

# Part I: Crypto

# Chapter 2: Crypto Basics

MXDXBVTZWVMXNSPBQXLIMSCCSGXSCJXBOVQXCJZMOJZCVC  
TVWJCZAAXZBCSSCJXBQCJZCOJZCNSPOXBXSBTWJC  
JZDXGXXMOZQMSCSCJXBOVQXCJZMOJZCNSPJZHGXXMOSPLH  
JZDXZAAXZBXHCSCJXTCSGXSCJXBOVQX

— plaintext from Lewis Carroll, *Alice in Wonderland*

The solution is by no means so difficult as you might be led to imagine from the first hasty inspection of the characters.

These characters, as any one might readily guess, form a cipher — that is to say, they convey a meaning...

— Edgar Allan Poe, *The Gold Bug*

# Crypto

- **Cryptology** — The art and science of making and breaking “secret codes”
- **Cryptography** — making “secret codes”
- **Cryptanalysis** — breaking “secret codes”
- **Crypto** — all of the above (and more)

---

# How to Speak Crypto

- A *cipher* or *cryptosystem* is used to *encrypt* the *plaintext*
- The result of encryption is *ciphertext*
- We *decrypt* ciphertext to recover plaintext
- A *key* is used to configure a cryptosystem
- A *symmetric key* cryptosystem uses the same key to encrypt as to decrypt
- A *public key cryptosystem* uses a *public key* to encrypt and a *private key* to decrypt

واژه های دیگه:

cipher or cryptosystem که به معنای سیستم رمز یا رمزنگاری است که کاری که میکنه اینه که plaintext رو می گیره و رمزش میکنه --> ینی روشی هست که رو میگیره و ciphertext تحویل ما میده ینی متن رمز رو تحویل ما میده اصل کیرشهف میگه تنها چیزی که trudy نمیدونه کلید ما است ولی بقیه چیزا رو باجزئیات می دونه trudy توی سیستم symmetric key هر دو طرف ارتباط از یک کلید مشترک دارند استفاده می کنند public key ینی هر کسی دو تا کلید داره که یکیش private است که خودش نمیدونه و یکی است که بقیه دارن

# Crypto

- Basic assumptions
  - The system is completely known to the attacker
  - Only the key is secret
  - That is, crypto algorithms are not secret
- This is known as **Kerckhoffs' Principle**
- Why do we make such an assumption?
  - Experience has shown that secret algorithms tend to be weak when exposed
  - Secret algorithms never remain secret
  - Better to find weaknesses beforehand

## رمزنگاری

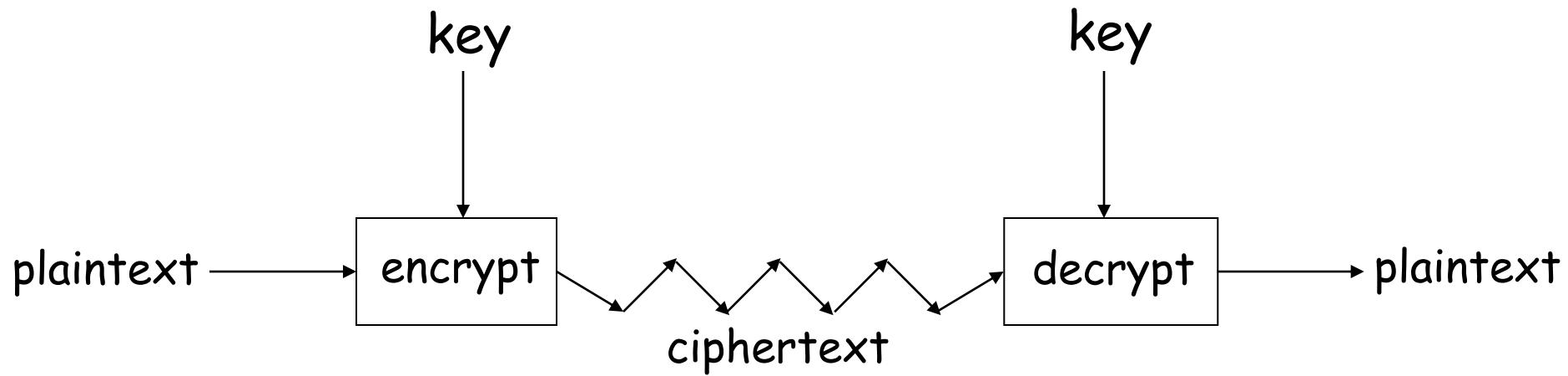
- مفروضات اساسی --> اولین فرض اساسی که میکنیم اینه که:
  - سیستم کاملاً برای مهاجم یا **trudy** شناخته شده است
  - فقط کلید راز است ینی **trudy** فقط کلید رو نمی دونه
  - یعنی الگوریتم های رمزنگاری مخفی نیستند --> چرا این اصل است؟ ینی اصل کیرشهف هست؟ چرا به **trudy** این همه جزئیات میدیم؟ ینی الگوریتم در اختیارش است چرا این کارو میکنیم؟

- یک سری ادم خوب دیگه هستند که باعث میشن این الگوریتم ما بهتر بشه ینی اگه باگی داشت ادم خوب ها بیاین رفعش کن
- فرض رو بر این می ذاریم که مهاجم الگوریتم رو می بینه و با این فرض الگوریتم رو درست می کنیم --> ینی می تونیم با مهندسی معکوس مهاجم به الگوریتمش پی بره پس هر چقدر هم مخفیش کنیم باز مهاجم می فهمه
- نکته: ولی خیلی ها از این اصل پیروی نمیکنند مثل مایکروسافت که توی یک ماه خیلی اسیب پذیری داره

## □ این به عنوان اصل کیرشهف شناخته می شود

- چرا چنین فرضی می کنیم؟
  - تجربه نشان داده است که الگوریتم های مخفی در هنگام افشاری ضعیف تمایل دارند
  - الگوریتم های مخفی هرگز مخفی نمی مانند
  - بهتر است نقاط ضعف را از قبل پیدا کنید

# Crypto as Black Box



A generic view of symmetric key crypto

نمای کلی از رمزنگاری کلید متقاض

- نکات:

- 1- توی کل ترم به اصل کیرشef پاییند هستیم
- 2- توی این ساز و کار trudy , encrypt فقط کلید رو نداره ینی trudy ما ciphertext , encrypt داره ولی key رو نداره و به همین خاطر همین هم decrypt رو نداره

# Simple Substitution

- ❑ Plaintext: fourscoreandsevenyearsago
- ❑ Key:

Plaintext	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
Ciphertext	D E F G H I J K L M N O P Q R S T U V W X Y Z A B C

- ❑ Ciphertext:  
IRXUVFRUHDQGVHYHQBDUVDJR
- ❑ Shift by 3 is “Caesar’s cipher”

4 تا سیستم رمز کلاسیک:

-1 simple substitution ینی جانشینی ساده:

ینی به ازای هر حرف یک حرفی جانشینی شده و توی این روش این انتقال با شیفت انجام شده نکته: اگر سه تا شیفت کرده باشیم این میشه همون رمز سزارمون ینی Caesar's cipher

# Caesar's Cipher Decryption

- ❑ Suppose we know a Caesar's cipher is being used:

Plaintext	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
Ciphertext	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C

- ❑ Given ciphertext:

VSRQJHEREVTXDUHSDQWV

- ❑ Plaintext: spongebobsquarepants

- اینجا فرض شده رمز ما سزاره و **chipertext** هم دارم حالا می خوایم به **plaintext** برسیم یعنی بشکونیمش پس برای این کار 3 تا می ریم عقب برای هر حرف

نکته: این شیفت لزومی نداره سه تا باشه می تونه از 0 تا 25 تا باشه و اینجا کلید ما  $n$  است  
یعنی همون تعداد شیفت میشه

# Caesar's Cipher

- Shift by  $n$  for some  $n \in \{0, 1, 2, \dots, 25\}$
- Then key is  $n$
- Example: key  $n = 7$

Plaintext

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
H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G

Ciphertext

فضای کلید اینجا خیلی کوچیکه چون 26 تا دونس و برای trudy خیلی راحته که بیاد حمله جستجو کامل کنه

پس

باید فضای کلید رو خیلی بزرگ کنیم

لزومی نداره هر حرفی ترتیب داشته باشه مثلا  $a$  بشه  $h$  اینطوری برای  $a$  ما 26 حالت داریم و برای  $b$  میشه 25 تا پس فضای کلید میشه 26! توی این حالت باید هر دو طرف بدونن که  $a$  به چی نگاشت شده و  $b$  به چی و به همین صورت

# Cryptanalysis I: Try Them All

- We know Caesar's cipher (shift by n) used
  - But the specific key is unknown
- Given ciphertext: **CSYEVIXIVQMREXIH**
- How to determine the key?
- Only 26 possible keys — try them all!
- **Exhaustive key search**
- Solution: key is  $n = 4$

---

# Simple Substitution: General Case

- In general, simple substitution key can be any **permutation** of letters
  - Not necessarily a Caeser's cipher (shift)
- For example

Plaintext	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
Ciphertext	J	I	C	A	X	S	E	Y	V	D	K	W	B	Q	T	Z	R	H	F	M	P	N	U	L	G	O

- In general,  $26! > 2^{88}$  possible keys

---

# Cryptanalysis II: Be Clever

- We know that a simple substitution used
- But *not* necessarily a Ceasar's cipher (shift)
- Find the key given the ciphertext:

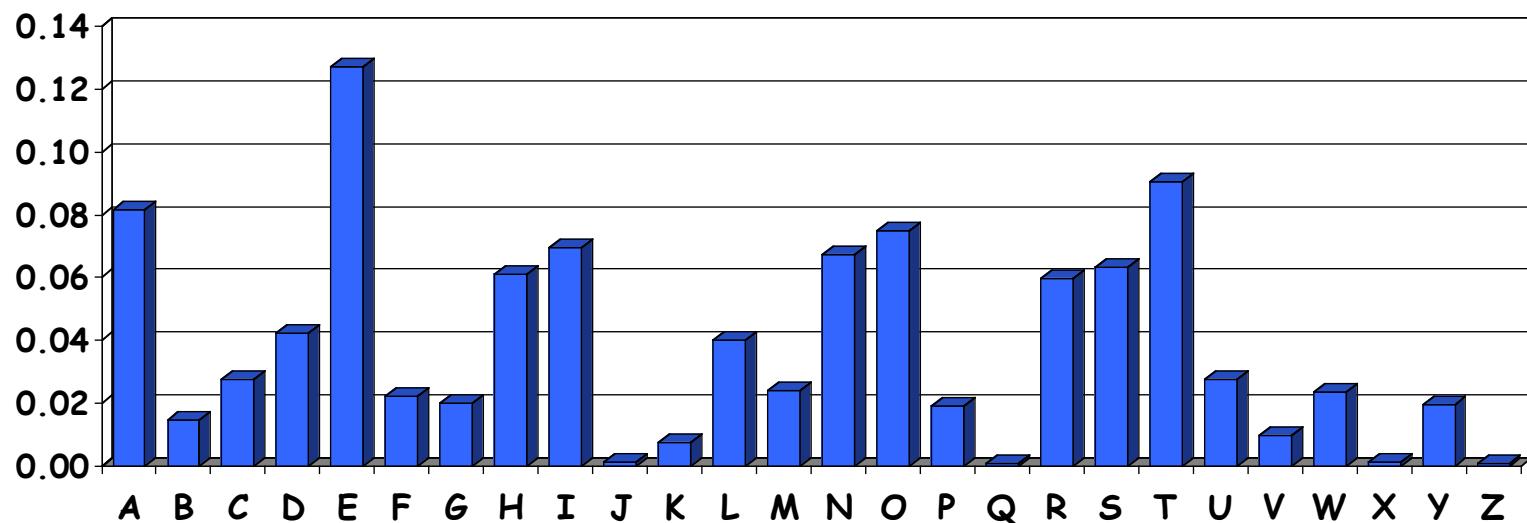
PBFPVYFBQXZTYFPBF EQJHDXXQVAPTPQJKTOYQWIPBVWLXTOX  
BTFXQWAXBVCXQWAXFQJVWLEQNT OZQGGQLFXQWA KVWLXQ  
WAEBIPBFXFQVXGTVJVWLBT P QWAEBFPBFHCVLXBQUFEVWLXGD  
PEQVPQGVPPBFTIXPFHXZHVFAGFO THFEFBQUFTDHZBQPOTHXTY  
FTODXQHFTDPTO GHFQP BQWAQJJTODXQHFOQPWTBDHHIXQV  
APBFZQHCFWPFHPBFIPBQWKFABVYYDZBOTHPBQPQJTQOTOGHF  
QAPBFEQJHDXXQVAVXE BQPEFZBVFOJIWFFACFCCFHQWAUVWF  
LQHGFVXVA FXQHFUFHILTTAVWAFFAWTEVOITDHFHFQAITIXPFH  
XAFQHEFZQWGFLVWPTOFFA

---

.

# Cryptanalysis II

- Cannot try all  $2^{88}$  simple substitution keys
- Can we be more clever?
- English letter frequency counts...



---

# Cryptanalysis II

## □ Ciphertext:

PBFPVYFBQXZTYFPBFEQJHDXXQVAPTPQJKTOYQWIPBVWLXTOBTFXQ  
WAXBVCXQWAXFQJVWLEQNTOZQGGQLFXQWAKVWLXQWAEBIPBFXFQ  
VXGTVJVWLBTTPQWAEBFPBFHCVLXBQUFEVWLXGDPEQVPQGVPPBFTIXPFH  
XZHVFAFGOTHFEFBQUFTDHZBQPOTHXTYFTODXQHFTDPTOGHFQPBQW  
AQJJTODXQHFOQPWTBDHHIXQVAPBFZQHCFWPFBFIPBQWKFABVYY  
DZBOTHPBQPQJTQOTOGHFQAPBFEQJHDXXQVAVXEBCPEFZBVFOJIWFF  
ACFCCFHQWAUVWFLQHGFXVAFXQHFUFHILTTAVWAFFAWTEVOITDHFH  
FQAITIXPFHXAFQHEFZQWGFLVWPTOFFA

## □ Analyze this message using statistics below

### Ciphertext frequency counts:

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
21	26	6	10	12	51	10	25	10	9	3	10	0	1	15	28	42	0	0	27	4	24	22	28	6	8

فرکانس هر کدوم از این حرفارو میاد به دست میاره و مثلا می بینه چند درصد a استفاده شده و .. او ن متن رو هم داره و تهش به جدول صفحه رو برو می رسه و اینجا F از همه بیشتره پس می تونیم بگیم E نگاشت شده به F چون توی صفحه قبل دیدیم فرکانس E خیلی بوده و اینجا هم F خیلی زیاده پس در نتیجه F میشه E دونه دونه با این کلمات بازی میکنه و می بینه ایا جمله با معنی از توش درمیاد یا نه trudy

# Cryptanalysis: Terminology

- Cryptosystem is **secure** if best known attack is to try all keys
  - Exhaustive key search, that is
- Cryptosystem is **insecure** if **any** shortcut attack is known
- But then insecure cipher might be harder to break than a secure cipher!
  - What the ... ?

---

نکته: توی طراحی سیستم رمز باید فضای کلید اول بزرگ باشه و دوم اینکه سعی کنیم یه کاری بکنیم که بهترین راه برای مهاجم این باشه که بخواهد دونه به دونه بیاد اینارو تست کنه

# Vigenere Cipher

- Simple substitution is **monoalphabetic**
- Vigenere cipher is simple example of a **polyalphabetic** substitution
  - Caesars ciphers, based on a keyword
- For example, keyword CAT indicates shift by 2, shift by 0, shift by 19
  - Then repeat as needed

---

توی این روش میگه هر حرفی بیاد به چندتا حرف نگاشت بشه ابهام برای دو طرف از طریق  
کلمه keyword حل میشه اینجا می شه CAT

# Vigenere Example

- Suppose that we want to encrypt  
attackatdawn
- Encryption:  
رمزگذاری

keyword:	CATCATCATCAT
plaintext:	attackatdawn
ciphertext:	ctmcccdctwcwg
- Ciphertext is ctmcccdctwcwg
- How to decrypt? How to attack?

