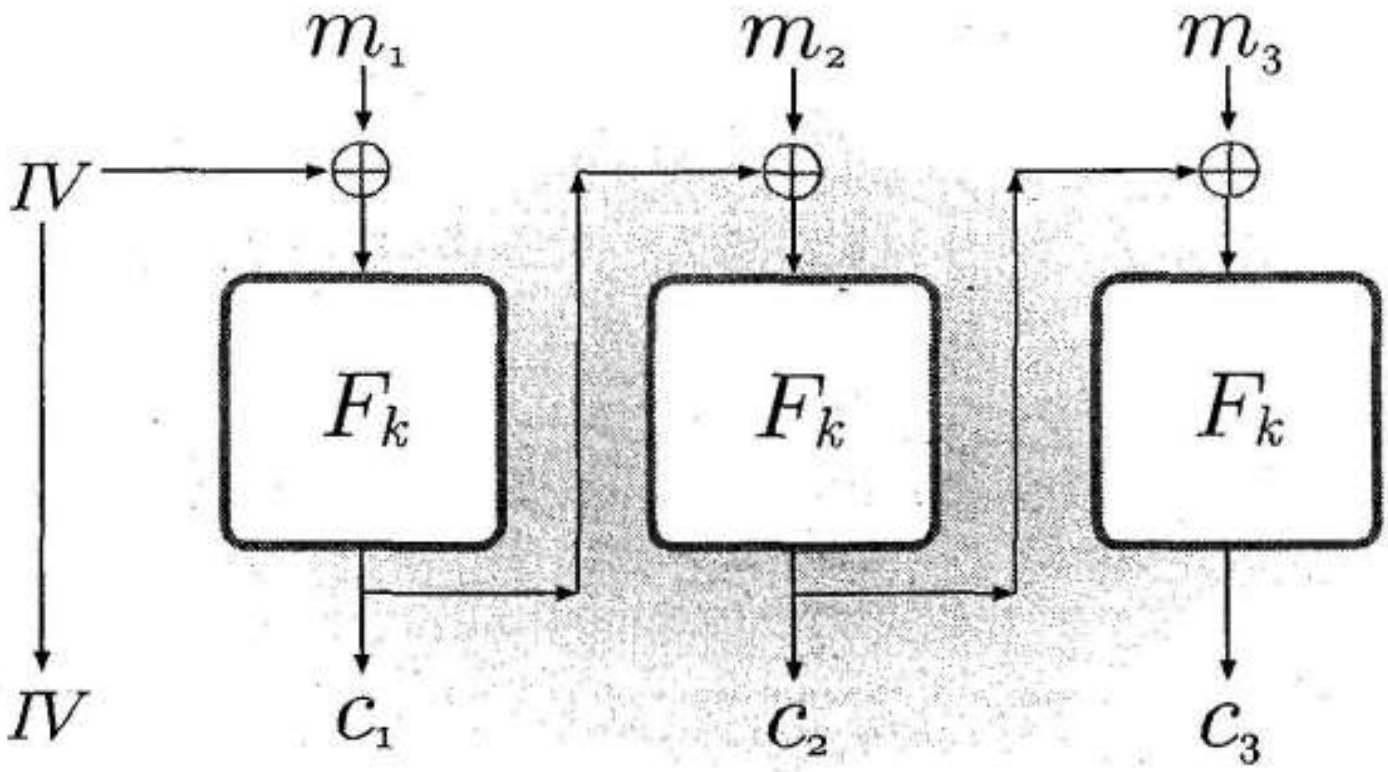
$$m := F_k(r) \oplus s.$$

A CPA-secure encryption scheme from any pseudorandom function.

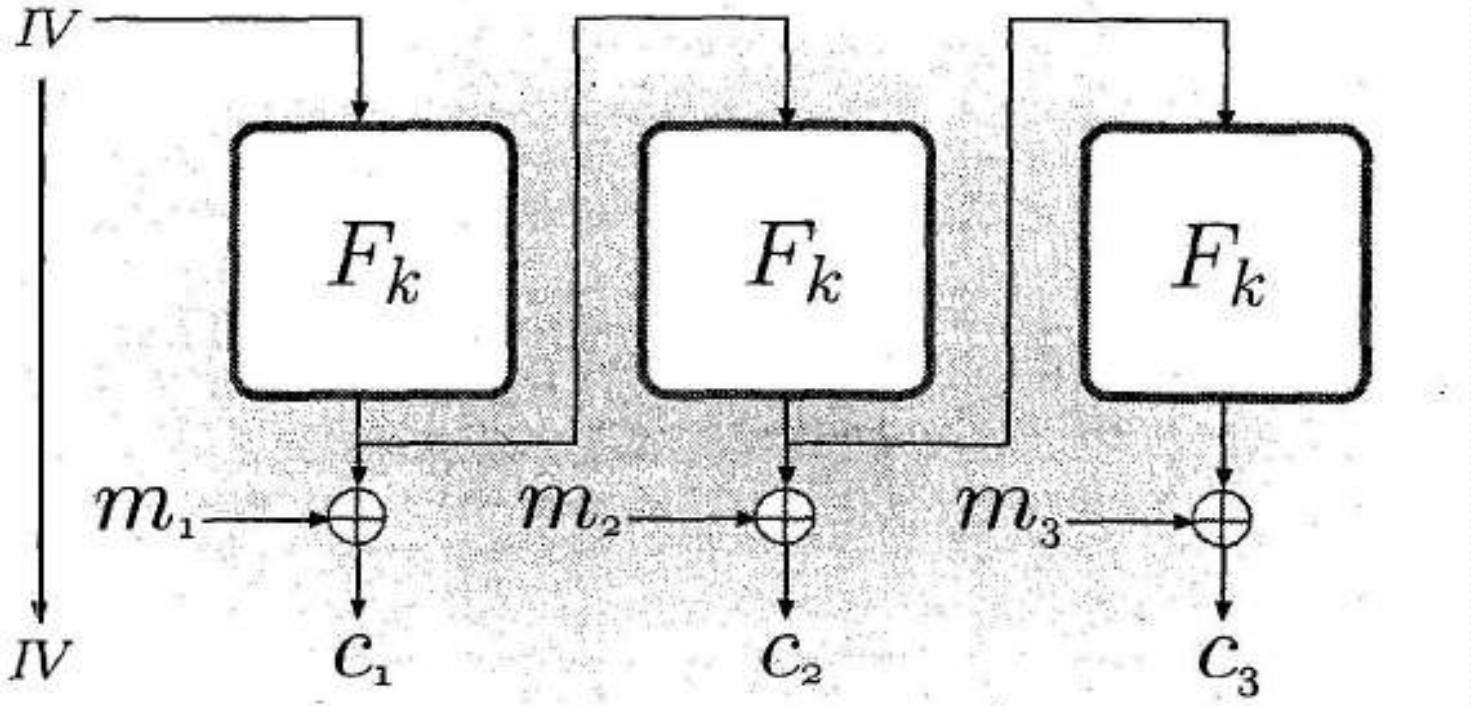
MODES OF OPERATION

Using Blockciphers/PRFs

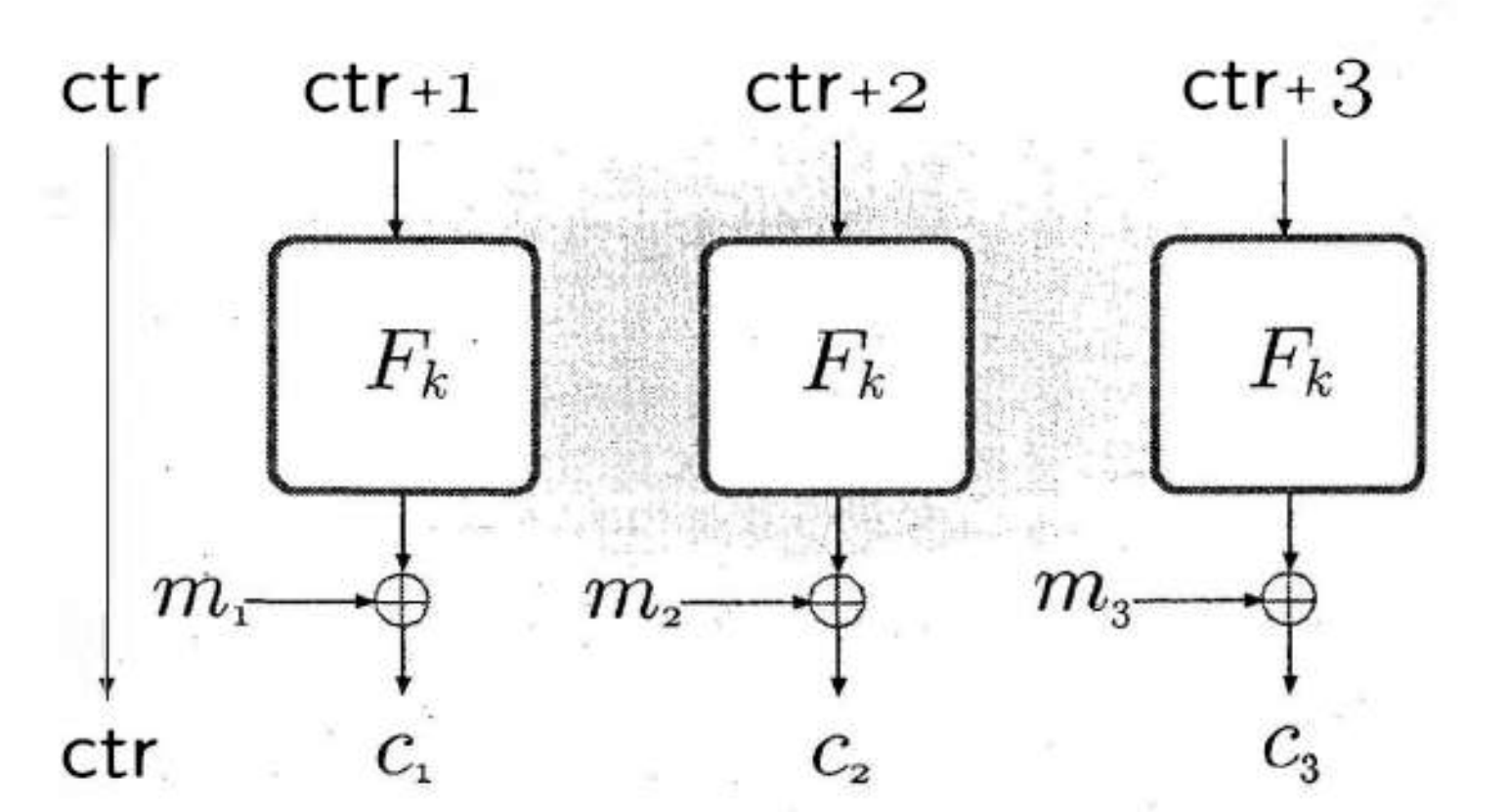
CIPHER BLOCK CHAINING (CBC)



OUTPUT FEEDBACK MODE (OFB)