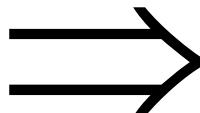
چگونه اینو بشکنیم؟ اون بحث اماری (فرکانسها) توی keyword وجود داره --> اینجا روی حمله میشه ینی میاد تعداد زیادی شنود میکنه و تهش حدس میزنه

# Double Transposition

- Plaintext: attackxatxdawn

	col 1	col 2	col 3
row 1	a	t	t
row 2	a	c	k
row 3	x	a	t
row 4	x	d	a
row 5	w	n	x

Permute rows  
and columns



	col 1	col 3	col 2
row 3	x	t	a
row 5	w	x	n
row 1	a	t	t
row 4	x	a	d
row 2	a	k	c

- Ciphertext: xtawxnattxadakc
- Key is matrix size and permutations:  
(3,5,1,4,2) and (1,3,2)

انتقال دوگانه:

این یکی از اون 4 رمز کلاسیک است

اینجا او مده به جای اسپیس  $\times$  گذاشته ولی نیاز نیست چون در این حالت تعداد  $\times$  ها زیاد میشه و مهاجم می فهمه هر جا که  $\times$  یک کلمه دیگه داره شنود میشه

اینجا میاد توی یک ماتریسی اینارو میچینه و بعد میاد سطر و ستون ها رو قاطی میکنه

اینجا کلید اینه که کدوم سطر با کدوم ستون جایه جا شده و سایز ماتریسمون چند در چند (این

بین دو نفر مشترکه) --> این چیزیه که گیرنده پیام باید بدونه

نکته: اگر مهاجم بتونه یک تیکه از این ماتریس رو مرتب کنه فقط همون یه تیکه نیست و خیلی چیزای دیگه رو می فهمه

# One-Time Pad: Encryption

e=000 h=001 i=010 k=011 l=100 r=101 s=110 t=111

رمزگذاری **Encryption: Plaintext  $\oplus$  Key = Ciphertext**

h e i l h i t l e r

Plaintext: 001 000 010 100 001 010 111 100 000 101

Key: 111 101 110 101 111 100 000 101 110 000

Ciphertext: 110 101 100 001 110 110 111 001 110 101

s r l h s s t h s r

امن ترین رمزنگاری هست که ما توی کل عمر مون خواهیم دید on time pad

اینجا یه الفبای 8 تایی داریم که با سه تا بیت می تونیم نشونش بدیم --> می تونیم بیشتر از این هم حرف داشته باشیم اینجا مثلا 8 حرف گرفتیم  
نکته: همون کلیدی که فرستنده داره، گیرنده هم داره

# One-Time Pad: Decryption

e=000 h=001 i=010 k=011 l=100 r=101 s=110 t=111

رمزگشایی

Decryption: Ciphertext  $\oplus$  Key = Plaintext

	s	r		h	s	s	t	h	s	r
Ciphertext:	110	101	100	001	110	110	111	001	110	101
Key:	111	101	110	101	111	100	000	101	110	000
Plaintext:	001	000	010	100	001	010	111	100	000	101
	h	e	i		h	i	t		e	r

نکته:

فرض میکنیم Alice اینجا یک جاسوس دو جانبی است این میخواهد پیام بده به المان و یک جاسوس خوب است و trudy هم میشه بریتانیا و بریتانیا شک کرده به Alice و این پیام را از Alice گرفته

و Alice می تونه یک کلید بسازه و اون پیام رو بکنه kill hitler پس او مد یه چیزی ساخت که معنی داره و به نفع خودش باشه ینی بحث انکار

این مثل one time pad Simple Substitution نیست که می او مدیم دونه به دونه کلیدها رو تست میکردیم و می دیدیم اینجا معنی دار شد و می گفتیم احتمالاً این کلیده ولی اینجا هر کلیدی می تونه یک پیام معنی داره بسازه ینی می تونه ادعا کنه که ما اینو گفتیم و این موضوع باعث امنیتش میشه

توی این روش هر چیزی رو می تونه ادعا کنه که به نفع خودش باشه اینجا مهاجم نمی تونه از طریق جستجوی کامل فضای کلید به چیز خاصی بررسه چون ممکنه هر چیزی در بیاد

پس چرا از این روش استفاده نمیکنیم؟  
چون انتقال کلیدمون سخت است  
نکات:

اینجا ciphertext هیچ اطلاعات بدردخوری راجع به plaintext به ما نمیده هر plaintext رو میتوانیم داشته باشیم اما

به شرطی که on time pad رو به درستی استفاده بکنیم و شرط اولش اینه که این pad ما باید رندوم باشه

و نکته بعدی اینه که این pad باید از طریق یک کانالی بین گیرنده و فرستنده به اشتراک گذاشته بشد --> ادامش صفحه بعدی

# One-Time Pad

Double agent claims following “key” was used:

	s	r		h	s	s	t	h	s	r
Ciphertext:	110	101	100	001	110	110	111	001	110	101
“key”:	101	111	000	101	111	100	000	101	110	000
“Plaintext”:	011	010	100	100	001	010	111	100	000	101
	k	i			h	i	t		e	r

e=000 h=001 i=010 k=011 l=100 r=101 s=110 t=111

ادعاهای نماینده دوگانه زیر "کلید" استفاده شد:

ادامه صفحه بالایی:

اگه ما می تونیم برای 100 حرف مثلا 300 بیت کلید رو به صورت امن انتقال بدیم خب میایم پیام رو انتقال بدیم:) اصلا چه کاریه این:)

اینجا کلید یه بار استفاده میشه و کلید رندوم است که هم اندازه و هم طول `plaintext` مون است

# One-Time Pad

Or, might claim the key is... ممکن است ادعا کند که کلید این است

	s	r		h	s	s	t	h	s	r
Ciphertext:	110	101	100	001	110	110	111	001	110	101
"key":	111	101	000	011	101	110	001	011	101	101
"Plaintext":	001	000	100	010	011	000	110	010	011	000
	h	e		i	k	e	s	i	k	e

e=000 h=001 i=010 k=011 l=100 r=101 s=110 t=111

---

# One-Time Pad Summary

- **Provably** secure
  - Ciphertext gives **no** useful info about plaintext
  - All plaintexts are **equally likely**
- BUT, only when be used correctly
  - Pad must be random, used only once
  - Pad is known only to sender and receiver
- Note: pad (key) is same size as message
- So, why not distribute message itself, instead of the pad?

---

# Real-World One-Time Pad

- Project VENONA
  - Soviet spies encrypted messages from U.S. to Moscow in 30's, 40's, and 50's
  - Nuclear espionage, etc.
  - Thousands of messages
- Spy carried one-time pad into U.S.
- Spy used pad to encrypt secret messages
- Repeats within the “one-time” pads made cryptanalysis possible

اما از این روش استفاده شده مثلا المانی ها ازش استفاده می کردند و مهمترین استفادش مال شوروی بود --> از روش on time pad او جش بحث هسته ای است

مثلا این جاسوس یک کتاب با خودش می برد و هر کجا که نیاز میشد می دونست از این تیک تا اون تیکشو باید استفاده کنه به عنوان pad اشکالش این بود که:

1- توی این one time pad ها وقتی می خواستیم یک رشته خیلی بزرگی رو بصورت رندوم تولید کنیم خود این دیگه معمولا رندوم نیست

2- و توی این one time pad هی تکرار میشد یعنی توی این رشته طولانی یه سری زیررشته ها بود که هی تکرار میشد و این باعث شد که این بشکنه

# VENONA Decrypt (1944)

[C% Ruth] learned that her husband [v] was called up by the army but he was not sent to the front. He is a mechanical engineer and is now working at the ENORMOUS [ENORMOZ] [vi] plant in SANTA FE, New Mexico. [45 groups unrecoverable]

detain VOLOK [vii] who is working in a plant on ENORMOUS. He is a FELLOWCOUNTRYMAN [ZEMLYAK] [viii]. Yesterday he learned that they had dismissed him from his work. His active work in progressive organizations in the past was cause of his dismissal. In the FELLOWCOUNTRYMAN line LIBERAL is in touch with CHESTER [ix]. They meet once a month for the payment of dues. CHESTER is interested in whether we are satisfied with the collaboration and whether there are not any misunderstandings. He does not inquire about specific items of work [KONKRETNAYa RABOTA]. In as much as CHESTER knows about the role of LIBERAL's group we beg consent to ask C. through LIBERAL about leads from among people who are working on ENOURMOUS and in other technical fields.

- "Ruth" == Ruth Greenglass
- "Liberal" == Julius Rosenberg
- "Enormous" == the atomic bomb

-  
این متن رمز رو شکستن

سه تا رمز تا اینجا گفته شد:

Simple Substitution -1

Double Transposition -2

one time pad -3

# Codebook Cipher

- ❑ Literally, a book filled with “codewords”
- ❑ Zimmerman Telegram encrypted via codebook

Februar	13605
fest	13732
finanzielle	13850
folgender	13918
Frieden	17142
Friedenschluss	17149
:	:

- ❑ Modern block ciphers are codebooks!
- ❑ More about this later...

نکته: **codebook** از جنس رمزهای جانشینی است اینجا به جای اون الفبا کلمه داریم اون موقع یک کتاب بوده و امنیت ما بسته به این بوده که این کتاب به دست دشمن بیوافته یا نه پس از این کتاب خیلی مراقبت می کردند

نکته: اگر پیام رو فاش بفرستیم امن تر از اینکه پیام را با یک رمز بد بفرستیم چون اگه با یک الگوریتم رمز بد بکنیم طرف مقابل شک میکنه به ما پس پیامی که همین محتوا رو داشت ولی فاش بود اونو واسه امریکا ارسال کردند

# Codebook Cipher: Additive

- Codebooks also (usually) use **additive**
- Additive — book of “random” numbers
  - Encrypt message with codebook
  - Then choose position in additive book
  - Add additive sequence to get ciphertext
  - Send ciphertext and additive position (MI)
  - Recipient subtracts additives before decrypting
- Why use an additive sequence?

- 1- امنیت ما کلا توی این روش بسته به اون کتابه بود که دست کسی نیوفته

- 2- از طرفی ما روی رمز جانشینی حمله داشتیم و اون هم اماری بود روی codebook ها هم اینه که بیایم کلمه ها رو بشماریم

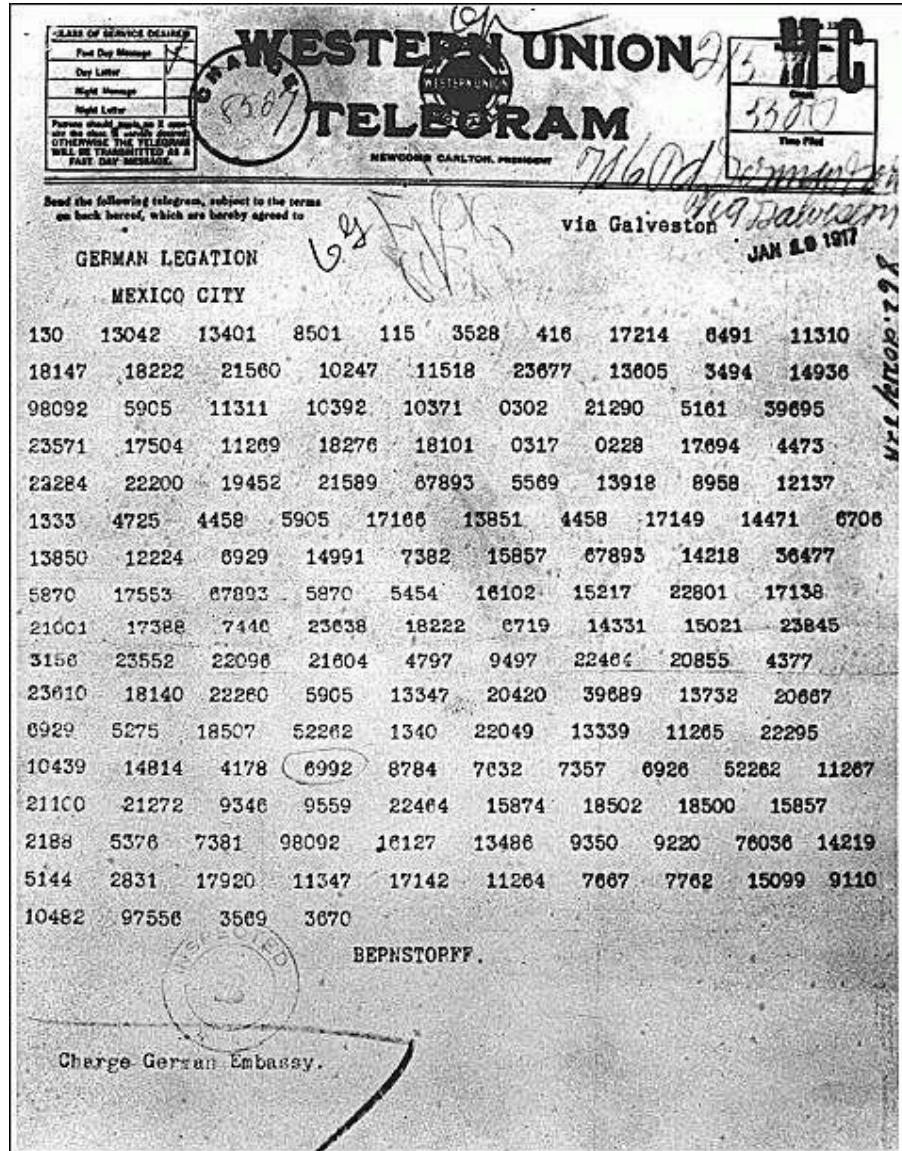
چون کلمه ها رو باید بشماریم پس باید trudy متن های بیشتری رو شمرده باشه و تحلیل کرده باشه تا بتونه بفهمه این کلمه الان معادل چیه

به این دو دلیل نیاز بود codebook هی عوض بشه و اینکه یکی بیاد طراحی کنه و یکی codebook پخش بشه خطرناک بود پس به جای عوض کردن اومدن گفتن که با یک کتاب به اسم additive میایم

مثلا ما میخوایم یک جمله 5 کلمه ای رو رمزش کنیم: اولا می ریم سراغ codebook و عدهای متناظرشو در میاریم بعد می ریم توی additive از یه جای دلخواه 5 تا عدد به صورت رندوم که پشت سر هم اند بر می داریم و جمع می کنیم با اون عدداي از اون codebook پس دوتا لایه داریم فقط فرستنده به صورت فاش میگه به گیرنده که من مثلا از این صفحه و از این خط 5 تا عدد برداشتم --> اون حمله اماری اینجا ممکن نیست

# Zimmerman Telegram

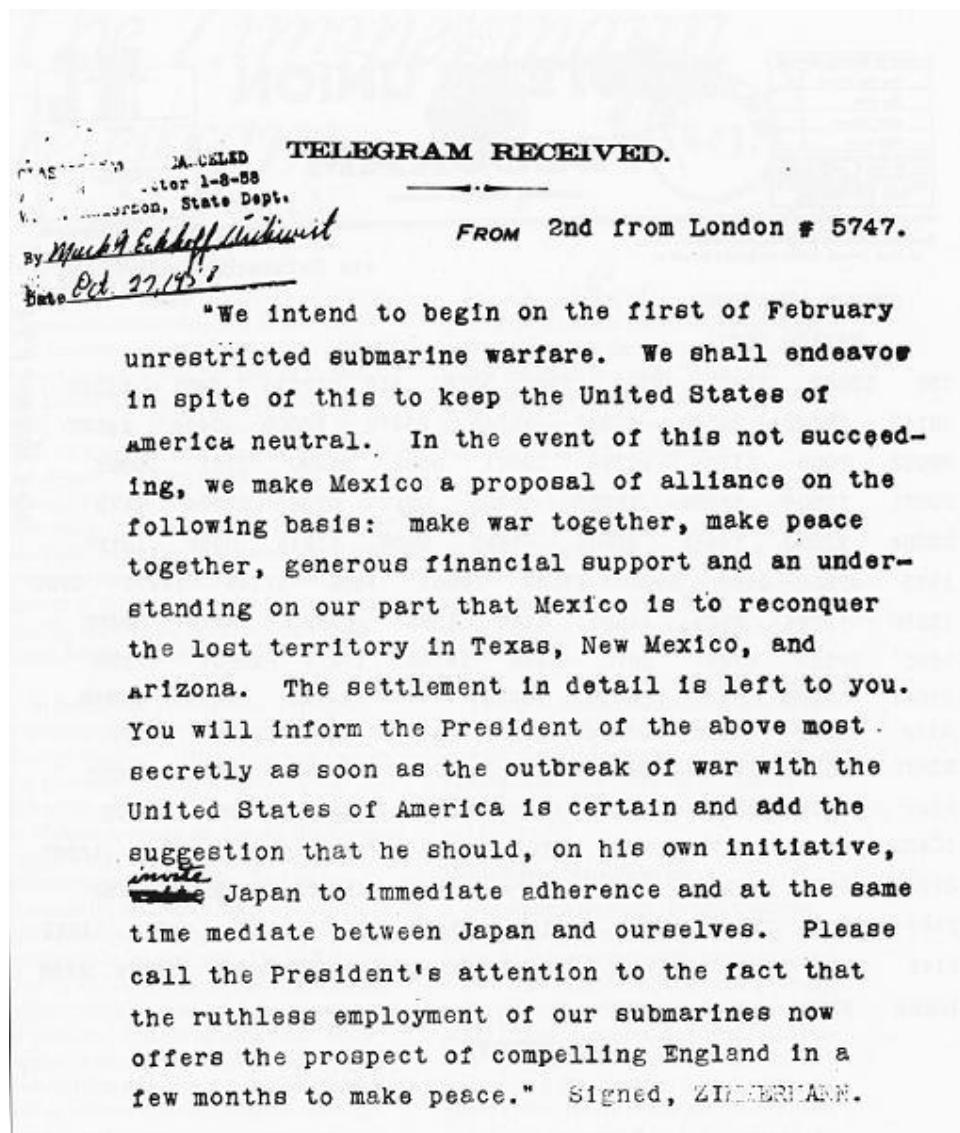
- ❑ Perhaps most famous codebook ciphertext ever
- ❑ A major factor in U.S. entry into World War I



---

# Zimmerman Telegram Decrypted

- British had recovered partial codebook
- Then able to fill in missing parts



---

# Early 20th Century

- WWI — Zimmerman Telegram
- "Gentlemen do not read each other's mail"
  - Henry L. Stimson, Secretary of State, 1929
- WWII — **golden** age of cryptanalysis
  - Midway/Coral Sea
  - Japanese **Purple** (codename MAGIC)
  - German **Enigma** (codename ULTRA)

---

# Post-WWII History

- Claude Shannon — father of the science of information theory
- Computer revolution — lots of data to protect
- Data Encryption Standard (DES), 70's
- Public Key cryptography, 70's
- CRYPTO conferences, 80's
- Advanced Encryption Standard (AES), 90's
- The crypto genie is out of the bottle...

---

# Claude Shannon

- Founded field of information theory
- His 1949 paper: *Comm. Thy. of Secrecy Systems*
- Fundamental concepts
  - **Confusion** — obscure relationship between plaintext and ciphertext
  - **Diffusion** — spread plaintext statistics through the ciphertext
- Proved one-time pad is secure
- One-time pad is confusion-only, while double transposition is diffusion-only

کلود شانون بحث رمز رو کاملا علمیش کرد و یک تئوری رو گفت به اسم تئوری اطلاعات  
کلود شانون رو برآمون خیلی مهمه چون یک هنر رو تبدیل کرد به علم  
دوتا مفهوم خیلی مهم رو گفت:

Confusion: ارتباط بین plaintext و ciphertext باید مبهم باشه مثل رمز otp ,  
simple transposition <--> این کافی است

diffusion: میگه بیا plaintext رو توی ciphertext پخشش کن مثل دابل ترنس پوزیشن  
----> اینا کافی نیستند و امن نیستند

# Taxonomy of Cryptography

## □ Symmetric Key

- Same key for encryption and decryption
- Modern types: Stream ciphers, Block ciphers

## □ Public Key (or “asymmetric” crypto)

- Two keys, one for encryption (public), and one for decryption (private)
- And digital signatures — nothing comparable in symmetric key crypto

## □ Hash algorithms

- Can be viewed as “one way” crypto

تاكسونومي رمزنگاري

□ کلید متقارن

0 کلید يکسان برای رمزگذاری و رمزگشایی --> هر دو طرف از یک کلید دارند استفاده میکنند

0 انواع مدرن: رمزهای جریانی، رمزهای بلوکی --> اینا میشن شکل های مدرنش که اسم انگلیسیشو ببین

□ کلید عمومی (یا رمزنگاری نامتقارن) --> هر کسی دوتا کلید داره یک عمومی و یک خصوصی و هر کسی که بخواهد به من پیام بده پیامشو با کلید عمومی اون طرف رمز می کنه بعد پیامی که رمز شده با کلید عمومی مثلًا من می تونم با کلید خصوصی خودم این پیام رو باز کنم

نکته: یک امكان خیلی خوبی به وجود اومد توی این کلید عمومی و اون این بود که ما پیامی رو با کلید خصوصی خودمون رمز کنیم و بعد برای هر کسی بفرستیم او نا می تونن با کلید عمومی من مثلًا پیام رو باز کنند--> مزیتش چیه؟ نشان دهنده اینه که ما فقط اون پیام رو رستادیم پس بحث همون امضا دیجیتاله میشه پس برای اینکه که ما این پیام رو فقط تولید کردیم --> امضا دیجیتال

0 دو کلید، یکی برای رمزگذاری (عمومی) و دیگری برای رمزگشایی (خصوصی) 0 و امضاهای دیجیتال - هیچ چیز در رمز ارزهای کلید متقارن قابل مقایسه نیست

□ الگوریتم های هش --> این رمزنگاری یک طرفه است

0 را می توان به عنوان رمزنگاری "یک طرفه" مشاهده کرد

# Taxonomy of Cryptanalysis

- From perspective of info available to Trudy...
  - Ciphertext only — Trudy's worst case scenario
  - Known plaintext
  - Chosen plaintext
    - "Lunchtime attack"
    - Some protocols will encrypt chosen data
  - Adaptively chosen plaintext
  - Related key بنی به جای اینکه کلید رو بدست بیاریم سعی کنیم چیزی وابستشو به دست بیاریم
  - Forward search (public key crypto)
  - And others...

- رمز شکنی:

بدترین حالت برای trudy اینه که فقط ciphertext رو داشته باشه  
حالت بعدی اینه که اون plaintext رو میدونه و جاهایی از متن مثلًا این بوده و نظریش توی  
ciphertext رو می خواد چیز کنه و این یک امتیازه برای ciphertext

حالت بعدی اینه که خودش بتونه یک سری plaintext های خاصی رو به اون سیستم رمز بده و نتیجشو  
ببینه این خیلی برایش بهتره نسبت به حالت قبل --> به این lunchtime attack هم میگن  
ادaptively chosen plaintext یعنی مهاجم یک plaintext رو ارسال کرده و یک جوابی گرفته  
مبتنی بر ciphertext بعد مبتنی بر این جوابی که دریافت کرده یک plaintext دیگه میده با این کار یه  
سری چیزای خاص دیگه رو میفهمه

: همه کلید عمومی رو دارند فرض میکنیم فضای پیام که الیس و باب دارند و اسه هم  
پیام می فرستند یک فضای محدودی است مثلًا فلان ساعت حمله کنیم بعد الیس می گه بله یا خیر--> الیس  
وقتی می خواد اینو به باب بفرسته میاد با کلید عمومی باب اینو رمز می کنه و این کارو trudy هم می  
تونه انجام بده و بعد میاد اینو با پیام رمزشده رو با پیام الیس مقایسه میکنه که بفهمه الیس چی میگه  
نکته= همون طوری که فضای کلید نباید محدود باشه که بتونه جستجوی کامل انجام بده نکته بعدی هم اینه  
که برای رمز های کلید عمومیمون فضای پیام هم نباید محدود باشه  
ادامه صفحه بعدی...

# Chapter 3: Symmetric Key Crypto

The chief forms of beauty are order and symmetry...  
— Aristotle

“You boil it in sawdust: you salt it in glue:  
    You condense it with locusts and tape:  
Still keeping one principal object in view —  
    To preserve its symmetrical shape.”  
— Lewis Carroll, *The Hunting of the Snark*

ادامه صفحه قبل:

چوری این فضای پیام رو درست کنیم که محدود نباشد؟

می دونه فضای اون پیام چیه و مثلًا میدونه جواب الیس فقط yes or no است  
اینجا فضای پیام خیلی خیلی کوچیک است

برای رفع این مشکل میایم یه چیز رندوم می ذاریم کنار اون جواب اصلی می ذاره الیس و  
واسه باب می فرسته اصطلاحا میگن pad اش میکنه و باب میدونه اون چیز رندومه و جواب  
اصلی مثلًا yes  
با این کار فضای پیام بزرگ میشه و trudy نمی تونه کل این فضا رو پیدا کنه

# Symmetric Key Crypto

- Stream cipher — generalize one-time pad
  - Except that key is relatively short
  - Key is stretched into a long **keystream**
  - Keystream is used just like a one-time pad
- Block cipher — generalized codebook
  - Block cipher **key** determines a codebook
  - Each key yields a different codebook
  - Employs both “confusion” and “diffusion”

- سیستم کلید های متقارن ک دو نوع است:

-1 **stream cipher** <-- اینا میخواین ادای otp رو در بیارن

توی این حالت کلید رندوم است و کلید رو بزرگ کنند و از اون keystream می سازند و  
این keystream نقش اون one time pad رو بازی میکنه

توی این حالت **stream cipher** ما امن نیستند مثل otp ولی به جاش **practical** هستند ینی  
به خیلی راحت ازش استفاده کنیم

توی این حالت شبیه otp همون **confusion** رو داریم  
و

-2 **block cipher** <-- اینا هم ادای codebook ها رو درمیارن

توی این حالت هر کلید برای ما codebook متمایز می سازه ینی هر بار که کلید رو عوض  
کنیم codebook ما عوض میشه

توی این حالت هم **diffusion** و هم **confusion** رو داریم

رمزهای جریان

# Stream Ciphers



---

# Stream Ciphers

- Once upon a time, not so very long ago... stream ciphers were the king of crypto
  - Today, not as popular as block ciphers
- We'll discuss two stream ciphers:
- A5/1
  - Based on shift registers
  - Used in GSM mobile phone system
- RC4
  - Based on a changing lookup table
  - Used many places

یه روزگاری پادشاهی می کردند و خیلی از شون استفاده میشه ولی بعدا اینا مردن دو تا **stream cipher** معرفی میکنیم:

1- A5/1 مبتنی بر شیفت رجیستر است و برای GSM موبایل استفاده میشه و یک روش سخت افزاری است

2- RC4 هنوز داره توی پایه بعضی از پروتکل ها استفاده میشه و خیلی از روش های جدید مبتنی بر این روشه پس خیلی مهمه

این روش هم مبتنی بر **lookup table** است یعنی یک جدول جستجو داریم و ازش کلید رو بدهست میاریم و پیاده سازی نرم افزاری داره

# A5/1: Shift Registers

## □ A5/1 uses 3 shift registers

- X: 19 bits ( $x_0, x_1, x_2, \dots, x_{18}$ )
- Y: 22 bits ( $y_0, y_1, y_2, \dots, y_{21}$ )
- Z: 23 bits ( $z_0, z_1, z_2, \dots, z_{22}$ )

سه تا رجیستر داریم که جمع سه رجیستر میشه 64 بیت  
که این 64 بیت به صورت اولیه با یک کلید اولیه 64 بیتی پر میشه  
پس یکی کلید 64 بیتی داریم که به صورت اولیه 3 تا رجیستر رو پر میکنه  
هر چقدر دلمون بخواهد می تونه بیت تولید کنه

---

# A5/1: Keystream

- At each iteration:  $m = \text{maj}(x_8, y_{10}, z_{10})$ 
  - Examples:  $\text{maj}(0,1,0) = 0$  and  $\text{maj}(1,1,0) = 1$
- If  $x_8 = m$  then X steps
  - $t = x_{13} \oplus x_{16} \oplus x_{17} \oplus x_{18}$
  - $x_i = x_{i-1}$  for  $i = 18, 17, \dots, 1$  and  $x_0 = t$
- If  $y_{10} = m$  then Y steps
  - $t = y_{20} \oplus y_{21}$
  - $y_i = y_{i-1}$  for  $i = 21, 20, \dots, 1$  and  $y_0 = t$
- If  $z_{10} = m$  then Z steps
  - $t = z_7 \oplus z_{20} \oplus z_{21} \oplus z_{22}$
  - $z_i = z_{i-1}$  for  $i = 22, 21, \dots, 1$  and  $z_0 = t$
- Keystream bit is  $x_{18} \oplus y_{21} \oplus z_{22}$

سه تا بیت میگیریم  $z10$ ,  $y10$ ,  $x8$  بعد این سه تا بیت رو با تابع اکثریت maj میاد پیدا میکنه اکثریتش چیه

بعد این برای ما m رو می سازه

اگر m با اون بیت های قرمز رنگ یکی بود رجیستر شیفت پیدا میکنه ینی یکی می ره جلو ینی step میکنه یه گام بر میداره

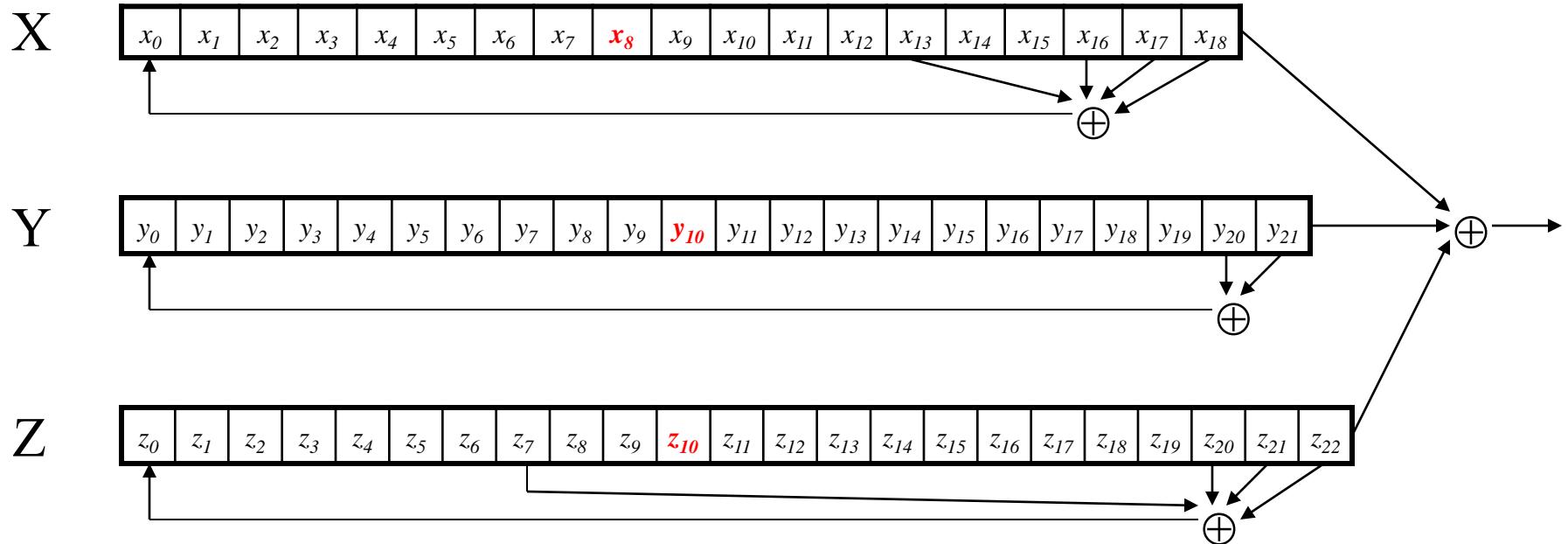
حالا اگر اون بیت های قرمز رنگ با m برابر نبودند رجیستر شیفت پیدا نمیکنه  $z0$ ,  $y0$ ,  $x0$  از طریق رابطه های رو برو به دست میاد ولی بقیه بیت ها از طریق همون شیفت هایی که دادیم پر شدند

مثلا توی مثل چند صفحه دیگه y شیفت پیدا نمیکنه چون با m یکی نیست پس y0 تغییر نمیکنه چون step رخ نداده

حالا می خوایم یه بیت کلید بسازیم که اینو میاریم از xor آخرین بیت های سه رجیستر به دست میاریم مثلا توی مثل صفحه 47 بیت کلید ما میشه 1 --> با این کار یه دونه بیت کلید تولید شد

با این روش می تونیم هر چقدر دلمون خواست بیت کلید تولید کنیم

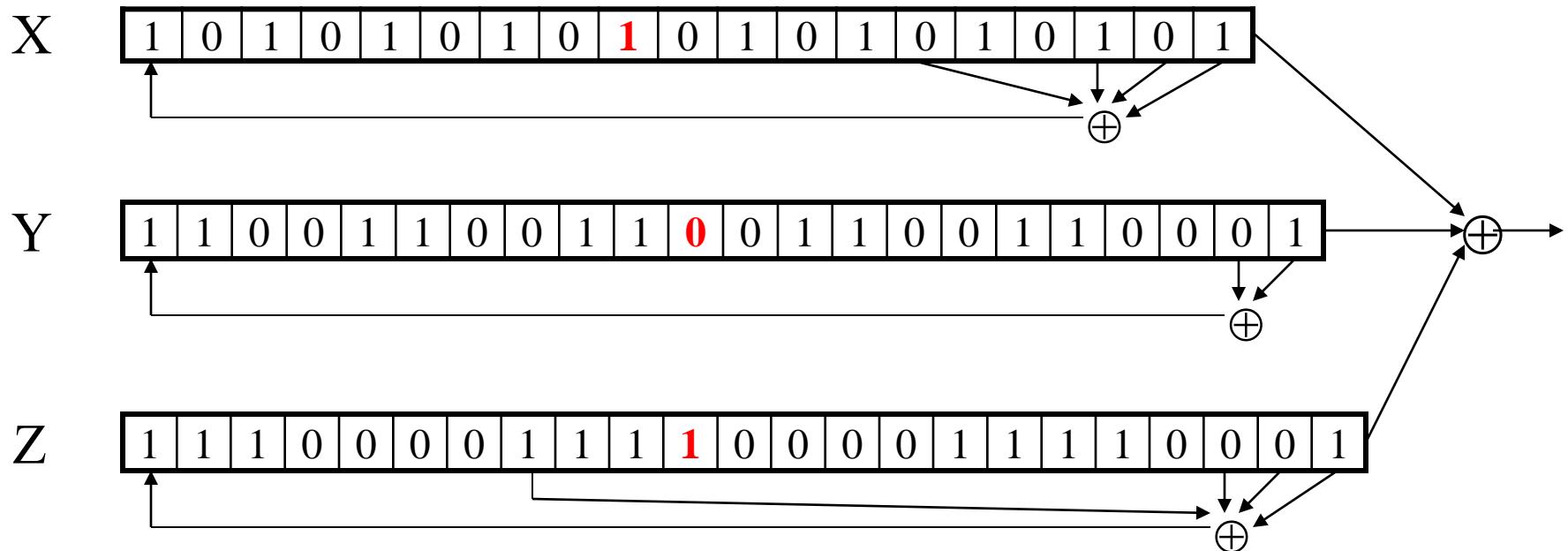
# A5/1



- Each variable here is a single bit
- Key is used as **initial fill** of registers
- Each register steps (or not) based on  $\text{maj}(x_8, y_{10}, z_{10})$
- **Keystream bit is XOR of rightmost bits of registers**

---

## A5/1



- In this example,  $m = \text{maj}(x_8, y_{10}, z_{10}) = \text{maj}(\mathbf{1,0,1}) = \mathbf{1}$
- Register X steps, Y does not step, and Z steps
- Keystream bit is XOR of right bits of registers
- Here, keystream bit will be  $0 \oplus 1 \oplus 0 = 1$

- این سه تا شیفت رجیستر داره که روی هم 64 بیت داره  
به صورت اولیه این 64 بیت اولیه رو کلید پر میکرد  
نکته: تابع **maj** یا اکثریت داریم که از سه تا بیت استفاده میکنه اگر این  $m$  با اون بیت های  
خاص که اینجا قرمز رنگ اند برابر بود می اوMD **step** میکرد یعنی یه دونه می رفت جلو در  
غیر اینصورت جلو نمی رفت

# Shift Register Crypto

- Shift register crypto efficient in hardware
- Often, slow if implemented in software
- In the past, very, very popular
- Today, more is done in software due to fast processors
- Shift register crypto still used some
  - Especially in resource-constrained devices

شیفت رجیسترها برای سخت افزاری مناسب است چون اگر توی نرم افزار بخوایم بريم اینکه  
اینا با هم برابر هستند یا نه یه گام زمانی توی نرم افزار از ما می گیره که توی سخت افزار  
اینا به صورت موازی استفاده میشه  
امروزه پیاده سازی ها بیشتر نرم افزاری هستند

# RC4

- A self-modifying lookup table
- Table always contains a permutation of the byte values  $0, 1, \dots, 255$
- Initialize the permutation using key
- At each step, RC4 does the following
  - Swaps elements in current lookup table
  - Selects a keystream byte from table
- Each step of RC4 produces a **byte**
  - Efficient in software
- Each step of A5/1 produces only a bit
  - Efficient in hardware