RANDOMIZED COUNTER MODE



PRF From PRG

CONSTRUCTION 6.24

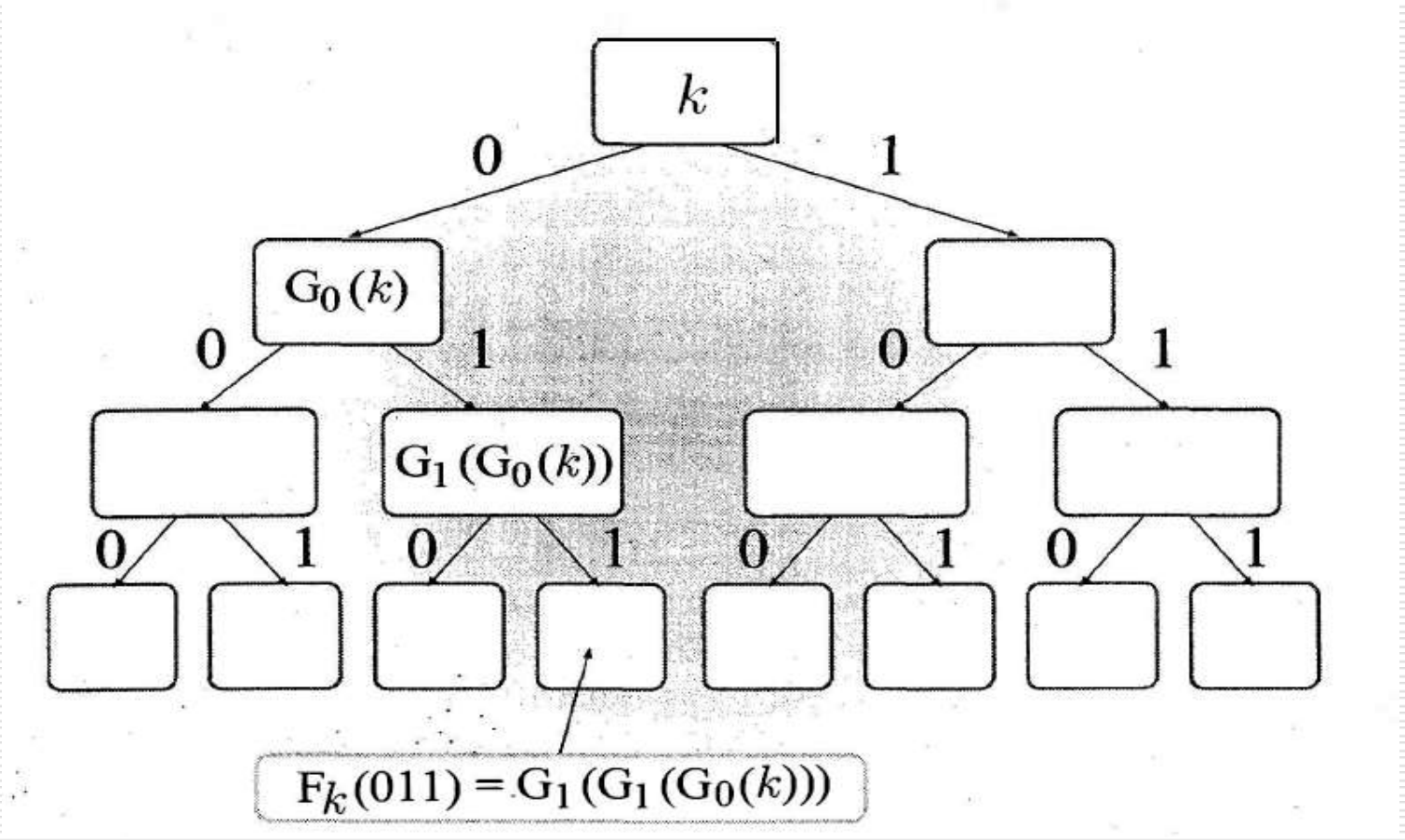
Let G be a pseudorandom generator with expansion factor $\ell(n) = 2n$. Denote by $G_0(k)$ the first half of G's output, and by $G_1(k)$ the second half of G's output. For every $k \in \{0,1\}^n$, define the function $F_k : \{0,1\}^n \to \{0,1\}^n$ as:

$$F_k(x_1 x_2 \cdots x_n) = G_{x_n}\left(\cdots\left(G_{x_2}\left(G_{x_1}(k)\right)\right)\cdots\right).$$

A pseudorandom function from a pseudorandom generator.

THEOREM 6.25 *If G is a pseudorandom generator with expansion factor $\ell(n) = 2n$, then Construction 6.24 is a pseudorandom function.*

PRF from PRG (Contd.)

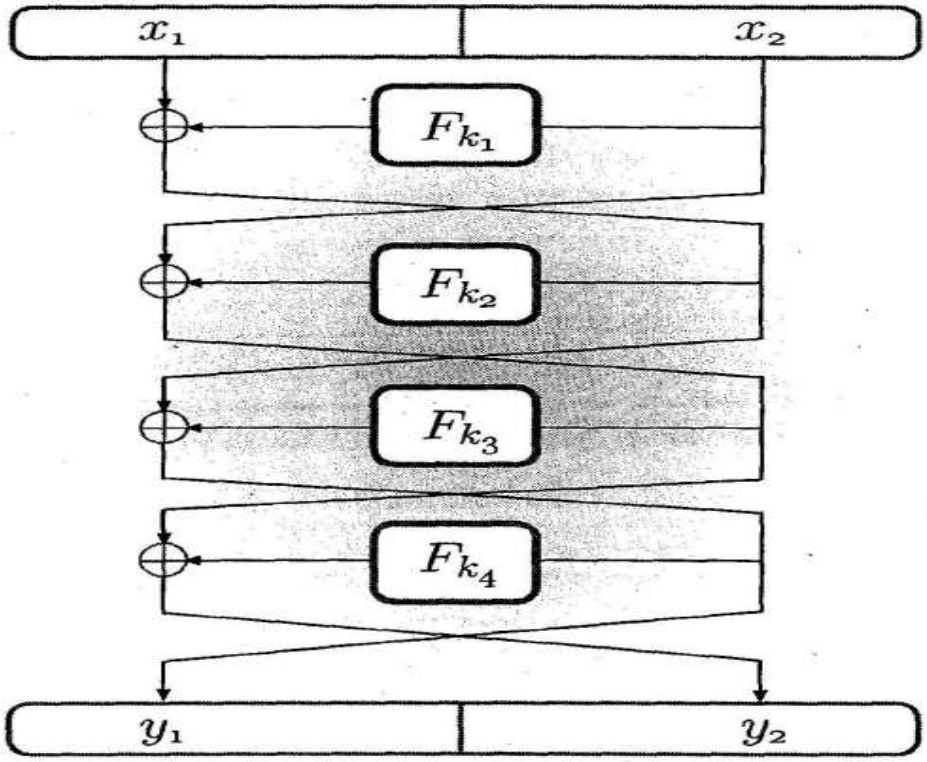


PRFs to Invertible PRFs

- Feistel Networks

$$L_i := R_{i-1} \quad \text{and} \quad R_i := L_{i-1} \oplus f_i(R_{i-1})$$

Multiple Round Feistel Structure



SUMMARY

- CPA attacks is easy in practice today
- CPA-security requires probabilistic encryption
- Pseudorandom Functions (PRF) in the right mode of operation
- Designing PRFs from PRGs

TASK (till date)

- Assuming Discrete Logarithm Problem to be a one-way permutation:
  - Build a provable secure PRG
  - Build a provably secure PRF
  - Use the PRF in some secure mode of operation to obtain a CPA-secure encryption scheme

THANK YOU

Any Questions?

CCA-SECURITY

And

Message Authentication Codes (MAC)

REVIEW

- CPA attacks is easy in practice today
- CPA-security requires probabilistic encryption
- Pseudorandom Functions (PRF) in the right mode of operation
- Designing PRFs from PRGs

PRF From PRG

CONSTRUCTION 6.24

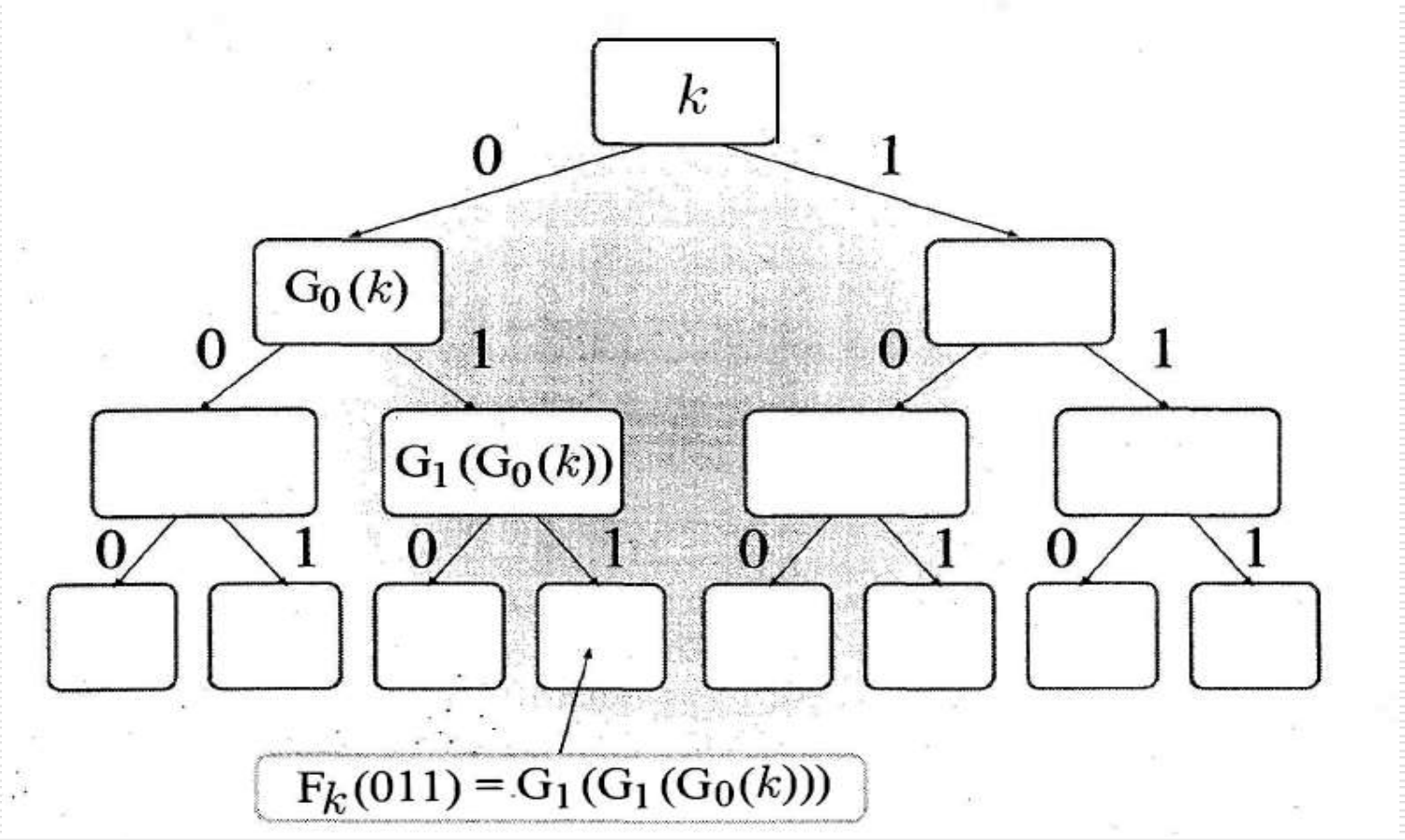
Let G be a pseudorandom generator with expansion factor $\ell(n) = 2n$. Denote by $G_0(k)$ the first half of G's output, and by $G_1(k)$ the second half of G's output. For every $k \in \{0,1\}^n$, define the function $F_k : \{0,1\}^n \to \{0,1\}^n$ as:

$$F_k(x_1 x_2 \cdots x_n) = G_{x_n}\left(\cdots\left(G_{x_2}\left(G_{x_1}(k)\right)\right)\cdots\right).$$

A pseudorandom function from a pseudorandom generator.