- اصل و اساس این روش یک **lookup table** است  
این **table** ما 256 تا خونه داره و توی هر کدوم از این خونه ها یه عددی از 0 تا 255 می تونه قرار بگیره  
پر کردن اولیه ما با کلید است  
توی هر گام RC4:  
میاد این عناصر رو جابه جا میکنه و میخواهد این حالت خوداصلاحی رو داشته باشه یعنی خودش خودشو تغییر میده که بایت بعدی که تولید میکنه این دیگه تکراری نشه و گام بعدی هم این است که یه دونه بایت از این **table** انتخاب میکنه که اون بایت مارو می سازه **keystream**  
پیاده سازیش در نرم افزار است

# RC4 Initialization

- ❑  $S[]$  is permutation of  $0, 1, \dots, 255$
- ❑  $\text{key}[]$  contains  $N$  bytes of key

```
for i = 0 to 255
    S[i] = i
    K[i] = key[i % N]
next i
j = 0
for i = 0 to 255
    j = (j + S[i] + K[i]) mod 256
    swap(S[i], S[j])
next i
i = j = 0
```

- مقدار دهی اولیه این روش:  
کلید ما از  $n$  بایت تشکیل شده  
عملیات باقی مانده است  $\text{mod}$

# RC4 Keystream

- At each step, swap elements in table and select keystream byte

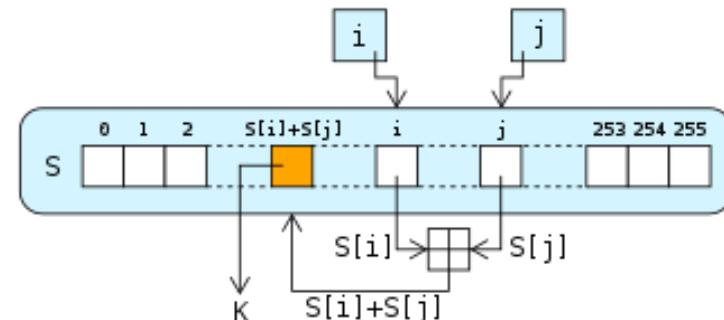
$$i = (i + 1) \bmod 256$$

$$j = (j + S[i]) \bmod 256$$

swap( $S[i]$ ,  $S[j]$ )

$$t = (S[i] + S[j]) \bmod 256$$

keystreamByte =  $S[t]$



- Use keystream bytes like a one-time pad
- Note:** first 256 bytes should be discarded
  - Otherwise, related key attack exists

- اینجا یه بایت keystream می خواد بسازه  
[s[t] به عنوان بایت برمیداره  
256 بایت اول رو دور می ریزیم چرا؟  
اگر این 256 بایت اول رو دور نریزیم حمله related key رخ می ده

# Stream Ciphers

- Stream ciphers were popular in the past
  - Efficient in hardware (shift register types)
  - Speed needed to keep up with voice data, etc.
  - Today, processors are fast, so software-based crypto is usually more than fast enough
- Future of stream ciphers?
  - Shamir declared “the death of stream ciphers”
  - May be greatly exaggerated...

- یه زمانی خیلی معروف بودند به خصوص اینکه همه فکر میکردند داره otp رو پیاده می کنه
- رفته رفته این روش کنار گذاشته شد
- گفت این stream cipher ها مردن shamir -

بلاک رمزها

# Block Ciphers



---

# (Iterated) Block Cipher

- Plaintext and ciphertext consist of fixed-sized blocks
- Ciphertext obtained from plaintext by iterating a **round function**
- Input to round function consists of **key** and **output** of previous round
- Usually implemented in software

-  
توى اين روش يه بلاکى از **ciphertext** رو ميگيره و يه بلاکى از **plaintext** رو به ما ميده  
و اين سايىزش معمولاً ثابت است  
شبيه **codebook** هستند

عملياتى كه روی **plaintext** انجام مىشه توى چندتا دور انجام مىشه پس توش تكرار دارييم كه  
داره اينجا هي تكرار مىشه --> توى كاغذ شكلشو كشيدم  
مون وروديش شامل کلید هم مىشه  
اين روش روی نرم افزار هم کار ميکند

# Feistel Cipher: Encryption

- Feistel cipher is a type of block cipher
  - Not a specific block cipher
- Split plaintext block into left and right halves:  $P = (L_0, R_0)$
- For each round  $i = 1, 2, \dots, n$ , compute

$$L_i = R_{i-1}$$

$$R_i = L_{i-1} \oplus F(R_{i-1}, K_i)$$

where  $F$  is round function and  $K_i$  is subkey

- Ciphertext:  $C = (L_n, R_n)$

یک الگوریتم رمز خاص نیست بلکه یک نوعی از بلاک سایفر است  
کارش اینه که:

اول اون **plaintext** رو به دو نیمه تقسیم میکنه  
توی هر راندی که این راند از یک تا  $n$  میاد سمت چپ جدید میشه سمت راست قبلی و سمت  
راست جدید میشه سمت چپ قبلی که **xor** میشه با تابع  $F$   
توی کاغذ شکلشو کشیدم --> جلسه 4 میشه  
نکته:  $k_i$  کلید نیست بلکه یک زیرکلید است که ساخته میشه از کلیدمون  
بعد از  $n$  تا راند ما **ciphertext** رو داریم

# Feistel Cipher: Decryption

- Start with ciphertext  $C = (L_n, R_n)$
- For each round  $i = n, n-1, \dots, 1$ , compute

$$R_{i-1} = L_i$$

$$L_{i-1} = R_i \oplus F(R_{i-1}, K_i)$$

where  $F$  is round function and  $K_i$  is subkey

- Plaintext:  $P = (L_0, R_0)$
- Decryption works for any function  $F$ 
  - But only *secure* for certain functions  $F$

- حالا حالت رمزگشایی:

این تابع  $F$  رو اونظرف که داره اینو رمزگشایی میکنه هم داره  
هم می تونه تولید کنه **subkey**  
این برای هر تابع  $F$  کار می کند  
هر  $F$  می تونیم اینجا بذاریم فقط مهم اینه که اون  $F$  امن باشه

# Data Encryption Standard

- **DES** developed in 1970's
- Based on IBM's Lucifer cipher
- DES was U.S. government standard
- Development of DES was controversial
  - NSA secretly involved
  - Design process was secret
  - Key length reduced from 128 to 56 bits
  - Subtle changes to Lucifer algorithm

- DES توی دهه 70 ظاهر شد

چرا اومدن سراغ این که یک استانداردی داشته باشیم؟

چون کامپیوتر ها اومدن، دیگه صرفا دولت ها و ارتش ها نبودن که بحث رمزنگاری رو میخواستند ویه عالمه مشتری داشتیم که کلی داده با کامپیوتر تولید کردند و دوست داشتند این هارو رمز کنند

IBM اوMD فراخوانی داد که هر کسی الگوریتم رمزی داره بیاد ارائه بکنه یکی از پروپوزالها lucifer بود که بهترینش هم بود منتها خود IBM خودش رمزنگار نداشت که بیاد تحلیل بکنه اینارو پس از NSA کمک خواست

و NSA هم lusifer هم انتخاب کرد

IBM به صورت مخفیانه از NSA کمک خواست

NSA یکم این lusifer رو دست کاری کرد مثلًا طولش رو از 128 کرد 56 با این کار NSA خواست ضعیفش بکنه که بعدا خودش بتونه اونو بشکونه خیلی ها این وسط مشکوک شدند که NSA ک اومده کلید رو کمش کرده به این خاطر بوده که بعدا اینو بشکونه

یه سری تغییرات دیگه هم NSA داد

# DES Numerology

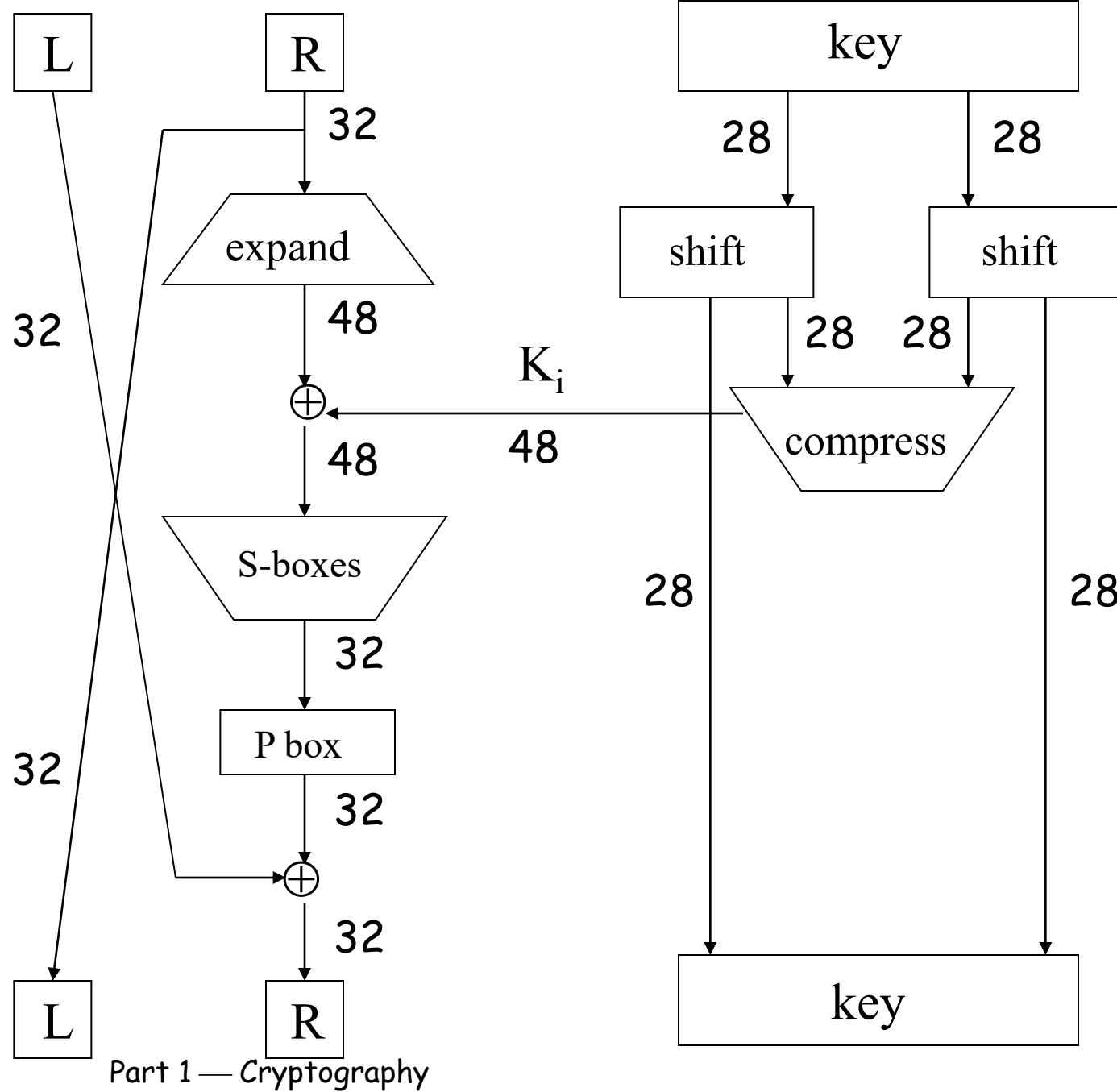
- DES is a Feistel cipher with...
  - 64 bit block length
  - 56 bit key length
  - 16 rounds
  - 48 bits of key used each round (subkey)
- Round function is simple (for block cipher)
- Security depends heavily on “S-boxes”
  - Each S-box maps 6 bits to 4 bits

-  
یک رمز des feistel است به علت این که هم یکی از طراحای لوسیفر بود  
توی هر دور 48 بیت است

هم خیلی ساده است  
امنیت des s-box ها وابسته است

عملیات هایی که داره روی des انجام میشه همه خطی اند تنها جایی که غیر خطی است همین  
است s-box

# One Round of DES



نکته: شیفت های ما چرخشی هستند

رو می سازند round function دارند با هم expand , s-box , pbox

# DES Expansion “Permutation”

## □ Input 32 bits

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

## □ Output 48 bits

31	0	1	2	3	4	3	4	5	6	7	8
7	8	9	10	11	12	11	12	13	14	15	16
15	16	17	18	19	20	19	20	21	22	23	24
23	24	25	26	27	28	27	28	29	30	31	0

گسترش DES "جایگشت":  
expand

سمت راست 32 بیت دارد ینی 32 بیت ورودی داره و 48 بیت خروجی می خواد ینی 16 بیت تکرار شده به صورت پخش

این نزدیک به deffusion است ولی منتها ترتیب بهم نخورده ینی این معنا نیست فقط نزدیک بهش است

این تکرار شدن ثابت است

این 48 بیت رندوم نیست --> این تکرار همیشه به همین صورت است

# DES S-box

- 8 “substitution boxes” or S-boxes
- Each S-box maps 6 bits to 4 bits
- Here is S-box number 1

input bits (0,5)



input bits (1,2,3,4)

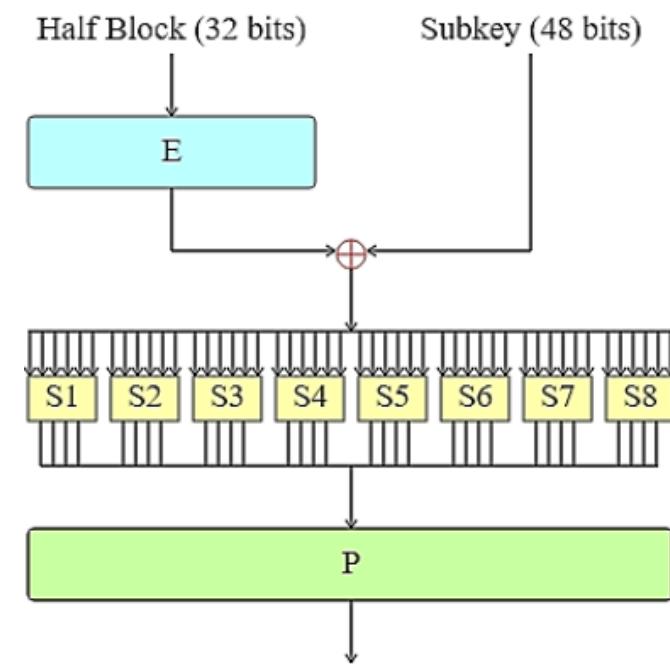
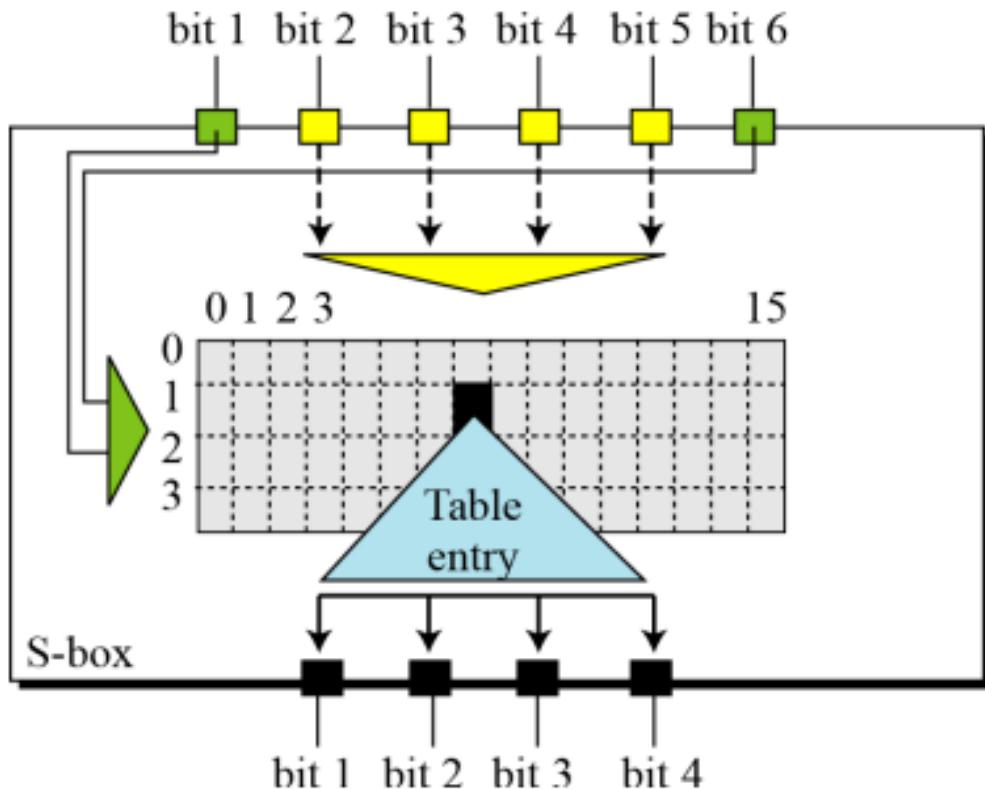
	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
-----																
00	1110	0100	1101	0001	0010	1111	1011	1000	0011	1010	0110	1100	0101	1001	0000	0111
01	0000	1111	0111	0100	1110	0010	1101	0001	1010	0110	1100	1011	1001	0101	0011	1000
10	0100	0001	1110	1000	1101	0110	0010	1011	1111	1100	1001	0111	0011	1010	0101	0000
11	1111	1100	1000	0010	0100	1001	0001	0111	0101	1011	0011	1110	1010	0000	0110	1101

- 8 تا s-box داریم --> 48 بیت وارد این 8 تا میشه یعنی به هر s-box شش بیت می رسه با اون 6 بیت میاد:  
ما از b0 تا b5 داریم میاد از بیت 0 و 5 میاد اینو جستجو میکنه یعنی سطر رو پیدا میکنه  
با اون 4 تا بیت دیگه میاد ستون رو پیدا میکنه  
با این سطر و ستون میاد یک عدد 4 بیتی به دست میاره پس خروجی s-box ما 32 بیت  
است یعنی میشه  $8^{4*8}$   
مثالش توی برگه زده شده

این 1 s-box ما است یعنی ما 8 تا جدول این شکلی داریم  
نکته: این جدول ها رو مردم نمی دونستند و بحث ما اینجا خطی نیست  
برگشت s-box اصلا راحت نیست چون توی هر کدام از ردیف ها از 0 تا 15 داریم  
ما s-box confusion است

نکته: 8 تا s-box که داریم fix هستند یعنی مقدار داخل خونه ها ثابت است با اون کلید ما میایم  
سطر و ستون رو پیدا میکنیم که کدامش رو برداریم یعنی اون کلید تعیین کننده است که سطر و  
ستون ما چی هست که بعد اون مقدار رو برداریم

# DES S-box



-----

# DES P-box

## □ Input 32 bits

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

## □ Output 32 bits

15	6	19	20	28	11	27	16	0	14	22	25	4	17	30	9
1	7	23	13	31	26	2	8	18	12	29	5	21	10	3	24

با p box می خواه deffusion رو بسازه  
ینی ورودی 32 بیتی رو میاد بهم می ریزه  
اینجا هم رندوم نداریم  
امنیت ما به s-box وصل است بقیه مراحل بیشتر داره پیچیدگی رو اضافه میکنه

# DES Subkey

- 56 bit DES key, numbered  $0, 1, 2, \dots, 55$
- Left half key bits, LK

49	42	35	28	21	14	7
0	50	43	36	29	22	15
8	1	51	44	37	30	23
16	9	2	52	45	38	31

- Right half key bits, RK

55	48	41	34	27	20	13
6	54	47	40	33	26	19
12	5	53	46	39	32	25
18	11	4	24	17	10	3

## 56 بیت کلید وارد میشه

بعد نصفشو توی یک شیفت می ریزه و 28 تا دیگر رو هم توی یک شیفت دیگه نکته: این 28 تا مرتب نمیاد این شیفت رو پرش کنه مثالو نگاه کن متوجه میشی

# DES Subkey

- For rounds  $i=1, 2, \dots, 16$ 
  - Let  $LK = (LK \text{ circular shift left by } r_i)$
  - Let  $RK = (RK \text{ circular shift left by } r_i)$
  - Left half of subkey  $K_i$  is of LK bits

13	16	10	23	0	4	2	27	14	5	20	9
22	18	11	3	25	7	15	6	26	19	12	1

- Right half of subkey  $K_i$  is RK bits

12	23	2	8	18	26	1	11	22	16	4	19
15	20	10	27	5	24	17	13	21	7	0	3

-  
des ما 16 تا راند داشت  
شیفت ها به صورت چرخشی است

# DES Subkey

- For rounds 1, 2, 9 and 16 the shift  $r_i$  is 1, and in all other rounds  $r_i$  is 2
- Bits 8,17,21,24 of LK omitted each round
- Bits 6,9,14,25 of RK omitted each round
- **Compression permutation** yields 48 bit subkey  $K_i$  from 56 bits of LK and RK
- **Key schedule** generates subkey

-  
برای راند 1 و 2 و 9 و 16 میاد یه دونه شیفت میده  
و برای بقیه راند ها 2 تا دونه شیفت میده  
و بعضی از بیت ها هم دور می ریزه که نوشته شده توی اسلاید  
در اخر از اون 56 تا که وارد شد 48 تا خارج میشه

# DES Last Word (Almost)

- An initial permutation before round 1
- Halves are swapped after last round
- A final permutation (inverse of initial perm) applied to  $(R_{16}, L_{16})$
- None of this serves any security purpose

قبل از راند اول ما یک initial permutation داریم ینی اون 64 تا بیتی که اول گرفت یه بوری اولش زده

بعد نیمه ها رو بعد از راند اخر میاد بر عکسشون میکنه  
نکته: هیچ کدام از اینا هیچ هدف امنیتی و اسه ما نداره

# Security of DES

- Security depends heavily on S-boxes
  - Everything else in DES is linear
- 45+ years of analysis revealed no back door
- Attacks? Essentially exhaustive key search
- Inescapable conclusions
  - Designers of DES knew what they were doing
  - Designers of DES were way ahead of their time (at least wrt certain cryptanalytic techniques)

## امنیت DES

- امنیت به شدت به جعبه های S بستگی دارد
- هر چیز دیگری در DES خطی است
- بیش از 45 سال تجزیه و تحلیل هیچ درب پشتی را نشان نداد
- حملات؟ جستجوی کلید اساساً جامع --> از نظر امنیت خوبه
- نتیجه گیری های اجتناب ناپذیر
- طراحان DES می دانستند که چه کار می کنند
- طراحان DES بسیار جلوتر از زمان خود بودند (حداقل در برخی از تکنیک های رمزنگاری خاص)

اوایل دهه 90 معلوم شد توی این s-box چی می گذره پس بعد تحلیل des راحت تر شد  
توی جزو هم ببین!!! ص 8  
سال 1997 مشخص شد امنیت des قابل اتکا نیست

# Block Cipher Notation

- $P$  = plaintext block
- $C$  = ciphertext block
- Encrypt  $P$  with key  $K$  to get ciphertext  $C$ 
  - $C = E(P, K)$
- Decrypt  $C$  with key  $K$  to get plaintext  $P$ 
  - $P = D(C, K)$
- Note:  $P = D(E(P, K), K)$  and  $C = E(D(C, K), K)$ 
  - But  $P \neq D(E(P, K_1), K_2)$  and  $C \neq E(D(C, K_1), K_2)$  when  $K_1 \neq K_2$

E بىنى مياد يك **plaintext** رو با كلید رمز مى كنه  
D هم مياد رمز گشايى ميكنه يك **ciphertext** رو با كلید

# Triple DES

- 56 bit DES key is too small
  - Exhaustive key search is feasible
- But DES was everywhere, so what to do?
- **Triple DES** or **3DES** (112 bit key)
  - $C = E(D(E(P, K_1), K_2), K_1)$
  - $P = D(E(D(C, K_1), K_2), K_1)$
- Why Encrypt-Decrypt-Encrypt with 2 keys?
  - Backward compatible:  $E(D(E(P, K), K), K) = E(P, K)$
  - And 112 is a lot of bits

فضای کلید des کم بود ولی توی دنیا پخش شده بود ولی به Des حمله شده بود و همون موقع همه نمی تونستن کنار بذارنش پس یک ایده این بود که:  
1- بیایین از 3 تا Des پشت سر هم استفاده کنند --> سه تا Des کلیدش میشه  $3 \times 56$  ولی از دو تا کلید استفاده می کرد و اسه همین شد 112 --> که این 112 بیت کافیه نکته: 2 به توان 111 باید جستجو می کردیم برای جستجوی کامل چرا نمی گه EEE چرا مثلا می گه EDE ؟  
بعضی ها بودن که می خواستن با یک کلید کار کنند ینی دو طرف همچنان امکان کار کردن داشته باشند ینی اگه ما گذاشتیم EEE ینی سه تا K شد و اگه اون طرف می خواست از یک دونه des استفاده کنه با ما نمی تونست کار بکنه ینی دو طرف بتوان با یک کلید هم کار بکنند ینی مثلا یک طرف des داره و طرف دیگه یعنی داره و اسه اینه که از DED یا EDE استفاده می کنیم triple des

# 3DES

- Why not  $C = E(E(P, K), K)$  instead?
  - Trick question — still just 56 bit key
- Why not  $C = E(E(P, K_1), K_2)$  instead?
- A (semi-practical) **known plaintext** attack
  - Pre-compute table of  $E(P, K_1)$  for every possible key  $K_1$  (resulting table has  $2^{56}$  entries)
  - Then for each possible  $K_2$  compute  $D(C, K_2)$  until a match in table is found
  - When match is found, have  $E(P, K_1) = D(C, K_2)$
  - Result gives us keys:  $C = E(E(P, K_1), K_2)$

---

# Advanced Encryption Standard

- Replacement for DES
- AES competition (late 90's)
  - NSA openly involved
  - Transparent selection process
  - Many strong algorithms proposed
  - Rijndael Algorithm ultimately selected  
(pronounced like "Rain Doll" or "Rhine Doll")
- Iterated block cipher (like DES)
- Not a Feistel cipher (unlike DES)

---

# The AES finalist candidate algorithms

- MARS
- RC6
- Rijndael
- Serpent
- Twofish

# Rijndael

- ❑ Rijmen (1970- )
- ❑ Daemen (1965- )
- ❑ Based on Galois Field
  - (out of scope)



# AES: Executive Summary

- Block size: 128 bits (others in Rijndael)
- Key length: 128, 192 or 256 bits  
(independent of block size in Rijndael)
- 10 to 14 rounds (depends on key length)
- Each round uses 4 functions (3 "layers")
  - ByteSub (nonlinear layer)
  - ShiftRow (linear mixing layer)
  - MixColumn (nonlinear layer)
  - AddRoundKey (key addition layer)

رقبتش اخر دهه 90 بود

مشابه des این هم بلاک سایفر است ولی feistel نیست

Dو تا بلژیکی طراحیش کردند AES

Rijndael که طراحی کردند مبانای ریاضیش کاملا مشخص بود

: AES

یک بلاک سایفر است

سایز مشخصی دارد

توی حالت استاندارد شد 128 بیت

کلیدمون می تونه 128 یا 192 یا 256 باشه بسته به اینکه چقدر می خواین امن باشه طول

کلید رو انتخاب می کنیم

هر چقدر طول کلید بیشتر باشه تعداد راندهامون هم فرق میکنه --> اگر کلید 128 بیت باشه

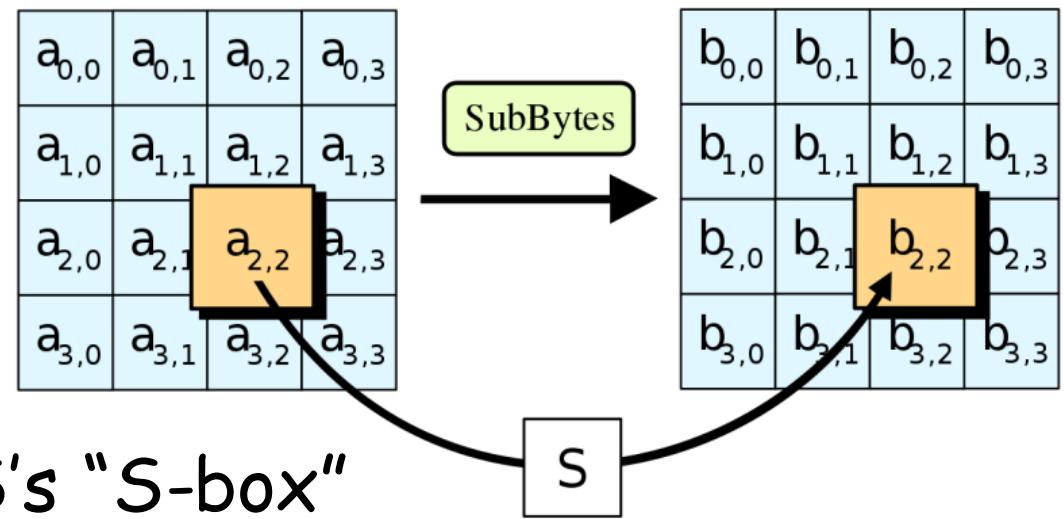
10 تا راند - 192 - 256 بیت باشه 12 راند - 14 راند داریم

توی هر کدوم از دور ها هم این 4 تا کار رو انجام میدیم غیر از راند صفرم و راند 10 ام

توی راند صفر هیچکدام از اینارو نداریم و فقط xor داریم و توی راند 10 ام اخرين مرحله رو نداریم

# AES ByteSub

- Treat 128 bit block as  $4 \times 4$  byte array



- ByteSub is AES's "S-box"
- Can be viewed as nonlinear (but invertible) composition of two math operations

-  
bytesub معادل s-box دس است و غیر خطی است  
128 تا رو تبدیل می کنه به 16 تا بایت و به صورت یک ماتریس  $4 \times 4$   
ستونی می چینه و میاد پرس میکنه  
توی این مرحله میاد --> هر کدام از این بایت ها رو جانشین می کنه با یک بایت دیگه  
یک بایت 2 تا هگز است --> با 4 بیت اول میاد سطر پیدا میکنه و با 4 بیت دوم ستون را  
اینجا یه دونه s-box داریم  
یک بایت 8 بیت است  
یک هگز 4 بیت است

# AES "S-box"

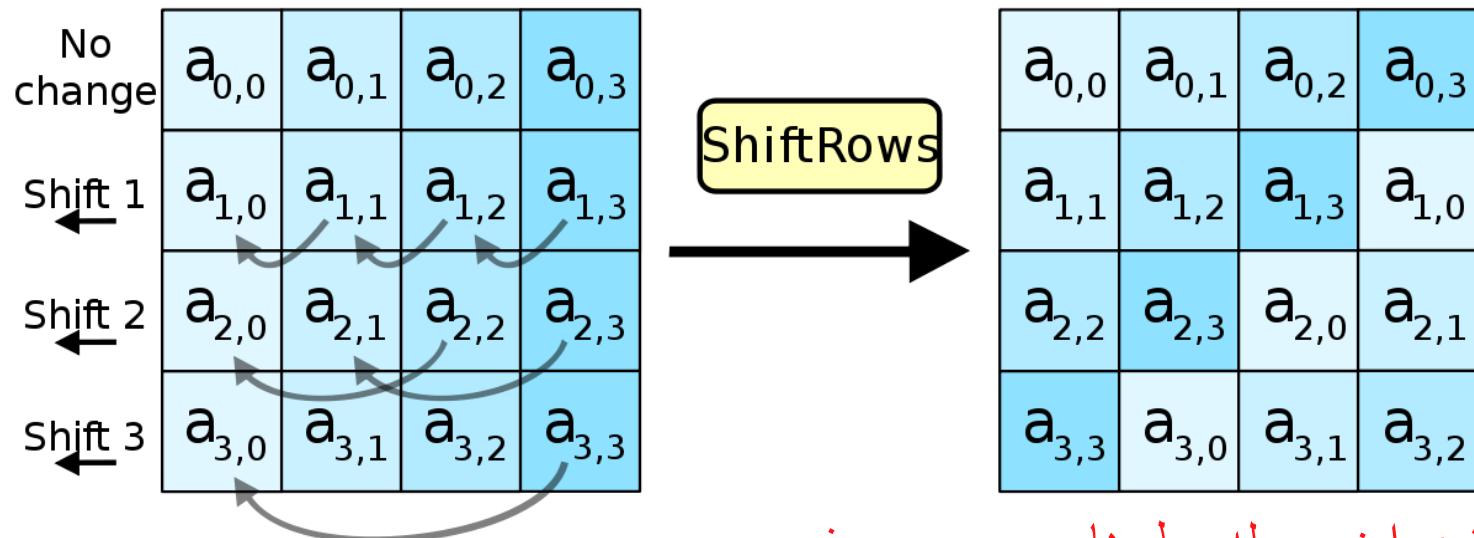
Last 4 bits of input

First 4  
bits of  
input

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

# AES ShiftRow

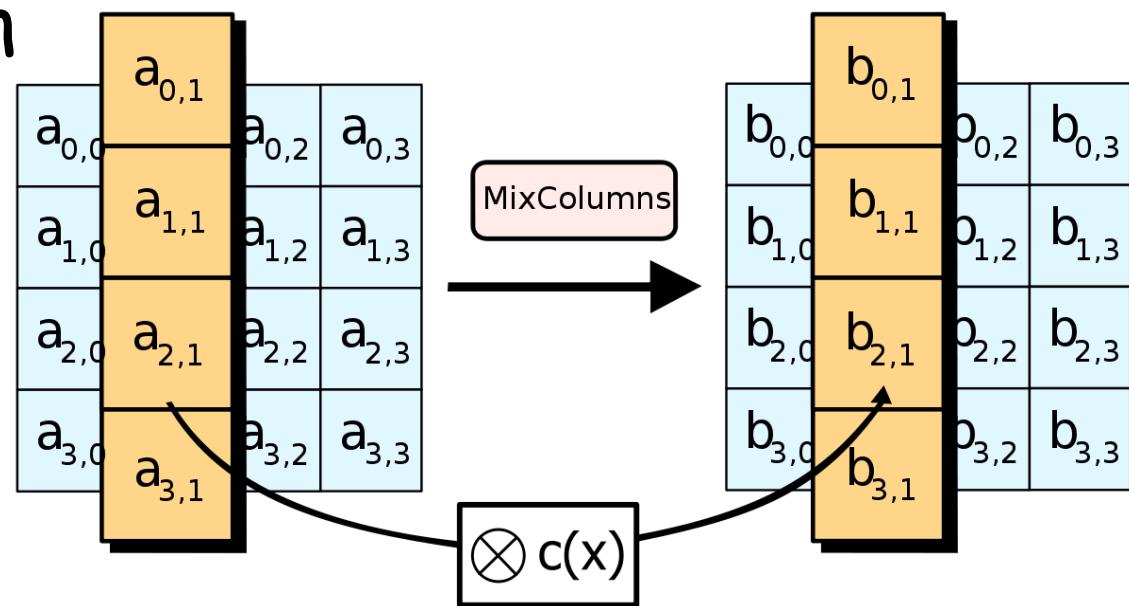
## □ Cyclic shift rows



توی این مرحله سطرها رو بهم می ریزه  
سطر اول رو تغییر نمی ده  
سطر دوم رو یکی شیفت می ده به چپ  
سطر سوم رو 2 تا شیفت میده  
سطر چهارم رو 3 تا شیفت میده  
نکته: این شیفت به چه و توی هر سطری تعداد شیفت ها متفاوته<sup>72</sup>