THEOREM 6.25 *If G is a pseudorandom generator with expansion factor $\ell(n) = 2n$, then Construction 6.24 is a pseudorandom function.*

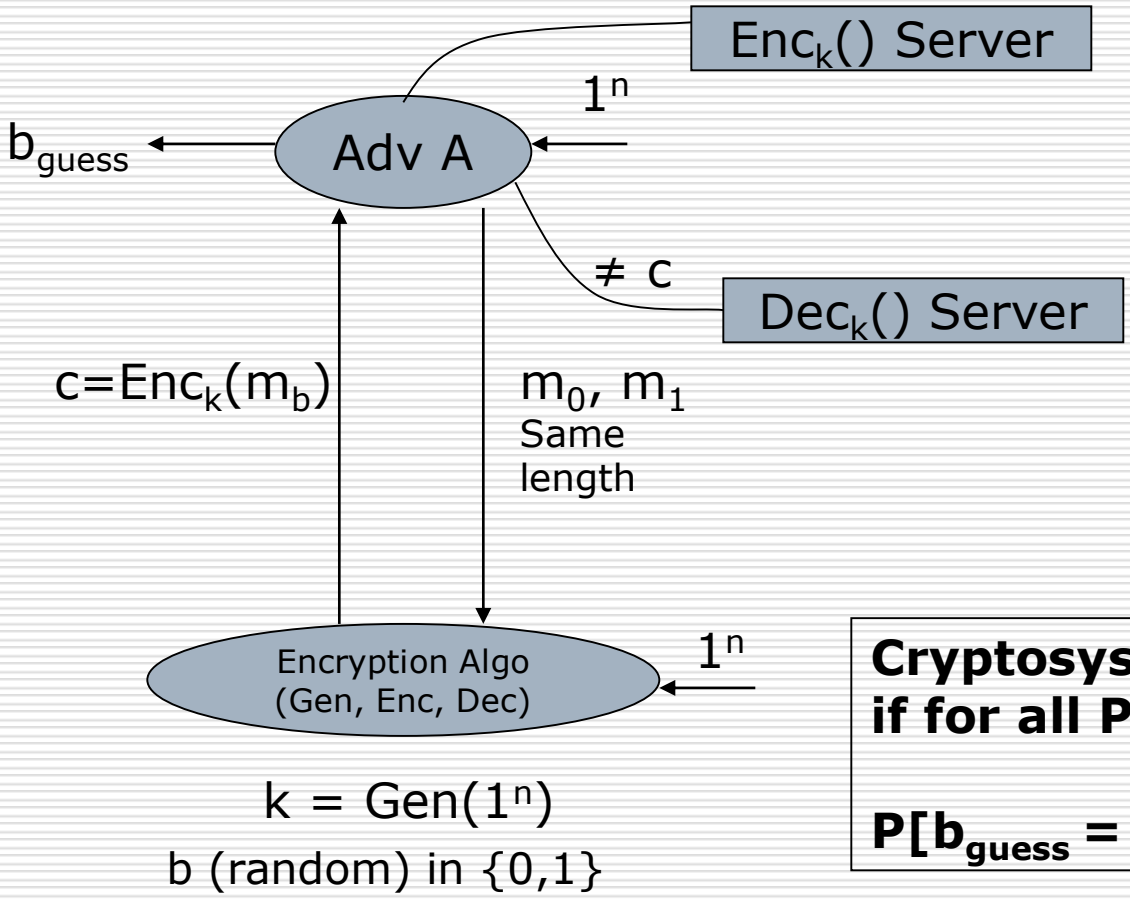
PRF from PRG (Contd.)



CPA-Security Doesn’t Suffice!

CCA-Security

Defining Computational Security Against CHOSEN CIPHERTEXT ATTACK (CCA)



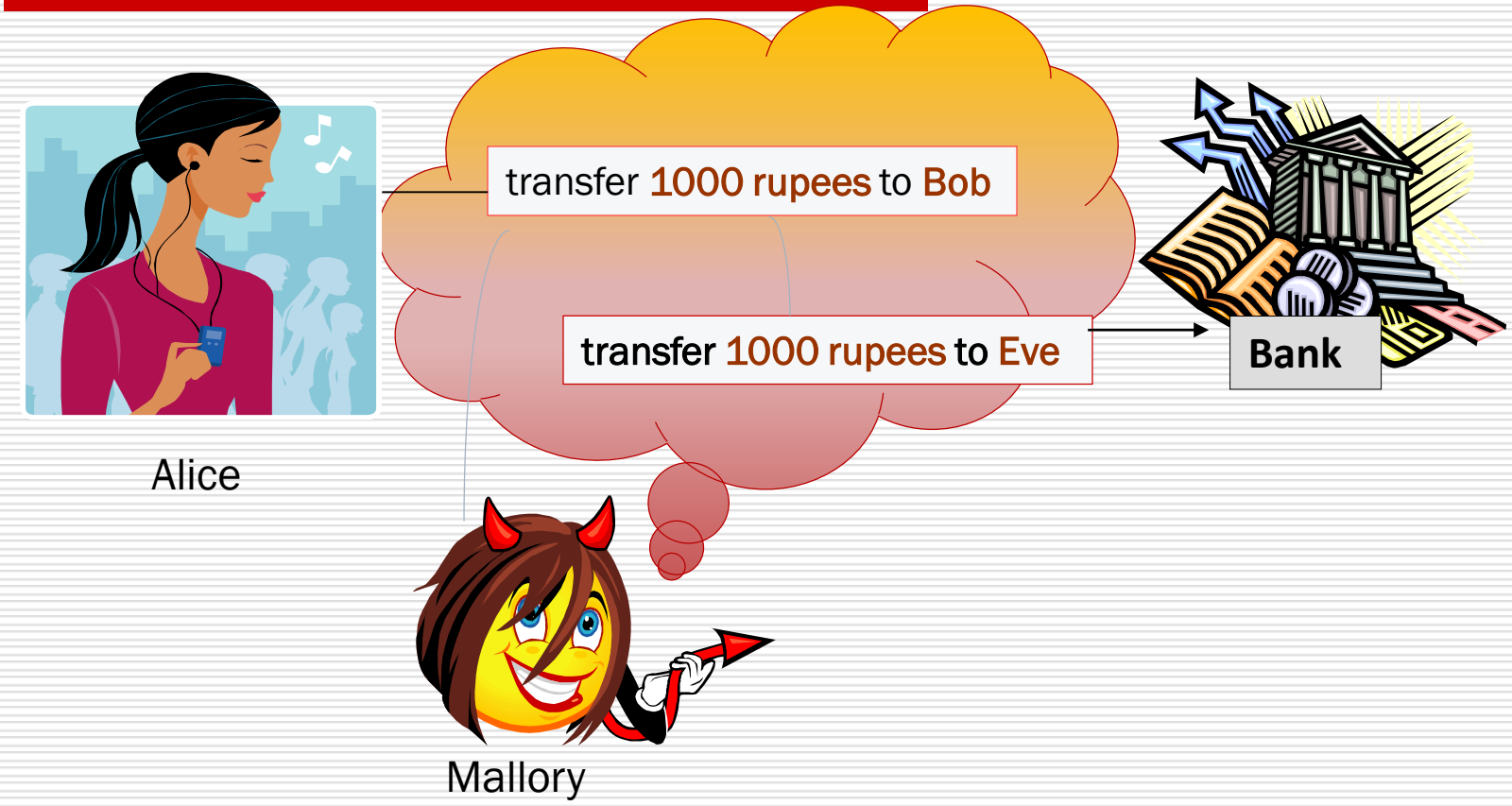
Cryptosystem is CCA-secure if for all PPT adversaries A:

$P[b_{guess} = b] <= ½ + negl(n)$

Message Authentication Codes

Need, Construction and Attacks

Problem of Data Integrity

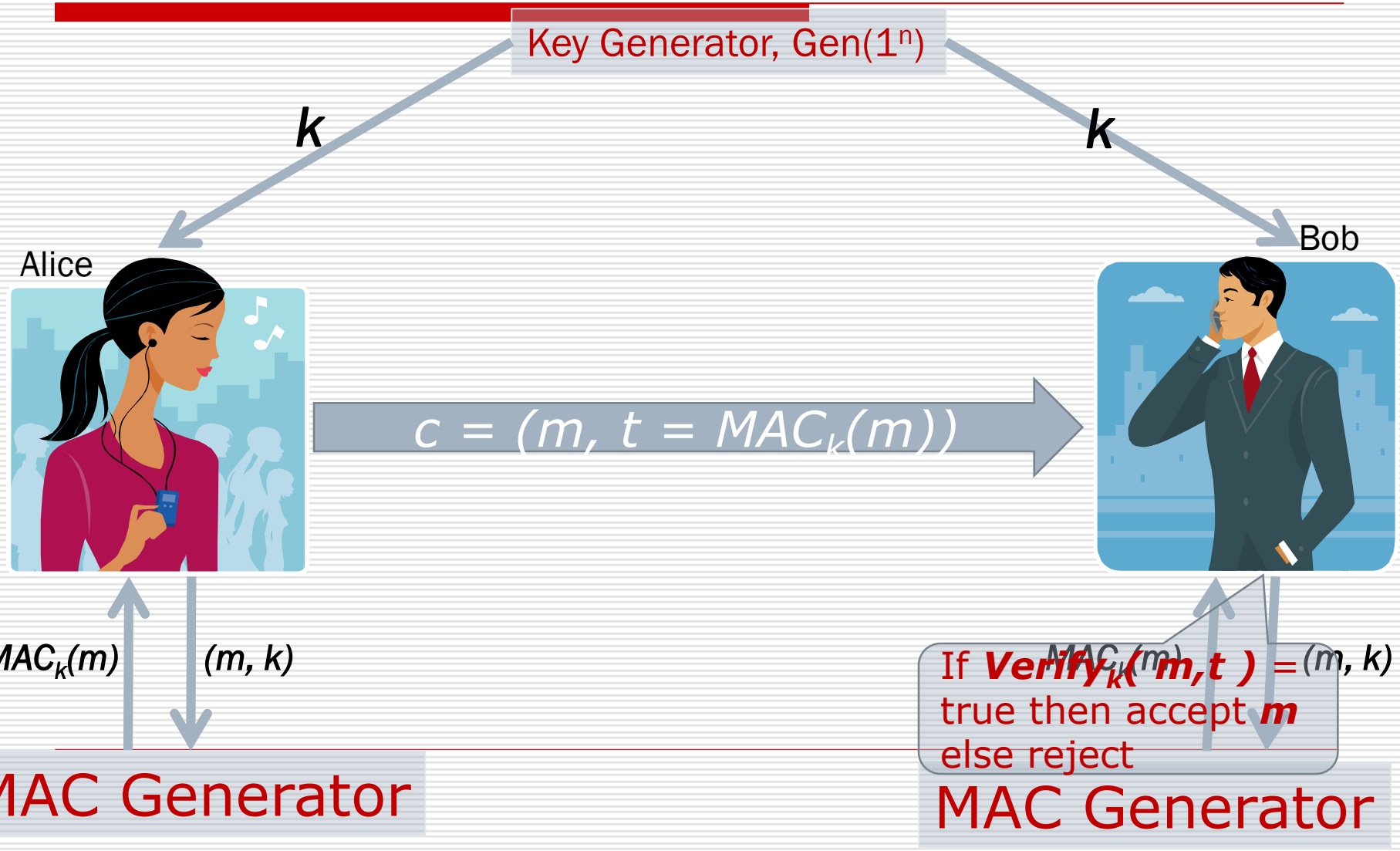


How to prevent such a modification of data?

Does Encryption Guarantee Integrity?

NO!