# AES MixColumn

- ❑ Invertible, linear operation applied to each column



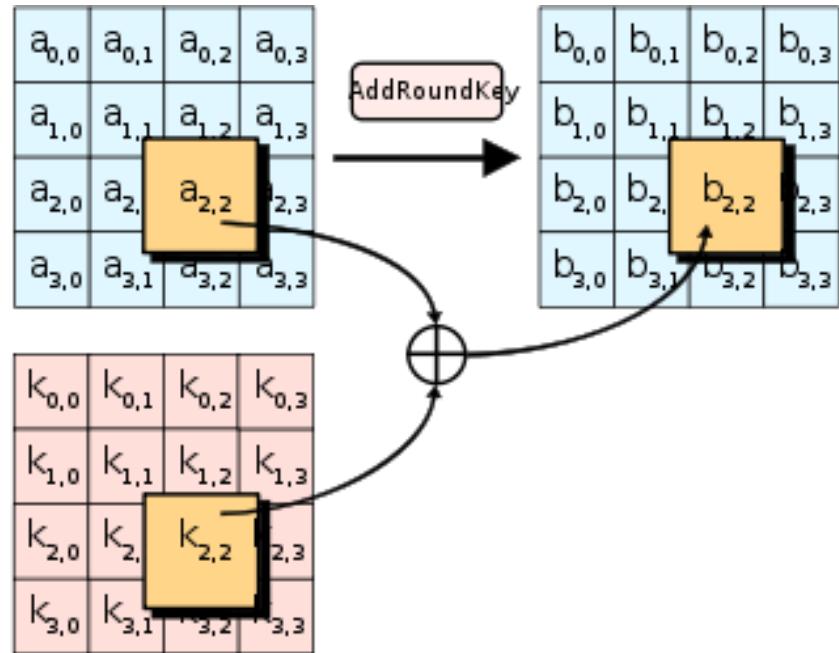
- ❑ Implemented as a (big) lookup table

توی این مرحله یک ماتریسی داره که هر کدام از ستون ها رو میاد در این ماتریسه ضرب میکنه که بررسه به ستون جدید این روش هم غیر خطی است

این به صورت **lookup table** قابل پیاده سازی است پس اونقدر بار محاسباتی نداره تا اینجا ما شیفت داشتیم و دو تا **lookup table**

# AES AddRoundKey

- ❑ XOR subkey with block



- ❑ RoundKey (subkey) determined by **key schedule algorithm**

یک بلاک کلید هم داره برای هر دور که بایت میاد اینا رو xor می کنه  
اینچوری AES انجام میشه

# AES Decryption

- To decrypt, process must be invertible
- Inverse of MixAddRoundKey is easy, since " $\oplus$ " is its own inverse
- MixColumn is invertible (inverse is also implemented as a lookup table)
- Inverse of ShiftRow is easy (cyclic shift the other direction)
- ByteSub is invertible (inverse is also implemented as a lookup table)

عملیات هایی که داره انجام میشه باید برگشت پذیر باشد  
اول از همه باید کلید رو xor کنه --> دو طرف کلید رو دارند پس subkey ها رو می تونن  
بسان

مون باید معکوس پذیر باشه تا از ستون خاصی که می خوایم رمزگشایش کنیم  
با اون ماتریس معکوس پذیر بتوانیم بررسیم به ستونی که قبلا داشتیم  
shiftRow هم به این صورته که اونجا که شیفت دادیم به سمت چپ اینجا شیفت میدیم به سمت  
راست

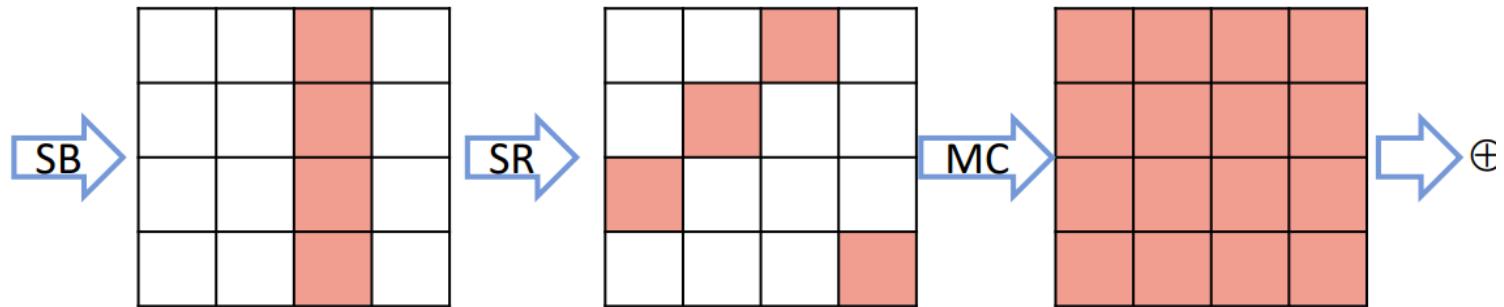
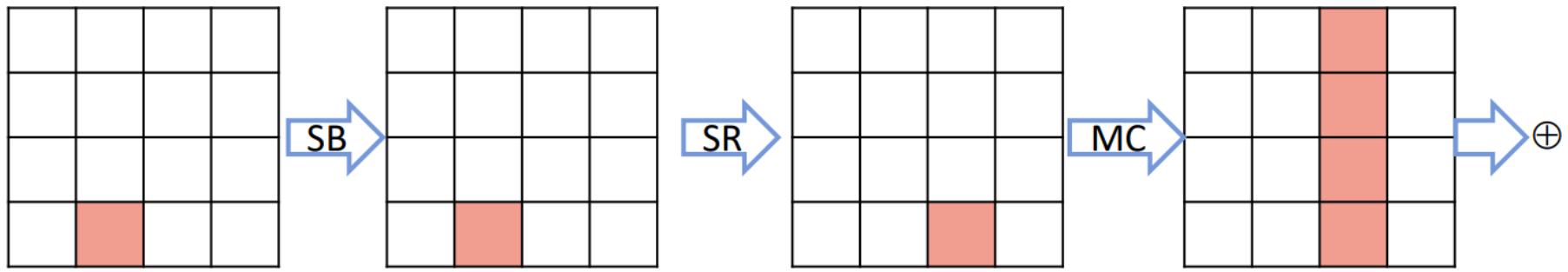
bytesub هم باید معکوس پذیر باشد  
این روش نسبت به des این پیچیدگی هارو دارد  
کل عملیات ها:

یا توی lookup table داریم جستجو میکنیم یا شیفت است یا Xor --> این سه تا است و این  
باعث میشه که سرعت ما خیلی خوب باشه برای AES  
این سرعت خوب نتیجیش این میشه که به صورت نرم افزاری خیلی راحت پیاده شد و چه الان  
هم به صورت سخت افزاری توی cpu پیاده سازی شده داریمش  
کلا این AES خیلی فراگیر است

# AES Animation :)

- [https://formaestudio.com/rijndaelinspector/archivos/Rijndael\\_Animation\\_v4\\_eng-html5.html](https://formaestudio.com/rijndaelinspector/archivos/Rijndael_Animation_v4_eng-html5.html)

# Single bit in two rounds!



# Block Cipher Modes

-----

# Multiple Blocks

- How to encrypt multiple blocks?
- Do we need a new key for each block?
  - If so, as impractical as a one-time pad!
- Encrypt each block independently?
- Is there any analog of codebook “additive”?
- How to handle partial blocks?
  - We won’t discuss this last issue here

-  
چگونه از این بلاک سایفر استفاده کنیم؟

# Modes of Operation

- Many modes — we discuss 3 most popular
- Electronic Codebook (**ECB**) mode
  - Encrypt each block independently
  - Most obvious approach, but a bad idea
- Cipher Block Chaining (**CBC**) mode
  - Chain the blocks together
  - More secure than ECB, virtually no extra work
- Counter Mode (**CTR**) mode
  - Block ciphers acts like a stream cipher
  - Popular for random access

چندتا مدل عملیاتی و اسه بلاک سایفر ها داریم که عبارتند از:

--> ECB هر بلاک رو ک بخوایم رمز کنیم به صورت مستقل این کارو میکنیم و این خیلی هم بدء --> و بعد نتیجه می گیریم که اصلا نباید از این مدل استفاده کنیم  
--> CBC این بلاک رو بهم زنجیر می کنه و وقتی زنجیر کرد پس مستقل از هم رمز نمیشن  
--> بار محاسباتی مثل ECB ولی نتیجه بهتری میگیریم  
--> CTR یک بلاک سایفری که می خود شبیه stream cipher ها باشه و شبیه اونا عمل می کنه و خیلی وقت ها که بخوایم کلید رندوم داشته باشیم از این استفاده میکنیم

# ECB Mode

هر کدام از اینا رو میایم مستقلا از هم رمز می کنیم یا رمزگشایی میکنیم

- Notation:  $C = E(P, K)$
- Given plaintext  $P_0, P_1, \dots, P_m, \dots$
- Most obvious way to use a block cipher:

## Encrypt

$$\begin{aligned}C_0 &= E(P_0, K) \\C_1 &= E(P_1, K) \\C_2 &= E(P_2, K) \dots\end{aligned}$$

## Decrypt

$$\begin{aligned}P_0 &= D(C_0, K) \\P_1 &= D(C_1, K) \\P_2 &= D(C_2, K) \dots\end{aligned}$$



- For fixed key  $K$ , this is "electronic" version of a codebook cipher (*without additive*)
  - With a different codebook for each key

- این هم مشابه codebook است ولی اینجا additive رو نداریم

# ECB Cut and Paste

- ❑ Suppose plaintext is

Alice digs Bob. Trudy digs Tom.

- ❑ Assuming 64-bit blocks and 8-bit ASCII:

$P_0 = \text{"Alice di"}$ ,  $P_1 = \text{"gs Bob. "}$ ,

$P_2 = \text{"Trudy di"}$ ,  $P_3 = \text{"gs Tom. "}$

- ❑ Ciphertext:  $C_0, C_1, C_2, C_3$

- ❑ Trudy cuts and pastes:  $C_0, C_3, C_2, C_1$

- ❑ Decrypts as

Alice digs Tom. Trudy digs Bob.

مثال است:

از ECB داریم استفاده میکنیم و 64 بیتی است و هر کاراکتر 8 بیت در نظر میگیریم پس توی هر بلاک ما می تونیم 8 تا کاراکتر داشته باشیم این وسط trudy ایستاده و از c0 تا c3 رسیده به دستش کاری که trudy میاد می کنه اینه که میاد ترتیبشو عوض میکنه و اون پیامی که به دست باب می رسه یه پیام کاملاً متفاوت است این از نوع حمله non plaintext است نقطه ضعف: اگر بدونیم که  $p_i$  با هم برابر است پس  $c_j$  هم با هم برابر میشه و این خیلی بده --> این کار به trudy یه سری اطلاعات اضافه میده اینجا trudy کلید رو نداره ولی plaintext رو می دونه و می دونه که ciphertext هم که داره پس میاد ترتیب بلاک های ciphertext رو بهم می ریزه و بعد می ده به باب

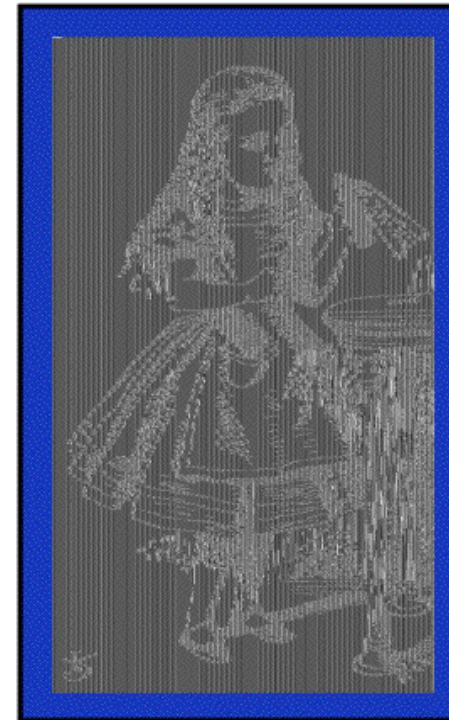
# ECB Weakness

- Suppose  $P_i = P_j$
- Then  $C_i = C_j$  and Trudy knows  $P_i = P_j$
- This gives Trudy some information, even if she does not know  $P_i$  or  $P_j$
- Trudy might know  $P_i$
- Is this a serious issue?

---

# Alice Hates ECB Mode

- ❑ Alice's uncompressed image, and ECB encrypted (TEA)



- ❑ Why does this happen?
- ❑ Same plaintext yields same ciphertext!

میدونه که  $pi$ ,  $pj$  با هم برابر هستند و این وسط هم نشسته --> این برابری چه امتیازی به **trudy** میده؟

این عکسو توی مد ECB رمز کرده و این جاهایی که تیکه های سفید که کنار هم هستند یعنی لاک من با بلاک کناری مقدارش یکی بوده خروجی رمزش هم یکی شده و **trudy** همون رمز شده رو نگا کنه انگار داره عکس plaintext رو می بینه و انگار چیزی به رمزش اضافه نشده

نکته: پس اگر پیاممون جاهایی از بلاکش با هم یکی باشه خروجیش هم یکی میشه و انگار پیام در او مده

# CBC Mode

- ❑ Blocks are “chained” together
- ❑ A random initialization vector, or IV, is required to initialize CBC mode
- ❑ IV is random, but not secret

## Encryption

$$C_0 = E(IV \oplus P_0, K),$$

$$C_1 = E(C_0 \oplus P_1, K),$$

$$C_2 = E(C_1 \oplus P_2, K), \dots$$

## Decryption

$$P_0 = IV \oplus D(C_0, K),$$

$$P_1 = C_0 \oplus D(C_1, K),$$

$$P_2 = C_1 \oplus D(C_2, K), \dots$$

- ❑ Analogous to classic codebook *with additive*

کلیت:

بلاک ها بهم **chained** شدند و ما اینجا یک **initial vector** یا IV هم داریم که معادل **additive** است

الیس **additive** رو برای باب به صورت فاش می فرسته اینجا هم همین طور است  
اینجا **additive** به بلاک اول اضافه میشه بعد خروجی بلاک اول یعنی C0 به عنوان **additive** بلاک بعدی استفاده میشه --> خروجی هر کدام از این بلاک ها برای بعدی مشه iv

اون طرف هم گیرنده IV رو به صورت فاش داره و بعد بیاد xor بکنه  
اینجا **encryptor** سه تا ورودی می خواهد  
سوال امتحانی:

اگر به جای دوتا کلید توی 3des از سه تا کلید استفاده کنیم ایا امنیت افزایش پیدا میکنه یا نه؟  
توی صفحه 9 شکلشو کشیدم

توضیح: نه چون 4 مین اول فیلم 6 بین وقتی که خوندی چون الان نفهمیدم؟://  
اینی CBC یعنی cipher ها رو بهم داریم **chained** می کنیم یعنی به همدیگر وابسته اند  
اگر اینجا یک بلاک از ciphertext مون خراب بشه یعنی مثلا trudy اون وسط نشسته و به  
جای اون بلاک خاص یه چیز دیگه میداره یعنی دست کاری میکنه اون بلاک رو مثلًا بلاک c1  
رو بر میداره و به جاش مثلًا g می ذاره و اتفاقی که می افته این میشه دوتا بلاک خراب میشه  
نکته: اینجا اون بحث دوتا plaintext برابر و ciphertext برابر رو نداریم یعنی اینجا دوتا  
plaintext برابر دیگه ciphertext های برابر به ما نمی ده

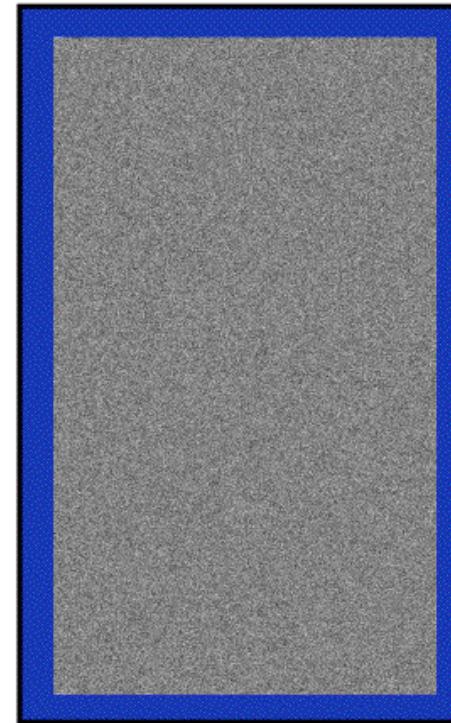
# CBC Mode

- Identical plaintext blocks yield different ciphertext blocks — this is very good!
- But what about errors in transmission?
  - If  $C_1$  is garbled to, say,  $G$  then
$$P_1 \neq C_0 \oplus D(G, K), P_2 \neq G \oplus D(C_2, K)$$
  - But  $P_3 = C_2 \oplus D(C_3, K), P_4 = C_3 \oplus D(C_4, K), \dots$
  - Automatically recovers from errors!
- Cut and paste is still possible, but more complex (and will cause garbles)

اینجا فرض میکنم  $c_1$  عوض شده برای ساخت  $p_2$  ،  $p_1$  بهش نیاز داریم ولی برای  $p_3$  به بعد دیگه نیازی به  $c_1$  نداریم و جواب  $p_3$  به بعد درسته چون وابستگی به  $c_1$  ندارند حالا اون قسمت هایی که خراب میشه رو می تونیم یه جورایی بفهمیم چون مثلا بی معنی میشه اون قسمت یا همچین چیزی و این ایده رو به ما میده که علاوه بر اینکه برای استفاده بکنیم / محترمانه بودن از CBC استفاده میکنیم برای Integrity confidentiality هم می تونیم

# Alice Likes CBC Mode

- ❑ Alice's uncompressed image, Alice CBC encrypted (TEA)



- ❑ Why does this happen?
- ❑ Same plaintext yields different ciphertext!