Data Authentication using a MAC

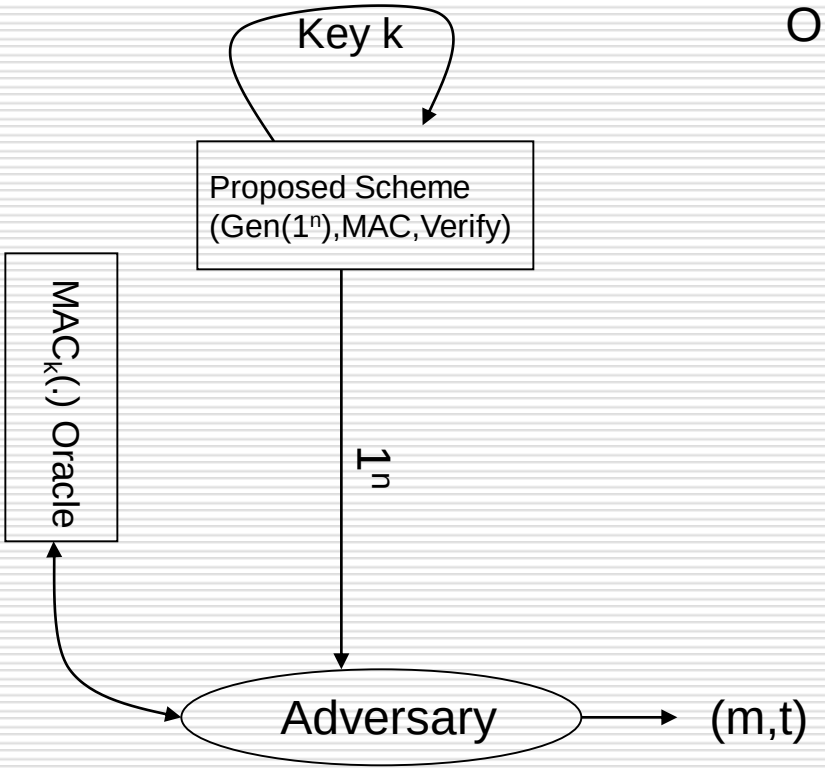


Components of the Authentication Protocol

- A Key Generation Algorithm that returns a secret key ***k***
- A MAC generating algorithm that returns a tag for a given message ***m***. Tag $\boldsymbol{t = MAC_k(m)}$
- A Verification algorithm that returns a bit $\boldsymbol{b = Verify_k(m_1, t_1)}$, given a message $\boldsymbol{m_1}$ and a tag $\boldsymbol{t_1}$
- If the message is not modified then with high probability, the value of ***b*** is true otherwise false

Security of MAC

Mac-Game(n)



Let Q be the set of all queries from Adv to oracle

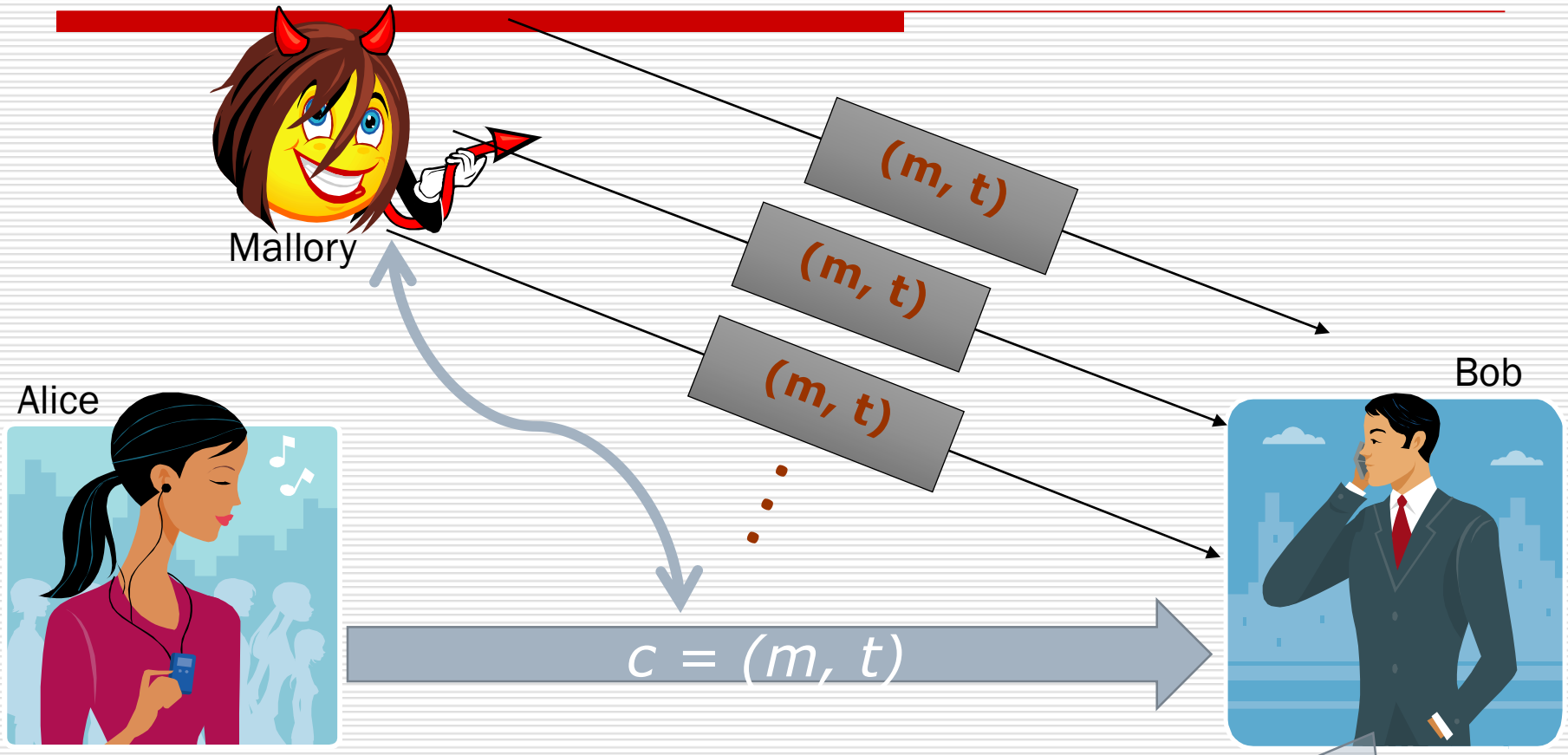
Output of the Game is 1 if and only if:

$Verify_k(m,t) = 1$ and m is not in Q

A message authentication code (Gen,MAC,Verify) is secure if for all probabilistic polynomial-time adversaries ***A***:

$Pr[\text{Mac-Game}(n) = 1] \leq negl(n)$

Replay Attack



Verify has no "*memory*", cannot detect that ***(m, t)*** is not fresh!

How to solve this problem?

Sequence numbers/timestamps can be used

Construction of MAC using a PRF

Gen (1^n) chooses k to be a random n-bit string

$MAC_k(m) = F_k(m) = t$ ***(the tag)***

Verify$_k$ ***(m, t) = Accept***, if and only if $t = F_k(m)$

Theorem: If F is a pseudorandom function, the above scheme is a secure *fixed length* **MAC**

Variable Length MACs (Trial 1)

- Partition the message ***m*** to ***n*** sized blocks $m_1 m_2 \ldots m_q$
- Calculate $MAC_k(m) = MAC_k(m_1 \oplus m_2 \ \ldots \oplus m_q)$
- Is this method Secure?

- NO! We are authenticating the ***xor*** of the message blocks but not the message itself. So we can always choose a message whose ***xor*** value is the same as some other message

Variable Length MACs (Trial 2)

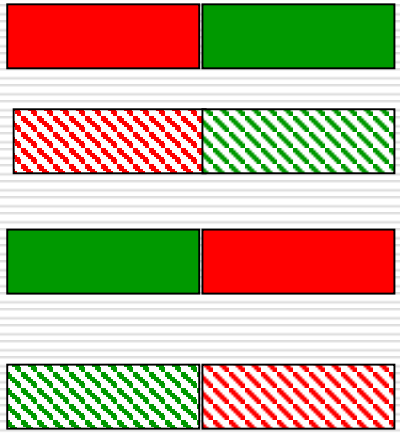
- Concatenate the TAG values of all blocks calculated separately
- But the adversary can rearrange the message blocks and respective tags generating the new message and tag

m:

$t = MAC_k(m)$:

m' = permutation(m):

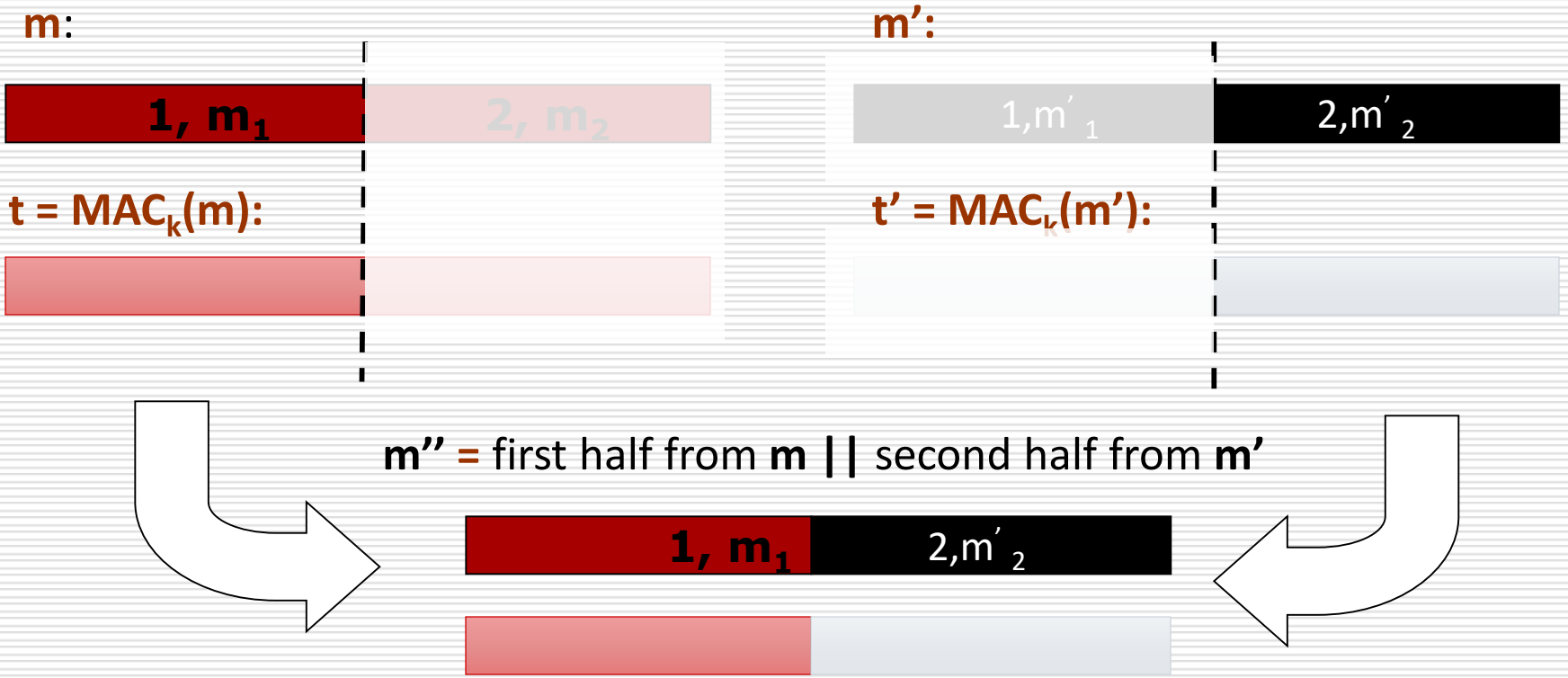
t' = permutation(t):



- Not Secure!

Variable Length MACs (Trial 3)

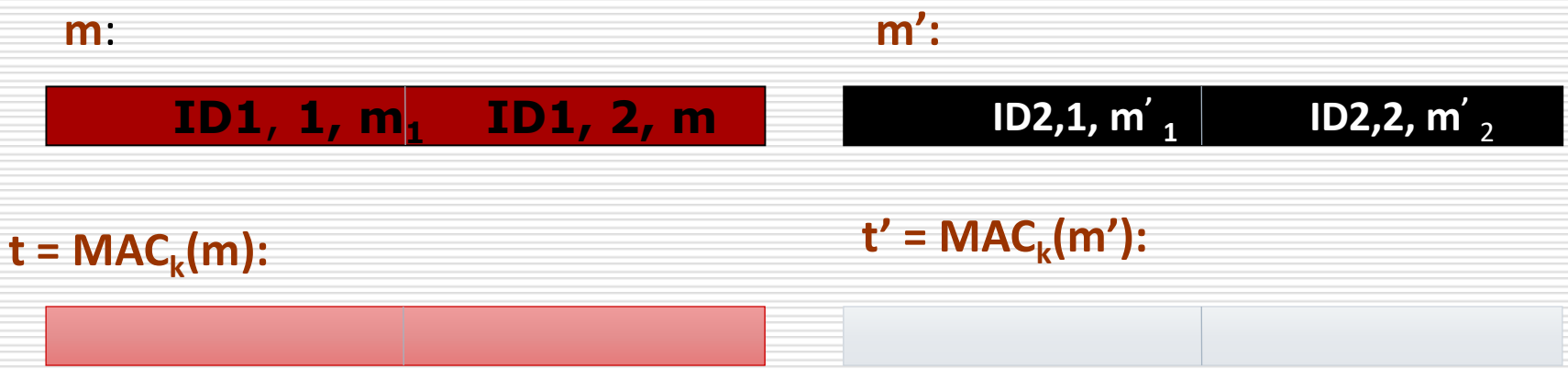
- To prevent the reordering in previous trial, we use sequence numbers. But consider the problem below:



- Then **t''** is a valid tag on **m''**. Not Secure!

Variable Length MACs (Trial 4)

- To prevent the above attack we may use random/unique message ids. So, we may send it as:



- The adversary cannot re-arrange the blocks.

FINAL PROTOCOL (textbook)

CONSTRUCTION 4.5

Let $\Pi' = (\mathsf{Gen}', \mathsf{Mac}', \mathsf{Vrfy}')$ be a fixed-length MAC for messages of length n. Define a MAC as follows:

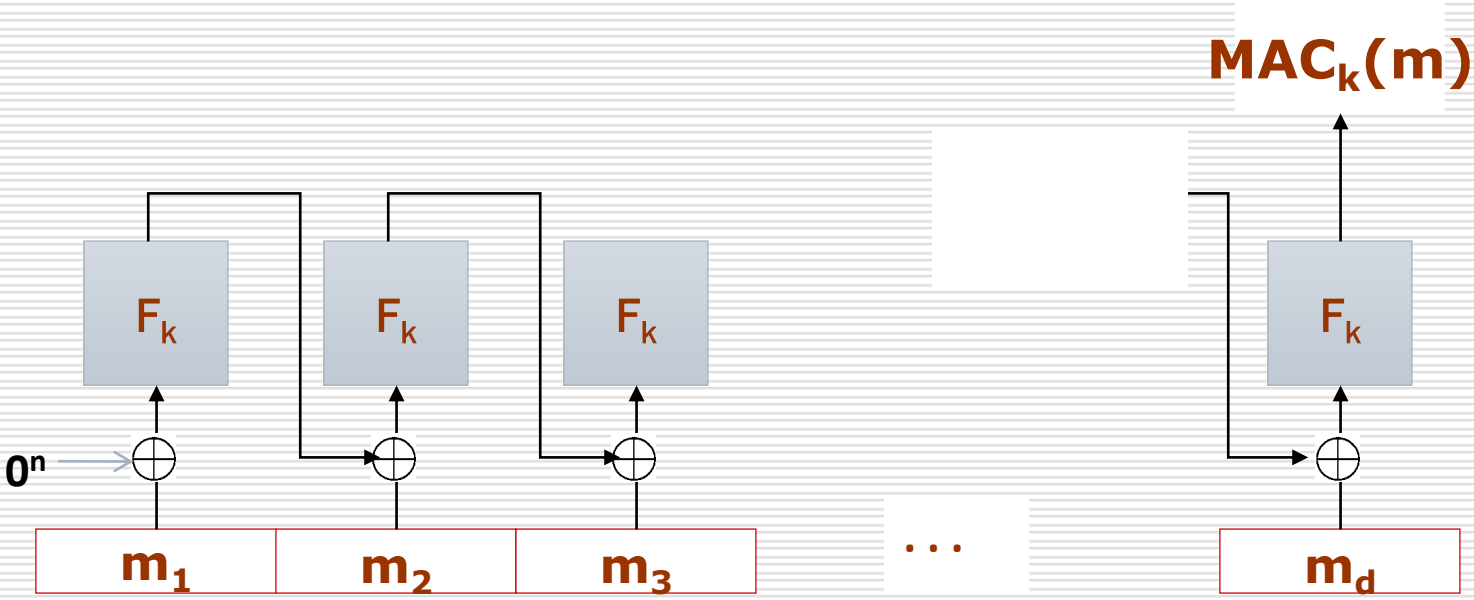
- Gen: this is identical to Gen'.
- Mac: on input a key $k \in \{0,1\}^n$ and a message $m \in \{0,1\}^*$ of length $\ell < 2^{\frac{n}{4}}$, parse m into d blocks $m_1, \ldots, m_d$, each of length $n/4$. (The final block is padded with 0s if necessary.) Next, choose a random identifier $r \leftarrow \{0,1\}^{n/4}$.

  For $i = 1, \ldots, d$, compute $t_i \leftarrow \mathsf{Mac}'_k(r \| \ell \| i \| m_i)$, where i and ℓ are uniquely encoded as strings of length $n/4$.[4] Finally, output the tag $t := \langle r, t_1, \ldots, t_d \rangle$.
- Vrfy: on input a key $k \in \{0,1\}^n$, a message $m \in \{0,1\}^*$ of length $\ell < 2^{\frac{n}{4}}$, and a tag $t = \langle r, t_1, \ldots, t_{d'} \rangle$, parse m into d blocks $m_1, \ldots, m_d$, each of length $n/4$. (The final block is padded with 0s if necessary.) Output 1 if and only if $d' = d$ and $\mathsf{Vrfy}'_k(r \| \ell \| i \| m_i, t_i) = 1$ for $1 \leq i \leq d$.

A variable-length MAC from any fixed-length MAC.

Cipher Block Chaining MAC (CBC-MAC)

CBC-MAC Construction



But again, CBC-MAC is secure for fixed length messages but not for variable length messages! Why?

Problem with Variable Length CBC-MAC

Mallory chooses these two messages that Alice has sent

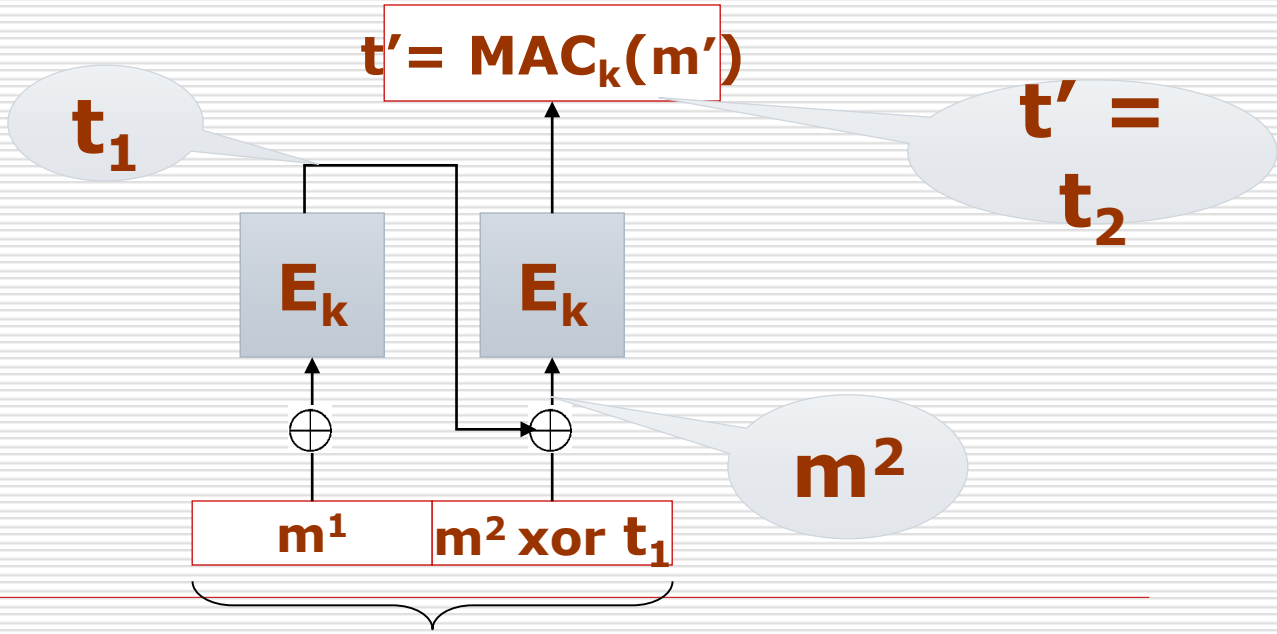
m^1 $t_1 = MAC_k(m^1)$

m^2 $t_2 = MAC_k(m^2)$

Problem with Variable Length CBC-MAC

Mallory has two message pairs as shown above. She now can construct a new message shown below