- اینجا عکس الیس رو توی مد CBC هم رمزش کرده  
اینجا اگر بگیم هر 16 تا پیکسل ما یک بلاک باشه و این بلاک هایی که سفیده درسته با هم  
برابرند ولی به خاطر ciphertext شدنشون chained شون با هم دیگه برابر نیست

# Counter Mode (CTR)

- CTR is popular for random access
- Use block cipher like a stream cipher

## Encryption

$$C_0 = P_0 \oplus E(IV, K),$$

$$C_1 = P_1 \oplus E(IV+1, K),$$

$$C_2 = P_2 \oplus E(IV+2, K), \dots$$

## Decryption

$$P_0 = C_0 \oplus E(IV, K),$$

$$P_1 = C_1 \oplus E(IV+1, K),$$

$$P_2 = C_2 \oplus E(IV+2, K), \dots$$

- Note: CBC also works for random access
  - But there is a significant limitation in this case

توی این مد ما داریم  $\text{IV}$  رو رمز میکنیم و به ما کلید جدید می ده نکته: هر دو طرف گیرنده و فرستنده باید از encryption استفاده کنند گر  $\text{AES}$  ما block cipher باشه هر بار چندتا بیت کلید تولید میکنه؟ به اندازه بلاک دادمون مثل اگر  $\text{AES}$  برای ما 128 بیت برای ما کلید تولید میکنه مد CTR هم برای random access استفاده میشه --> اگر ciphertext که فرستنده فرستاده و اسه گیرنده رسید به دست گیرنده اون همون جا میتونه اون ciphertext خاص رو بازش کنه و اینش است که random access براش ایجاد میکنه ینی ما وابسته به کسی نیستیم ینی مثلما 4 داریم برای گیرنده می فرستیم و مثل 2 تاش به دست گیرنده رسیده و بقیش نرسیده و همون دوتایی که به دستش رسیده رو همون موقع می تونه رمزگشایی کنه ولی برای CBC برای رمزگشایی به بلاک قبلی نیاز داشتیم ینی ciphertext قبلی رو نیاز داریم و اسه رمزگشایی --> بحث اینه که تا ciphertext رسید به دست ما می تونیم رمزگشایی کنیم که این میشه این که ما random access داریم نکته: CBC هم می تونه random access داشته باشه ولی یه کوچولو فرق داره و اون اینه که وابسته به قبلی ها که بالا گفتیم نکته: CTR خیلی بهتره

# Integrity



# Data Integrity

- **Integrity** — detect unauthorized writing  
(i.e., detect unauthorized mod of data)
- Example: Inter-bank fund transfers
  - Confidentiality is nice, but integrity is *critical*
- Encryption provides **confidentiality**  
(prevents unauthorized disclosure)
- Encryption alone does **not** provide integrity
  - One-time pad, ECB cut-and-paste, etc., etc.

- integrity ینی این که ما نوشتته رو سختش کنیم برای مهاجم و بتوانیم تشخیص بدیم این قسمت از داده ما دست کار شده ینی این همون چیزی نیست که الیس و اسه باب فرستاده مثالی از این که integrity confidentialiy مهم نباشه ولی مهم باشه --> توی بحث های مالی مخصوصا توی بلاک چین ینجا هم میخوایم confidentiality بذاریم کنار و صرفا به integrity بپردازیم و از یه چیزی به اسم mac استفاده میکنیم

# MAC

- Message Authentication Code (MAC)
  - Used for data **integrity**
  - Integrity **not** the same as confidentiality
- MAC is computed as **CBC residue**
  - That is, compute CBC encryption, saving only final ciphertext block, the MAC
  - The MAC serves as a cryptographic checksum for data

## authentication میشه احراز اصالت

اینجا میخوایم بفهمیم پیام ما اصیل هست یا نه؟ ینی چیزی که برای ما مهمه این که این دقیقاً پیام خود الیس هست یا نه؟ و از mac استفاده می کنیم

mac از CBC استفاده میکنه --> کاری که میکنه اینه که میاد CBC رو حساب میکنه و mac این میشه که اون اخرين خروجی که از اخرين بلاک plaintext ما وارد شده و خروجی اخري که میگیریم این مک ما رو می سازه

وقتی که میخوایم integrity رو با گیرنده چک کنیم کاری که میکنیم این است که ما plaintext ها رو براش می فرستیم با مک --> ینی اینجا اون بحث confidentialy رو گذاشتیم کنار

الیس اینجا مک و iv و plaintext رو با هم می فرسته

# MAC Computation

- MAC computation (assuming N blocks)

$$C_0 = E(IV \oplus P_0, K),$$

$$C_1 = E(C_0 \oplus P_1, K),$$

$$C_2 = E(C_1 \oplus P_2, K), \dots$$

$$C_{N-1} = E(C_{N-2} \oplus P_{N-1}, K) = \text{MAC}$$

- Send IV,  $P_0, P_1, \dots, P_{N-1}$  and MAC
- Receiver does same computation and verifies that result agrees with MAC
- Both sender and receiver must know K

- گیرنده از اون **plaintext** هایی که گرفته از الیس به یه سری مک می رسه و اونو مقایسه میکنه با مکی که فرستنده فرستاده و می بینه ایا دست کاری شده یا نه که میشه همون بحث **integrity**

# Does a MAC work?

- Suppose Alice has 4 plaintext blocks
- Alice computes

$$\mathbf{C}_0 = E(IV \oplus P_0, K), \mathbf{C}_1 = E(\mathbf{C}_0 \oplus P_1, K),$$

$$\mathbf{C}_2 = E(\mathbf{C}_1 \oplus P_2, K), \mathbf{C}_3 = E(\mathbf{C}_2 \oplus P_3, K) = \text{MAC}$$

- Alice sends IV,  $P_0, P_1, P_2, P_3$  and **MAC** to Bob
- Suppose Trudy changes  $P_1$  to X
- Bob computes

$$\mathbf{C}_0 = E(IV \oplus P_0, K), \mathbf{C}_1 = E(\mathbf{C}_0 \oplus X, K),$$

$$\mathbf{C}_2 = E(\mathbf{C}_1 \oplus P_2, K), \mathbf{C}_3 = E(\mathbf{C}_2 \oplus P_3, K) = \text{MAC} \neq \text{MAC}$$

- It works since error propagates into MAC
- Cannot make **MAC** == **MAC** without key K

- توى اين مثال فرض ميکنيم  $p_1$  عوض شد و شد  $x$   
اينجا زنجير وار بهم وصل شدند --> اينجا چون سمت گيرنده داره encrypt ميکنه  
plaintext های ما به صورت زنجير وار بهم متصل اند برای همين اگر ما يکی رو اون وسط  
عوض کردیم تا اخرين بلاک که داشت مک رو می ساخت داره دستکاري ميشه  
نکته: کاري که سمت گيرنده داره انجام ميشه روی plaintext ها است و اینا بهم زنجير شدند  
و این باعث ميشه تا مک اين تغيير بکنه

# Confidentiality and Integrity

- Encrypt with one key, MAC with another key
- Why not use the same key?
  - Send last encrypted block (MAC) twice?
  - This cannot add any security!
- Using different keys to encrypt and compute MAC works, even if keys are related
  - But, twice as much work as encryption alone
  - Can do a little better — about 1.5 “encryptions”
- Confidentiality and integrity with same work as one encryption is a research topic

همزمانی integrity , confidentiality ها را بفرستیم؟  
کاری که میکنه اینه که میاد از plaintext سمت گیرنده به ciphertext میرسیم و بعد با  
اون plaintext هایی که داریم میبینیم اون مکی که برای ما فرستاده با اون مکه که حساب  
کردیم یکی است یا نه

اینکه بیایم مک رو دوبار بفرستیم بدرد نمیخوره پس به یک کلید دیگر هم نیاز داریم اینجا  
اینجا عملیات ما دوبرابر است چون سمت گیرنده و فرستنده باید یه بار CBC رو بریم برای  
تا مک رو بسازیم و ciphertext و confidential هارو بسازیم و یه بار با یه کلید دیگه باید CBC بریم تا آخر

سمت فرستنده: یه دور ciphertext ها رو رمز میکنه که ciphertext ها رو بسازه و یه دور  
دیگه plaintext ها رو رمز میکنه با یک کلید دیگه که مک رو بسازه  
سمت گیرنده: فرستنده ciphertext ها رو میفرسته و مک رو حالا گیرنده ciphertext ها  
رو داره و یه دور با اون کلید اولیه میاد رمزگشایی میکنه و plaintext رو به دست میاره و  
این plaintext رو با کلید دومیه رمز میکنه که بینه مکشون یکی هست یا نه --> هر دو  
طرف دارند دوبار عملیات CBC رو انجام میدند --> جلوتر اینو 1.5 میکنیم

# Uses for Symmetric Crypto

- Confidentiality
  - Transmitting data over insecure channel
  - Secure storage on insecure media
- Integrity (MAC)
- Authentication protocols (later...)
- Anything you can do with a hash function (upcoming chapter...)

رمزگاری متقارن رو برای اینا استفاده میکنیم:

confidentiality : چه وقتی که بخوایم داده ای از طریق کانال نامن عبور بدیم یا چه وقتی که ذخیرش کنیم روی مدیا یا رسانه ای که می دونیم دسترسی روش هست و میدونیم مديای کاملا امنی نیست

integrity

authentication پروتکل های  
هش فانکشنها

# Chapter 4: Public Key Cryptography

You should not live one way in private, another in public.  
— Publilius Syrus

Three may keep a secret, if two of them are dead.  
— Ben Franklin

## فصل 4: رمزنگاری کلید عمومی

# Public Key Cryptography

- Two keys, one to encrypt, another to decrypt
  - Alice uses Bob's **public key** to encrypt
  - Only Bob's **private key** decrypts the message
- Based on "trap door, one way function"
  - "One way" means easy to compute in one direction, but hard to compute in other direction
  - Example: Given  $p$  and  $q$ , product  $N = pq$  easy to compute, but hard to find  $p$  and  $q$  from  $N$
  - "Trap door" is used when creating key pairs

- کلید عمومی از دو تا کلید استفاده میکنه

پس بخوايم برای باب یه پیامی رو رمزکنیم از کلید عمومیش استفاده میکنیم و اگه باب بخواد  
بگه من یه چیزی رو گفتم از کلید خصوصیش استفاده میکنه

trap door, one way function و عملیاتی که توی کلید عمومی انجام میدیم مبتنی بر است

برای این میگه one way function چون از یه طرف در یک جهت حساب کردنش راحته  
مثلما دو تا عدد داریم مثل  $p, q$  که اول باشند و این  $p, q$  اگر داشته باشیم و در هم ضرب  
کنیم کار راحتی ولی بر عکش اگه بخوایم فاکتور بگیریم ینی بگیم این  $N$  که عدد بزرگی است  
ضرب کدوم دو تا عدد اول بوده این سخته این میشه بخش one way function اش

بخش trap door اش هم میشه مثل همین فاکتورگیری  
پس شکستن این عدد  $N$  بر عکش سخته و مشخص نیست

# Public Key Cryptography

- Encryption
  - Suppose we **encrypt** M with Bob's public key
  - Bob's private key can **decrypt** C to recover M
- Digital Signature
  - Bob **signs** by "encrypting" with his private key
  - Anyone can **verify** signature by "decrypting" with Bob's public key
  - But only Bob could have signed
  - Like a handwritten signature, but much better...

---

# Knapsack



-----

# Knapsack Problem

- Given a set of  $n$  weights  $W_0, W_1, \dots, W_{n-1}$  and a sum  $S$ , find  $a_i \in \{0,1\}$  so that

$$S = a_0 W_0 + a_1 W_1 + \dots + a_{n-1} W_{n-1}$$

(technically, this is the *subset sum problem*)

- Example

- Weights (62,93,26,52,166,48,91,141)
- Problem: Find a subset that sums to  $S = 302$
- Answer:  $62 + 26 + 166 + 48 = 302$

- The (general) knapsack is NP-complete

مسئلش:  $n$  تا وزن داریم از  $W_0, \dots, W_{n-1}$  و یک جمعی داریم به اسم  $S$  و یک سری ضریب هم داریم  $a_i$  ها که از صفر تا  $n-1$  است ینی  $n$  تا هم ضریب داریم به ما یک سری وزن می دهند و میگن این  $a_i$  ها رو چجوری پیدا کن که یک جمع خاص رو این وزن ها با هم تشکیل بدهند

اگر  $S=302$  باشه جواب میشه  
 $48+166+26+62$   
کوله پشتی یک مسئله سخت است

# Knapsack Problem

- General knapsack (GK) is hard to solve
- But superincreasing knapsack (SIK) is easy
- SIK — each weight greater than the sum of all previous weights
- Example
  - Weights (2,3,7,14,30,57,120,251)
  - Problem: Find subset that sums to  $S = 186$
  - Work from largest to smallest weight
  - Answer:  $120 + 57 + 7 + 2 = 186$

یک زیرمجموعه از مسائل کوله پشتی است به اسم **superincreasing knapsack** که بهش میگن **SIK** که حل کردن این مسئله راحته و مسئله میگه که وزن هایی که چیدیم و هر کدام از وزن ها از جمع همه قبلی هاش بیشتره یعنی 3 از 2 بیشتره و 7 از جمع  $3+2$  بیشتره و 14 هم از جمع  $7+3+2$  و به همین ترتیب حل کردن **SIK** برآمده راحته مثلا  $S=186$  و می خوایم  $a_i$  هارو پیدا کنیم و کاری که میکنیم اینه که از بیشترین وزن کمترین شروع میکنیم به حرکت کردن ضریب 251 صفر است و ضریب 120 یک است بعدش میایم  $186-120=66$  میکنیم که میشه 66 پس ضریب 57 هم یک است و به همین ترتیب میریم و هر جا که یک شد اون مقدارو در نظر میگیریم و بعد از **sum** کم میکنیم و می ریم جلو

# Knapsack Cryptosystem

1. Generate superincreasing knapsack (SIK)
  2. Convert SIK to "general" knapsack (GK)
  3. Public Key: GK
  4. Private Key: SIK and conversion factor
- Goal...
- o Easy to encrypt with GK
  - o With private key, easy to decrypt (solve SIK)
  - o Without private key, Trudy has no choice but to try to solve GK

- سیستم رمز کوله پشتی:

اول کار کلیدمون رو تولید میکنیم بعد SIK رو تبدیل میکنه به GK

# Example

- Start with  $(2,3,7,14,30,57,120,251)$  as the SIK
- Choose  $m = 41$  and  $n = 491$  ( $m, n$  relatively prime,  $n$  exceeds sum of elements in SIK)
- Compute "general" knapsack

$$2 \cdot 41 \bmod 491 = 82$$

$$3 \cdot 41 \bmod 491 = 123$$

$$7 \cdot 41 \bmod 491 = 287$$

$$14 \cdot 41 \bmod 491 = 83$$

$$30 \cdot 41 \bmod 491 = 248$$

$$57 \cdot 41 \bmod 491 = 373$$

$$120 \cdot 41 \bmod 491 = 10$$

$$251 \cdot 41 \bmod 491 = 471$$

- "General" knapsack:  $(82,123,287,83,248,373,10,471)$

- مثال:

مثلا باب می خواهد کلید عمومی خودش را ایجاد کنه، کلید خصوصی را او مد یه  $SIK$  ساخت و بعد دو تا عدد انتخاب کرد که عدد  $n$  از مجموع همه عناصری که توی  $SIK$  داشت بیشتر بود و عدد  $m$  نسبت به  $n$  اول بود و بعد از اون  $GK$  رو ساختیم به این صورت که هر کدام از اون عناصر رو ضربدر  $m$  می کردیم توی مد  $n$  و نهایتا خروجی ها میشد  $GK$

# Knapsack Example

- **Private key:** (2,3,7,14,30,57,120,251)

$$m^{-1} \bmod n = 41^{-1} \bmod 491 = 12$$

- **Public key:** (82,123,287,83,248,373,10,471),  $n=491$

- **Example: Encrypt** 10010110

$$82 + 83 + 373 + 10 = 548$$

- **To decrypt, use private key...**

- $548 \cdot 12 = 193 \bmod 491$
  - Solve (easy) SIK with  $S = 193$
  - Obtain plaintext 10010110

- حالا چجوری رمزش میکنیم؟

# Knapsack Weakness

- **Trapdoor:** Convert SIK into “general” knapsack using modular arithmetic
- **One-way:** General knapsack easy to encrypt, hard to solve; SIK easy to solve
- But, the knapsack cryptosystem is **insecure**
  - Broken in 1983 with Apple II computer
  - The attack uses **lattice reduction**
- “General knapsack” is not general enough!
  - This special case of knapsack is easy to break

## - نقطه ضعف:

: trapdoor یعنی اون کسی که داره کلید خودش رو می سازه می اوMD SIK تبدیل میکرد به GK با استفاده از عملیات پیمانه ای one way با GK راحت بود رمزنگاری کنیم و اگه SIK رو داشتیم راحت بود که اینو رمزگشایی کنیم

نکته ای که هست اینه که کوله پشتی ما امن نیست --> سال 1983 توسط شمیر این شکسته شد و حمله ای هم که داشت این بود حمله lattice reduction بود ( یه توضیحی راجع به این حمله ینی فضایی که میخوایم بهش حمله کنیم رو متوجه میشیم که کوچیک تره ینی مهاجم به جای اینکه بخواهد مسئله کوله پشتی رو حل کنه که یک مسئله سخته میاد یک زیرمجموعه ای از این مسئله رو حل میکنه چون واقعا جنرال نیست و از SIK رسیدیم به GK )

نکته: این GK ما واقعا جنرال نیست توی GK وزن هایی که داریم به صورت رندوم بودن ولی اینجا رندوم نبودن ینی ما از یک SIK رسیدیم به GK و این باعث میشه که ضعیف بشه

# RSA

-----

# RSA

- Invented by Clifford Cocks (GCHQ) and Rivest, Shamir, and Adleman (MIT)
  - RSA is the *gold standard* in public key crypto
- Let p and q be two large prime numbers
- Let  $N = pq$  be the **modulus**
- Choose e relatively prime to  $(p-1)(q-1)$
- Find d such that  $ed \equiv 1 \pmod{(p-1)(q-1)}$
- **Public key is  $(N, e)$**
- **Private key is d**

- اولین نفری که RSA را تولید کرد rivest , shamir , adleman نبودند بلکه GCHQ بود از cocks

عمدتاً رمزنگاری کلید عمومی با اینا است

RSA: دو تا عدد بزرگ اول  $p, q$  رو میگیریم و در هم ضرب میکنیم که این  $N$  میشه پیمانه ما نکته: اگر بخوایم یک عددی رو بگیریم و اوно بشکنیم به عناصر عدد اول که اینو ساختن که این مسئله سخت است و بهش مسئله تجزیه میگن

کسی که داره کلید رو می سازه یک  $p, q$  انتخاب کرد و پیمانه رو باهاش ساخت و یک  $e$  انتخاب میکنه که نسبت به  $(p-1)(q-1)$  اول است و  $d$  رو انتخاب میکنیم طوری که رابطه  $ed=1 \text{ mod } (p-1)(q-1)$  برآمون بسازه ینی  $d$  توی اون پیمانه معکوس  $e$  است

صفحه 10 و 11 و 12 جزوه رو حتما بخون

# RSA

- Message  $M$  (plaintext) treated as an integer
- To encrypt  $M$  we compute  
$$C = M^e \text{ mod } N$$
- To decrypt ciphertext  $C$ , we compute  
$$M = C^d \text{ mod } N$$
- Recall that  $e$  and  $N$  are public
- If Trudy can factor  $N = pq$ , she can use  $e$  to easily find  $d$  since  $ed = 1 \text{ mod } (p-1)(q-1)$
- So, **factoring the modulus breaks RSA**
  - Is factoring the only way to break RSA?