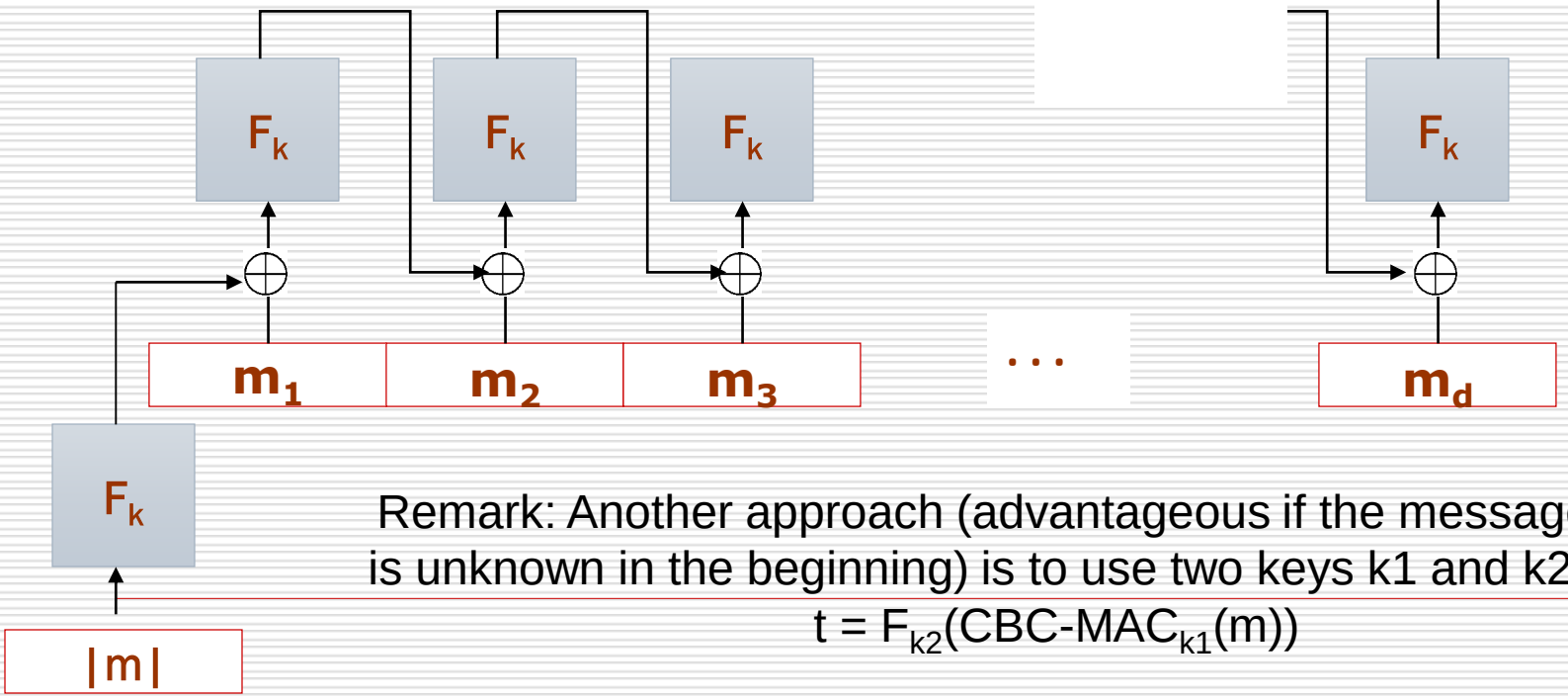
Mallory can now send this new valid pair ***(m′, t′)*** to Bob



CBC-MAC Construction

A secure CBC-MAC for variable length messages

Prepend length of the message |m| (encoded as an n-bit string) to m and then compute the tag (appending the length to the end is not secure!)



Remark: Another approach (advantageous if the message length is unknown in the beginning) is to use two keys k1 and k2 and set

$t = F_{k2}(\text{CBC-MAC}_{k1}(m))$

CCA-SECURITY

From CPA-security
and
Secure MAC

ENCRYPT-THEN-AUTHENTICATE

$$c = (r, F_{k1}(r)+m), MAC_{k2}(r, Fk1(r)+m)$$

CCA-Secure Encryption (textbook)

CONSTRUCTION 4.19

Let $\Pi_E = (\mathsf{Gen}_E, \mathsf{Enc}, \mathsf{Dec})$ be a private-key encryption scheme and let $\Pi_M = (\mathsf{Gen}_M, \mathsf{Mac}, \mathsf{Vrfy})$ be a message authentication code. Define an encryption scheme $(\mathsf{Gen}', \mathsf{Enc}', \mathsf{Dec}')$ as follows:

- Gen': on input 1^n, run $\mathsf{Gen}_E(1^n)$ and $\mathsf{Gen}_M(1^n)$ to obtain keys k_1, k_2, respectively.
- Enc': on input a key (k_1, k_2) and a plaintext message m, compute $c \leftarrow \mathsf{Enc}_{k_1}(m)$ and $t \leftarrow \mathsf{Mac}_{k_2}(c)$ and output the ciphertext $\langle c, t \rangle$
- Dec': on input a key (k_1, k_2) and a ciphertext $\langle c, t \rangle$, first check whether $\mathsf{Vrfy}_{k_2}(c, t) \stackrel{?}{=} 1$. If yes, then output $\mathsf{Dec}_{k_1}(c)$; if no, then output $\perp$.

A CCA-secure private-key encryption scheme.

TASKS (till date)

Assuming Discrete Logarithm Problem to be a one-way

- **Build a provable secure PRG**
- **Build a provably secure PRF**
- **Use the PRF in some secure mode of operation to obtain a CPA-secure encryption scheme**
- **Use the PRF to build a secure MAC**
- **Use CPA-security and secure MAC to design a state-of-the-art provably CCA-secure encrytpion scheme**

THANK YOU

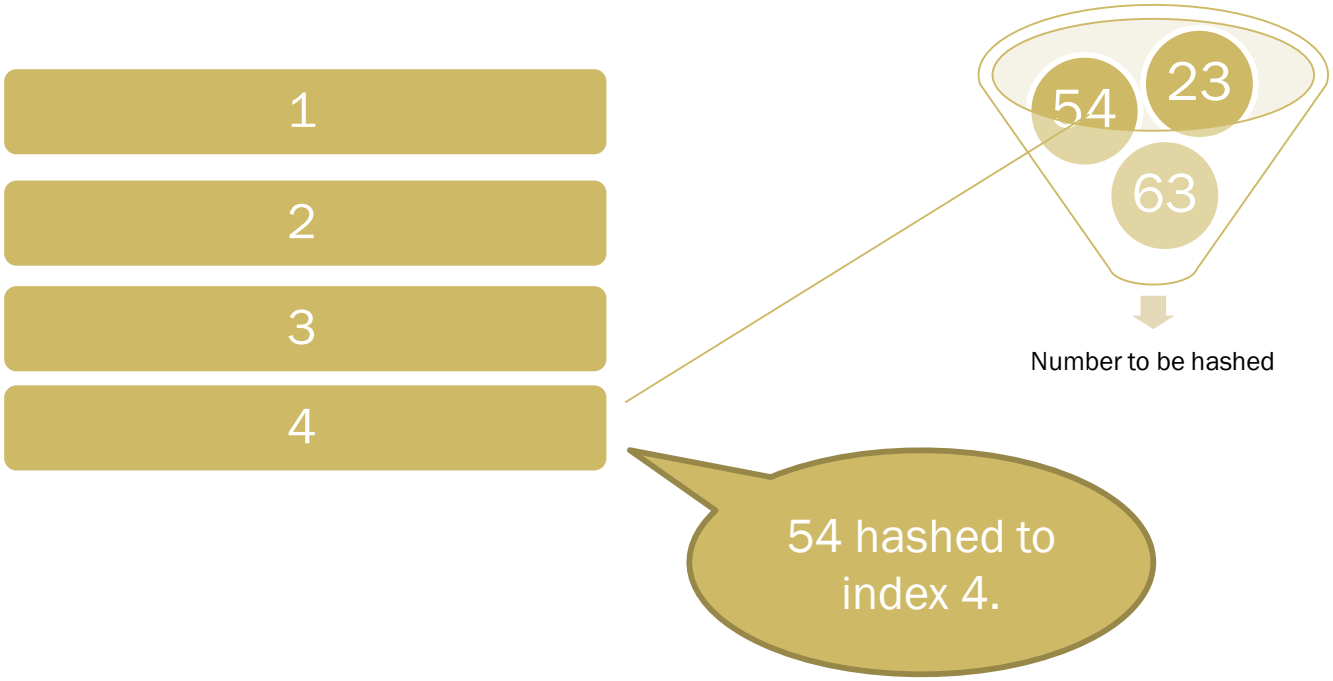
Any Questions?

Collision Resistant Hash Functions

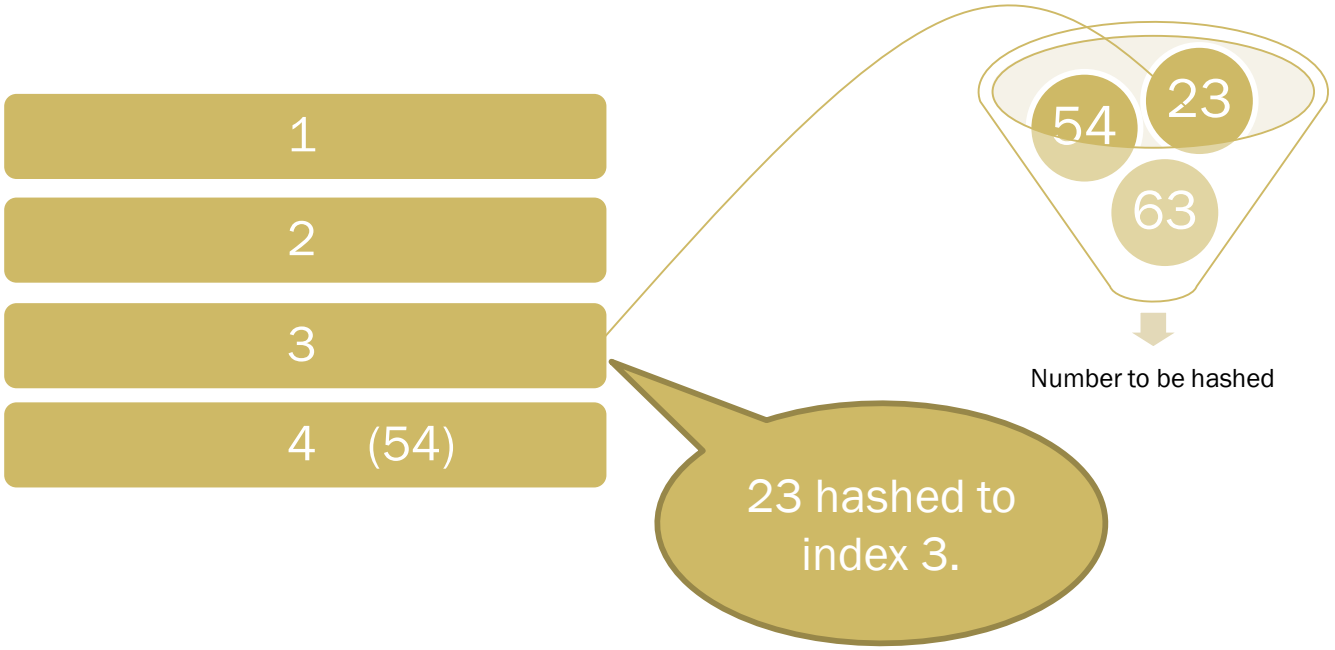
Hash Functions

- Traditionally, hash functions take arbitrary length strings and compress them into shorter strings
- Classically used in data structures for improved look-up times in storage/retrieval
- Collisions for the hash function H are distinct inputs x and y such that H(x) = H(y)

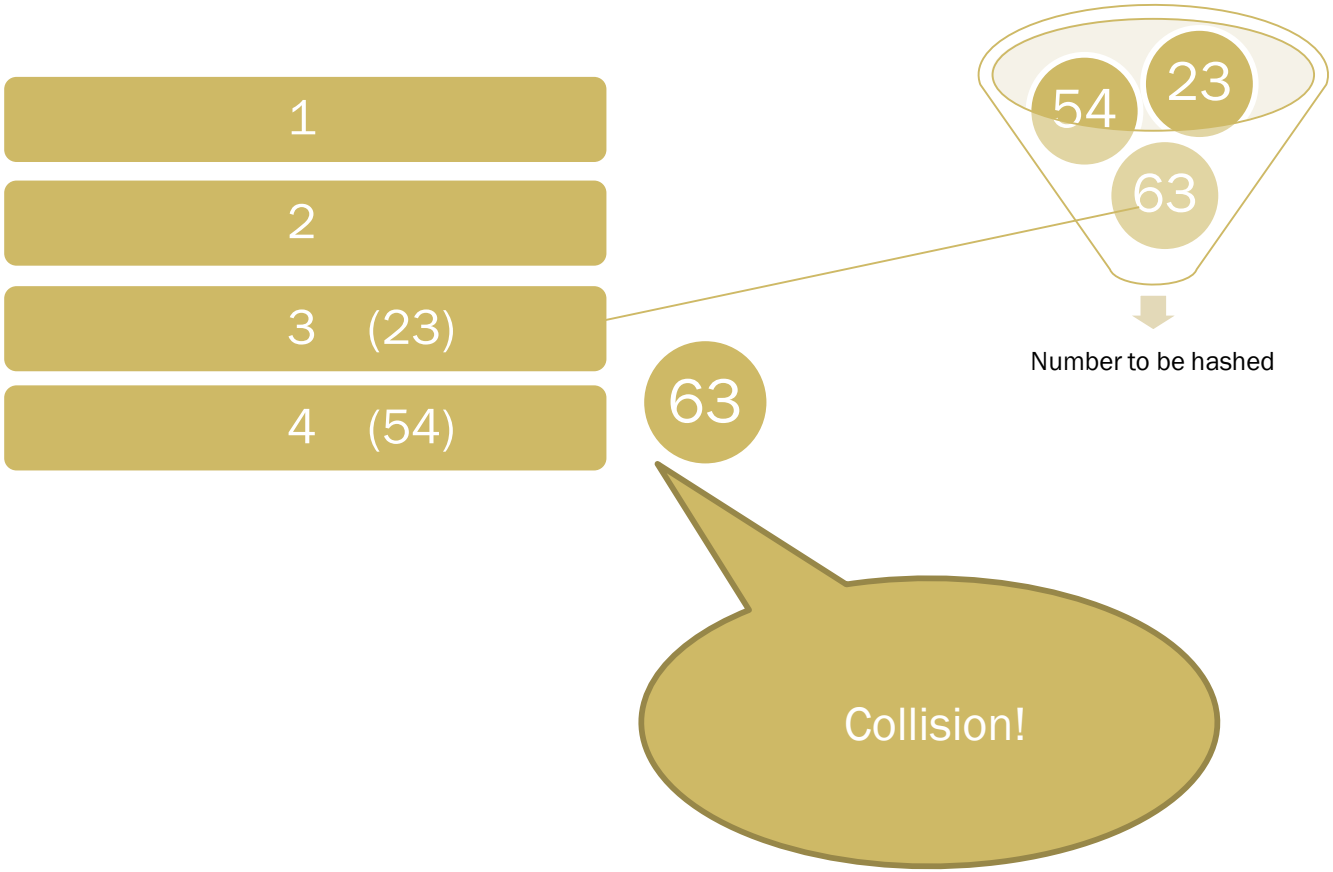
An Example



An Example



An Example

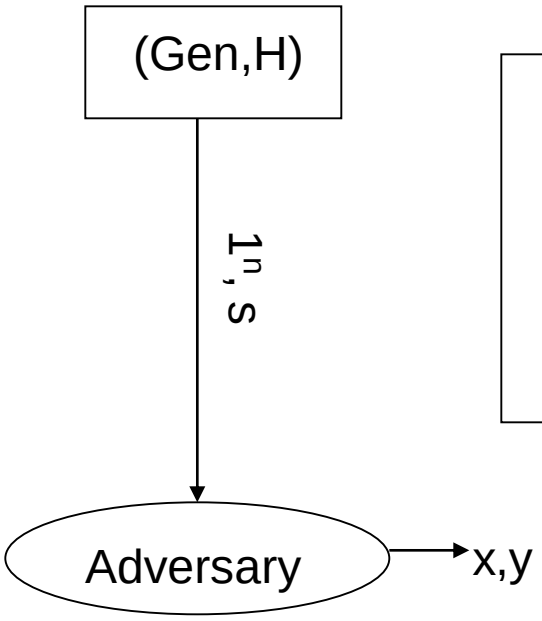


Collision Resistance

- A function H is *collision resistant* if it is infeasible for any probabilistic polynomial-time algorithm to find a collision in H
- We deal with a family of functions indexed by s, $H^s(x) = H(s,x)$
- A **hash function** is a pair of algorithms (Gen, H) where $Gen(1^n)$ outputs the index s (for choosing H^s)
- If H^s is defined only for inputs x of a certain length, we say it is a **fixed length** hash function

Defining Collision Resistant Hash Function

Hash-game



The output of Hash-game is 1 if and only if $x \neq y$ and $H^s(x) = H^s(y)$

A hash function (Gen,H) is collision resistant if for all probabilistic polynomial time adversaries A:

Pr[Output of Hash-game = 1] ≤ negl(n)

Birthday Attack

We can find a collision on a hash function by hashing random numbers until these hashed values have a collision

Relies on 'Birthday Paradox'

Theorem: If we pick independent random numbers in [1,2,...,N] with uniform distribution $\theta\sqrt{N}$ times, we get at least one number twice with probability tending to

$$(1-e^{-(\theta/2)*\theta})$$

For N = 365 and θ = 1.2 we get prob ~ 50%

Birthday Paradox: If there are 23 people,with probability at least ½ at least two of them have the same birthday

And for θ = 2.1 we get prob ~ 90% !

TEXTBOOK VERSION

LEMMA A.10 *Fix a positive integer N, and say $q \leq \sqrt{2N}$ elements $y_1, \ldots, y_q$ are chosen uniformly and independently at random from a set of size N. Then the probability that there exist distinct i, j with $y_i = y_j$ is at least $\frac{q(q-1)}{4N}$. That is,*

$$\mathsf{coll}(q, N) \geq \frac{q(q-1)}{4N}.$$

PROOF

- NoColl_i : Denotes the event of NO COLLISION upto i

$$\Pr[\text{NoColl}_q] = \Pr[\text{NoColl}_1] \cdot \Pr[\text{NoColl}_2 \mid \text{NoColl}_1] \cdots \Pr[\text{NoColl}_q \mid \text{NoColl}_{q-1}].$$

$$\Pr[\text{NoColl}_1] = 1 \qquad \Pr[\text{NoColl}_{i+1} \mid \text{NoColl}_i] = 1 - \frac{i}{N}$$

$$\Pr[\text{NoColl}_q] = \prod_{i=1}^{q-1} \left(1 - \frac{i}{N}\right)$$

PROPOSITION A.4 *For all x with $0 \le x \le 1$ it holds that*

$$e^{-x} \le 1 - \left(1 - \frac{1}{e}\right) \cdot x \le 1 - \frac{x}{2}.$$

$$\Pr[\text{NoColl}_q] \le \prod_{i=1}^{q-1} e^{-i/N} = e^{-\sum_{i=1}^{q-1}(i/N)} = e^{-q(q-1)/2N}$$

$$\Pr[\text{Coll}] = 1 - \Pr[\text{NoColl}_q] \ge 1 - e^{-q(q-1)/2N} \ge \frac{q(q-1)}{4N}$$

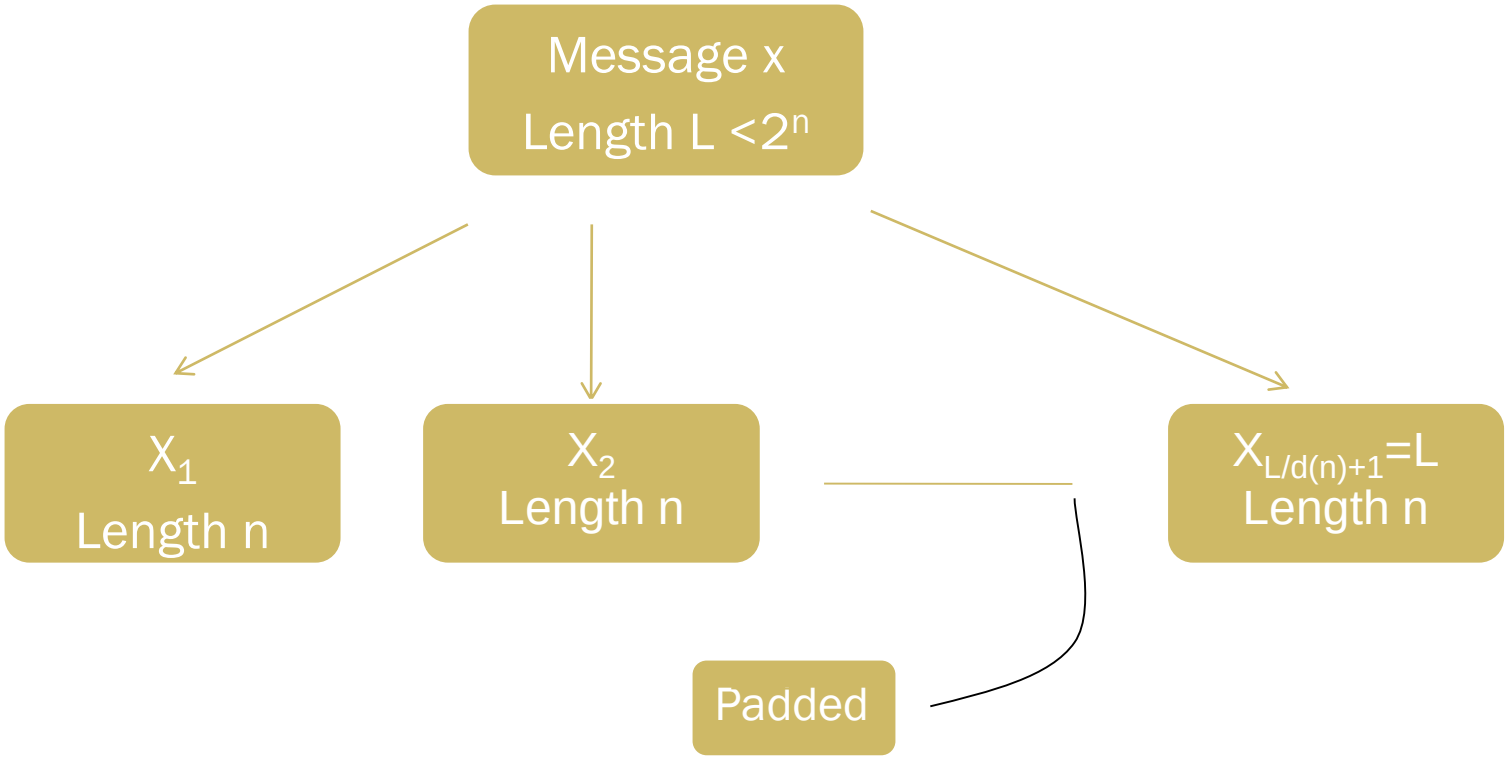
CONSTRUCTING COLLISION RESISTANT HASH FUNCTIONS

Step 1 (*Merkle-Damgard*): From Fixed Length Hash to Arbitrary Hash

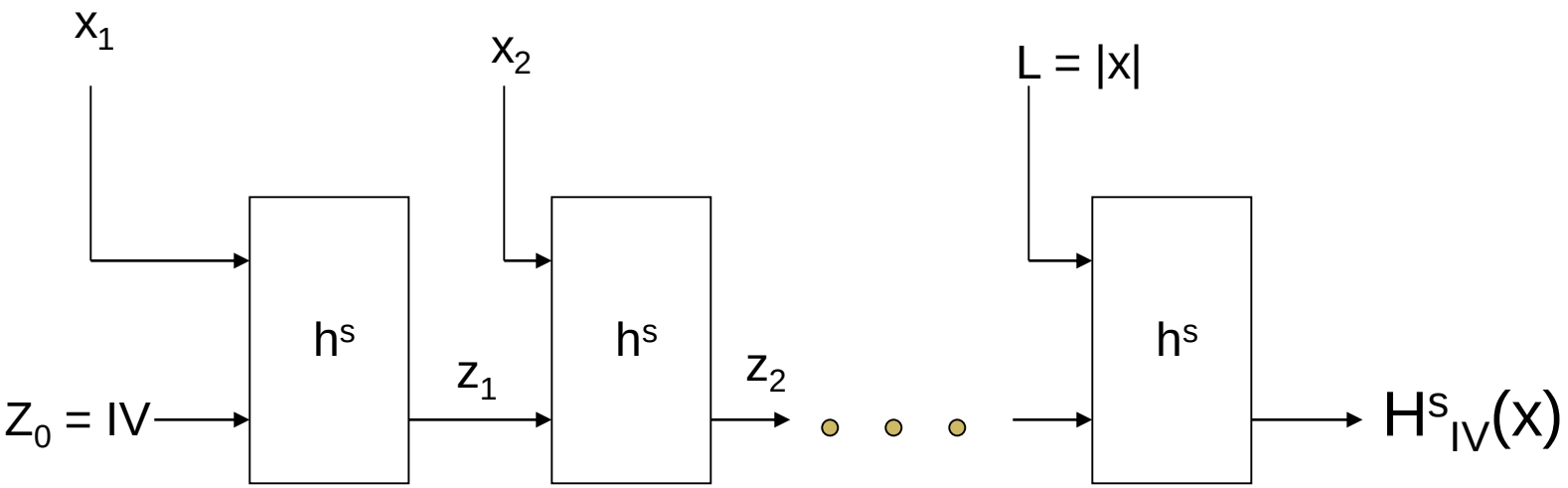
Step 2: From DLP to fixed length Hashing

Merkle Damgard Transform

Constructing hash functions $H^s(x)$ from fixed length hash functions (h^s) with inputs of length 2n and output length n



Merkle Damgard Transform



Theorem: If (Gen,h) is a fixed length collision resistant hash function, then (Gen, H) is a collision resistant hash function

TEXTBOOK VERSION

CONSTRUCTION 4.13

Let (Gen, h) be a fixed-length collision-resistant hash function for inputs of length $2\ell(n)$ and with output length $\ell(n)$. Construct a variable-length hash function (Gen, H) as follows:

- Gen: remains unchanged.
- H: on input a key s and a string $x \in \{0,1\}^*$ of length $L < 2^{\ell(n)}$, do the following (set $\ell = \ell(n)$ in what follows):

  1. Set $B := \lceil \frac{L}{\ell} \rceil$ (i.e., the number of blocks in x). Pad x with zeroes so its length is a multiple of ℓ. Parse the padded result as the sequence of ℓ-bit blocks $x_1, \ldots, x_B$. Set $x_{B+1} := L$, where L is encoded using exactly ℓ bits.
  2. Set $z_0 := 0^\ell$.
  3. For $i = 1, \ldots, B+1$, compute $z_i := h^s(z_{i-1} \| x_i)$.
  4. Output z_{B+1}.

The Merkle-Damgård transform.

SECURITY PROOF

1. *Case 1:* $L \neq L'$. In this case, the last step of the computation of $H^s(x)$ is $z_{B+1} := h^s(z_B \| L)$ and the last step of the computation of $H^s(x')$ is $z'_{B'+1} := h^s(z'_{B'} \| L')$. Since $H^s(x) = H^s(x')$ it follows that $h^s(z_B \| L) = h^s(z'_{B'} \| L')$. However, $L \neq L'$ and so $z_B \| L$ and $z'_{B'} \| L'$ are two different strings that collide for h^s.

2. *Case 2:* $L = L'$. Note this means that $B = B'$ and $x_{B+1} = x'_{B+1}$. Let z_i and z'_i be the intermediate hash values of x and x' during the computation of $H^s(x)$ and $H^s(x')$, respectively. Since $x \neq x'$ but $|x| = |x'|$, there must exist at least one index i (with $1 \leq i \leq B$) such that $x_i \neq x'_i$. Let $i^* \leq B + 1$ be the *highest* index for which it holds that $z_{i^*-1} \| x_{i^*} \neq z'_{i^*-1} \| x'_{i^*}$. If $i^* = B + 1$ then $z_B \| x_{B+1}$ and $z'_B \| x'_{B+1}$ are two different strings that collide for h^s because

$$h^s(z_B \| x_{B+1}) = z_{B+1} = H^s(x) = H^s(x') = z'_{B+1} = h^s(z'_B \| x'_{B+1}).$$

If $i^* \leq B$, then maximality of i^* implies $z_{i^*} = z'_{i^*}$. Thus, once again, $z_{i^*-1} \| x_{i^*}$ and $z'_{i^*-1} \| x'_{i^*}$ are two different strings that collide for h^s.

BULIDING MACs USING HASHING

Keyed Hashing

HMAC: A Message Authentication Code

HMAC is the current industry standard as CBC-MAC is deemed to be slow

(Gen,h): A fixed length hash function

(Gen,H): Hash function after applying MD transform to (Gen,h)

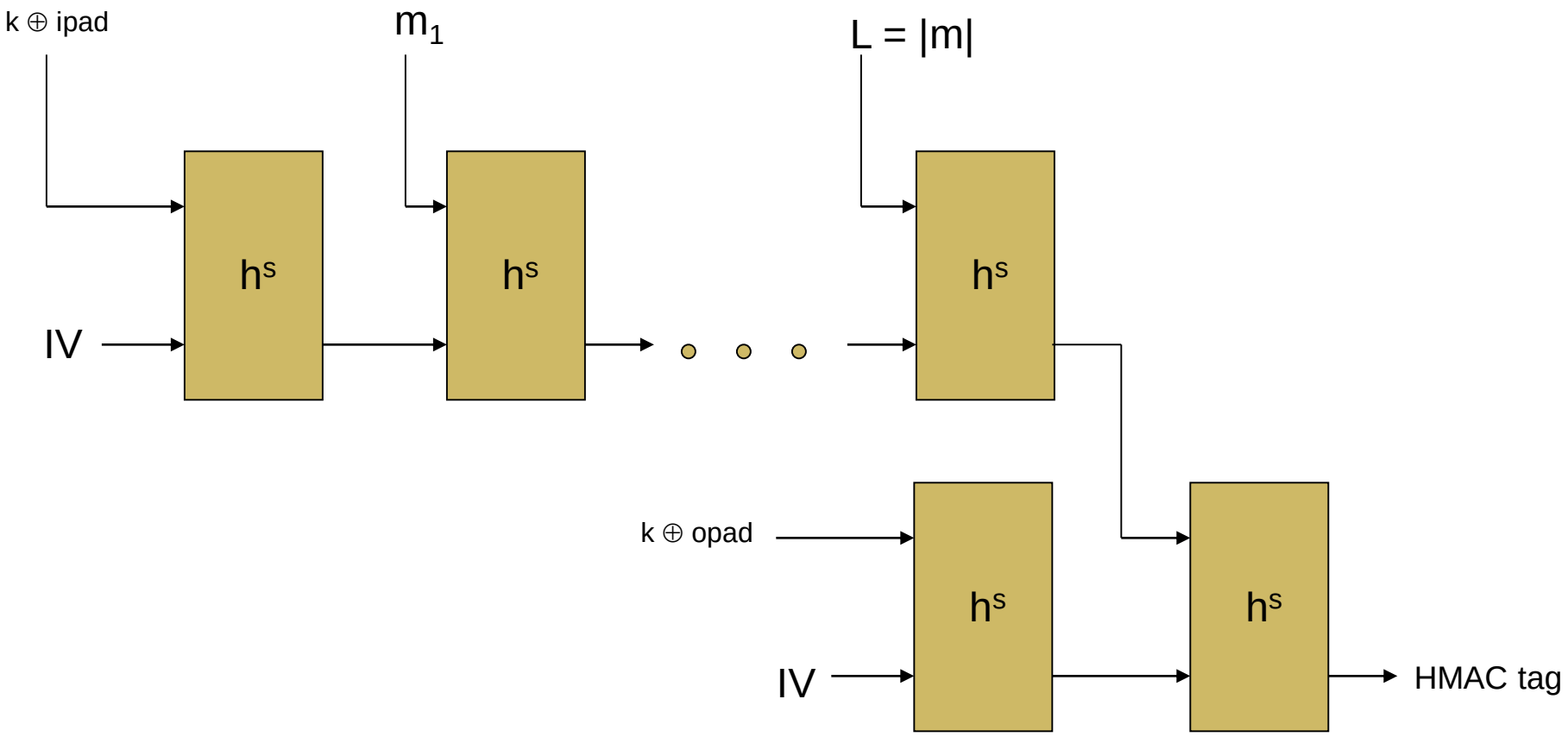
Fixed constants: *IV*, *opad* and *ipad*

HMAC tag for m = $H^s_{IV}((k \oplus opad) \,||\, H^s_{IV}((k \oplus ipad) \,||\, m))$

opad: 0x36 repeated as many times as needed

ipad: 0x5C repeated as many times as needed

HMAC Construction



Collision Resistant Hash Functions in Practice

- Popular practical constructions
  - MD5 (broken in 2004, should no longer be used)
  - SHA-1, SHA-2 etc. (uses Merkle-Damgard transform) and others
- Theoretical Constructions
  - Based on hardness of the discrete logarithm problem

A Fixed Length Hash Function

Let P be a polynomial time algorithm that on input 1^n outputs a cyclic group G of order q (length of q is n) and generator g

Gen: Run $P(1^n)$ obtain (G,q,g); select uniformly at random an element h from G; Output **s = (G,q,g,h)**

H: On input x_1 and x_2 (both in the range 0 to q-1), output

$$H^s(x_1,x_2) = g^{x_1} h^{x_2}$$

Theorem: If discrete logarithm problem is hard then the above is a fixed length collision resistant hash function

TEXTBOOK VERSION

CONSTRUCTION 7.72

Let $\mathcal{G}$ be as described in the text. Define a fixed-length hash function (Gen, H) as follows:

- Gen: on input 1^n, run $\mathcal{G}(1^n)$ to obtain $(\mathbb{G}, q, g)$ and then select $h \leftarrow \mathbb{G}$. Output $s := \langle \mathbb{G}, q, g, h \rangle$ as the key.
- H: given a key $s = \langle \mathbb{G}, q, g, h \rangle$ and input $(x_1, x_2) \in \mathbb{Z}_q \times \mathbb{Z}_q$, output $H^s(x_1, x_2) := g^{x_1} h^{x_2}$.

A fixed-length hash function.

SECURITY PROOF

THEOREM 7.73 *If the discrete logarithm problem is hard relative to $\mathcal{G}$, then Construction 7.72 is a fixed-length collision-resistant hash function*

$$H^s(x_1, x_2) = H^s(x_1', x_2') \Rightarrow g^{x_1} h^{x_2} = g^{x_1'} h^{x_2'}$$
$$\Rightarrow g^{x_1 - x_1'} = h^{x_2' - x_2}.$$

$$\Delta \stackrel{\text{def}}{=} x_2' - x_2$$

$$g^{(x_1 - x_1') \cdot \Delta^{-1}} = \left(h^{x_2' - x_2}\right)^{[\Delta^{-1} \bmod q]} = h^{[\Delta \cdot \Delta^{-1} \bmod q]} = h^1 = h.$$

TASKS (till date)

Assuming Discrete Logarithm Problem to be a one-way

- Build a provably secure PRG
- Build a provably secure PRF from PRG
- Use the PRF in some secure mode of operation to obtain a CPA-secure encryption scheme
- Use the PRF to build a secure MAC
- Use CPA-security and secure MAC to design a state-of-the-art provably CCA-secure encrytpion scheme

- Using DLP to build fixed-length Collision Resistant Hash function
- Using MD transform to obtain a (provably secure) Collision Resistant Hash Function
- Using a collision resistant hash function to build H-MACs

THANK YOU

Any Questions?