اینجا الیس می خواهد یه پیامی رو رمز کنه برای باب پس میاد:  
اول پیام رو تبدیل به عدد میکنه و میشه یک  $M$  که این یک عدد است و این  $M$  رو به توان  $e$  می رسوونه توی مد  $n$  و می فرسته برای باب و باب این  $C$  رو گرفته و کافیه این  $C$  رو به توان  $d$  برسونه توی همون پیمانه  $n$  و با این کار  $m$  به دست میاد و اگه  $trudy$  تونست این  $N=pq$  رو بشکونه پس می فهمه فی ان چی بوده و این محاسبات رو انجام بده ولی حسابش اینقدر راحت نیس

بنابراین اون چیزی که RSA رو بشکونه بحث factoring است ولی تنها روش برای شکستن RSA نیست

# Does RSA Really Work?

- Given  $C = M^e \text{ mod } N$  we want to show that  
 $M = C^d \text{ mod } N = M^{ed} \text{ mod } N$
- We'll need Euler's Theorem:  
If  $x$  is relatively prime to  $n$  then  $x^{\varphi(n)} = 1 \text{ mod } n$
- Facts:
  - 1)  $ed = 1 \text{ mod } (p - 1)(q - 1)$
  - 2) By definition of "mod",  $ed = k(p - 1)(q - 1) + 1$
  - 3)  $\varphi(N) = (p - 1)(q - 1)$
- Then  $ed - 1 = k(p - 1)(q - 1) = k\varphi(N)$
- So,  $C^d = M^{ed} = M^{(ed - 1) + 1} = M \cdot M^{ed - 1} = M \cdot M^{k\varphi(N)}$   
 $= M \cdot (M^{\varphi(N)})^k \text{ mod } N = M \cdot 1^k \text{ mod } N = M \text{ mod } N$

- اصلا RSA کار میکنه؟ ینی این  $C$  رو که باب میگیره اگه برسونه به توان  $d$  ایا واقعا  $M$  میشه یا نه؟ بله کار میکنه به قضیه اویلر نیاز داریم:

# RSA Keypair Example

- Generate RSA key pair...
  - Select “large” primes  $p = 11, q = 3$
  - Then  $N = pq = 33$  and  $(p - 1)(q - 1) = 20$
  - Choose  $e = 3$  (relatively prime to 20)
  - Find  $d$  such that  $ed = 1 \text{ mod } 20$ 
    - We find that  $d = 7$  works
- Public key:  $(N, e) = (33, 3)$
- Private key:  $d = 7$

- 3 نسبت به 20 اوله پس e رو گرفته 3  
و d هم میگیره 7

# Textbook RSA Example

- **Public key:**  $(N, e) = (33, 3)$
- **Private key:**  $d = 7$
- Suppose message to encrypt is  $M = 8$
- **Ciphertext C is computed as**
$$C = M^e \bmod N = 8^3 = 512 = 17 \bmod 33$$
- **Decrypt C to recover the message M by**
$$\begin{aligned}M &= C^d \bmod N = 17^7 = 410,338,673 \\&= 12,434,505 * 33 + 8 = 8 \bmod 33\end{aligned}$$
- Why is this “textbook” RSA?

-  
پیام رو چگونه رمزکنیم؟  
الیس 17 رو میفرسته برای باب  
باب 17 رو میگیره و به توان d میرسونه

# More Efficient RSA (1)

- Modular exponentiation example
  - $5^{20} = 95367431640625 = 25 \text{ mod } 35$
- A better way: **repeated squaring**
  - $20 = 10100$  base 2
  - $(1, 10, 101, 1010, 10100) = (1, 2, 5, 10, 20)$
  - Note that  $2 = 1 \cdot 2$ ,  $5 = 2 \cdot 2 + 1$ ,  $10 = 2 \cdot 5$ ,  $20 = 2 \cdot 10$
  - $5^1 = 5 \text{ mod } 35$
  - $5^2 = (5^1)^2 = 5^2 = 25 \text{ mod } 35$
  - $5^5 = (5^2)^2 \cdot 5^1 = 25^2 \cdot 5 = 3125 = 10 \text{ mod } 35$
  - $5^{10} = (5^5)^2 = 10^2 = 100 = 30 \text{ mod } 35$
  - $5^{20} = (5^{10})^2 = 30^2 = 900 = 25 \text{ mod } 35$
- No huge numbers and it's efficient!

میخوایم 5 به توان 20 رو حساب کنیم راه سنتیش اینه که بیایم 5 رو 20 بار در خودش ضرب کنیم --> راه سنتیه راه خوبی نیس پس

از راه صنعتی میریم که حساب و کتاب توی این روش راحت تر است روش ما repeated squaring است که میگه:

20 رو در مبنای دو به دست میاریم

ص 12 جزوه رو بخون --> double or add one

از صفر شروع میکنیم و می خوایم بررسیم به 20 اول از همه یکی بهش اضافه میکنیم و میشه یک و بعد یکو دوبرابر میکنیم میشه دو و بعد دوبرابر میکنیم و به اضافه یک میکنیم میشه 5 و به همین ترتیب میریم جلو تا به 20 بررسیم

پس می تونیم دوبرابر کنیم یا به اضافه یک کنیم یا هر دو رو پیاده کنیم حالا میخوایم 5 به توان 20 حساب کنیم از روی محاسبات ببین (:)

# More Efficient RSA (2)

- Use  $e = 3$  for all users (but not same  $N$  or  $d$ )
  - + Public key operations only require 2 multiplies
  - o Private key operations remain expensive
  - If  $M < N^{1/3}$  then  $C = M^e = M^3$  and **cube root attack**
  - For any  $M$ , if  $C_1, C_2, C_3$  sent to 3 users, cube root attack works (uses Chinese Remainder Theorem)
- Can prevent cube root attack by padding message with random bits
- Note:  $e = 2^{16} + 1$  also used ("better" than  $e = 3$ )

-  
اما برای اینکه RSA بیشتر کارآمدتر باشه اومدن گفتن ما  $e=3$  میگیریم برای همه کاربران ولی منتها هر کس اون چیزی رو که عوض میکنه  $n$  و نتیجتا  $d$  باشه چرا این کارو میکنه؟ برای اینکه معمولاً یک طرف ارتباط یک سروری است و یه عالمه کلاینت داره و میخواهد با اینا کلید متقارن شیر بکنه سمت سرور چون کار این کلید شیر کردن و به توان رسوندن اینا و برسونه به باب های مختلف سمت این سرور خیلی محاسبات زیاد میشه و اسه همین یه دیفالت گذاشتن که  $e=3$  باشه

اگر  $M$  از  $N$  به توان  $1/3$  کوچکتر باشه یک حمله ای اتفاق می افته به اسم cube root attack چه شکلی میشه؟ صفحه 12 جزوه نوشتمن دیفالت بعدی اینه که بیایم  $e$  رو بگیریم  $2^{16+1}$  بذاریم

# Diffie-Hellman

---

# Diffie-Hellman Key Exchange

- Invented by Williamson (GCHQ) and, independently, by D and H (Stanford)
- A "key exchange" algorithm
  - Used to establish a shared symmetric key
  - Not for encrypting or signing
- Based on **discrete log** problem
  - **Given:**  $g$ ,  $p$ , and  $g^k \text{ mod } p$
  - **Find:** exponent  $k$

دیفی هلمن رو Williamson ساخته بودش ولی به اسم دیفی هلمن می شناسیم  
الگوریتم دیفی هلمن که یک الگوریتم تبادل کلید است ینی یک الگوریتمی که ما استفاده کنیم  
واسه رمزنگاری نیست

پس از دیفی هلمن استفاده میشه واسه اینکه اون کلید متقارنه رو به اشتراک بذاریم برای دو  
طرف ارتباط و برای رمزنگاری و امضا استفاده نمیشه

روش هایی که داریم برای کلید عمومی trap door base هستند که یک مسئله سخت  
ریاضی بود که یه طرفشو که می رفتیم راحت بود برامون ولی برگشتش سخت بود  
مسئله دیفی هلمن اینجا لگاریتم گستته است

مسئله لگاریتم گستته که حالت پیمانه ای پیدا میکنه مسئله سختی است ینی اگر ما  $g, p$  رو  
داشته باشیم و یک عددی مثل  $g$  به توان  $k$  مد  $p$  رو داشته باشیم این که  $k$  رو بدست بیاریم یا  
لگاریتم گستته رو حل بکنیم این مسئله سختی است  
و دیفی هلمن براساس این لگاریتم گستته است

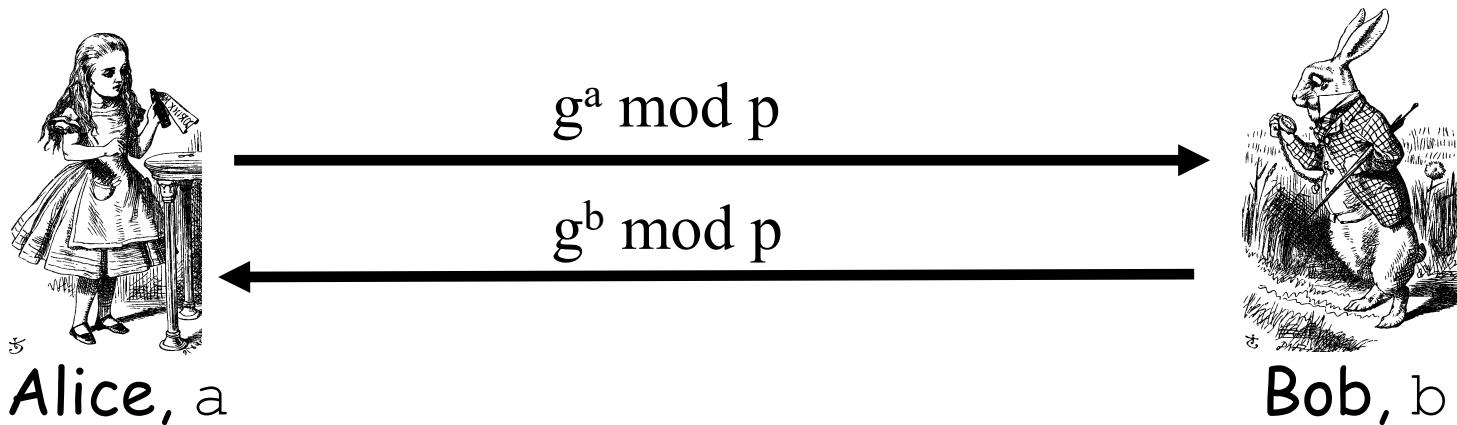
# Diffie-Hellman

- Let  $p$  be prime, let  $g$  be a **generator**
  - For any  $x \in \{1, 2, \dots, p-1\}$  there is  $n$  s.t.  $x = g^n \text{ mod } p$
- Alice selects her private value  $a$
- Bob selects his private value  $b$
- Alice sends  $g^a \text{ mod } p$  to Bob
- Bob sends  $g^b \text{ mod } p$  to Alice
- Both compute shared secret,  $g^{ab} \text{ mod } p$
- Shared secret can be used as symmetric key

چه شکلی الیس و باب این تبادل کلید رو انجام میدن؟  
الیس یک عدد خصوصی رو برای خودش انتخاب میکنه مثل  $a$   
و باب هم همین کارو میکنه مثل  $b$   
و الیس  $p$  mode  $g^a$  رو ارسال میکنه و اسه باب که این داره توی کانال ناامن داره ارسال  
میشه و کسایی مثل  $trudy$  و خود باب از این مقدار به  $a$  نمیتونن برسن و چوری می تونن  
برسن اگه بتونن لگاریتم گسته رو حل کنند که گفتیم لگاریتم گسته یک مسئله سخت است  
باب برای الیس  $p$  mode  $g^b$  رو ارسال میکنه  
و هردو تاشون میان  $p$  mode  $g^{ab}$  رو حساب میکنند و این میشه یک رازی که این دو نفر  
باهم شیر میکنند

# Diffie-Hellman

- **Public:**  $g$  and  $p$
- **Private:** Alice's exponent  $a$ , Bob's exponent  $b$



- Alice computes  $(g^b)^a = g^{ba} = g^{ab} \text{ mod } p$
- Bob computes  $(g^a)^b = g^{ab} \text{ mod } p$
- They can use  $K = g^{ab} \text{ mod } p$  as symmetric key

---

# Diffie-Hellman

- Suppose Bob and Alice use Diffie-Hellman to determine symmetric key  $K = g^{ab} \bmod p$
- Trudy can see  $g^a \bmod p$  and  $g^b \bmod p$ 
  - But...  $g^a g^b \bmod p = g^{a+b} \bmod p \neq g^{ab} \bmod p$
- If Trudy can find  $a$  or  $b$ , she gets  $K$
- If Trudy can solve **discrete log** problem, she can find  $a$  or  $b$

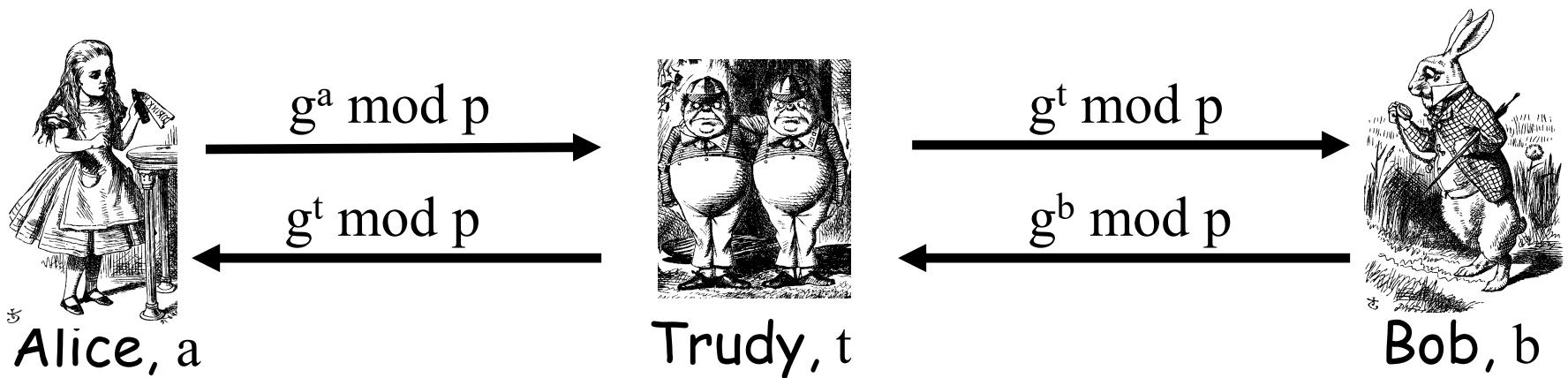
-  
چیا داره؟ داره کانال نامن رو شنود میکنه پس  $g^a$  mode p ,  $g^b$  mode p trudy داره

پس ایا می تونه  $g^{ab}$  mode p رو بسازه؟ نه نمی تونه

پس trudy برای اینکه بتونه  $g^{ab}$  mode p رو بسازه یا باید a رو داشته باشه یا b یا k یا مسئله لگاریتم گسته رو حل بکنه تا بتونه اینو بشکنه

# Diffie-Hellman

- Subject to man-in-the-middle (MiM) attack



- Trudy shares secret  $g^{at} \text{ mod } p$  with Alice
- Trudy shares secret  $g^{bt} \text{ mod } p$  with Bob
- Alice and Bob don't know Trudy is MiM

- حمله MitM به دیفی هلمن:

بنی  $p$   $g^a \text{ mod } p$  که رسید به دست  $trudy$  اینو نفرسته بباد خودش یه  $t$  انتخاب بکنه و  $g^a t \text{ mod } p$  رو بفرسته برای باب و برای باب هم همین طور چون اینجا  $trudy$  کلید رو داره پس هر پیامی که الیس بفرسته  $trudy$  میاد این پیام رو رمزگشایی میکنه و با  $bt$  رمزش میکنه و می فرسته واسه باب و برای باب هم همین طور بنی  $p$  رمزگشایی میکنه و با  $g^a t b \text{ mod } p$  رمزش میکنه و میفرسته واسه الیس بنی این وسط شده واسطه این دوتا

# Diffie-Hellman

- How to prevent MiM attack?
  - Encrypt DH exchange with symmetric key
  - Encrypt DH exchange with public key
  - Sign DH values with private key
  - Other?
- At this point, DH may look pointless...
  - ...but it's not (more on this later)
- You **MUST** be aware of MiM attack on Diffie-Hellman

- چگونه از MitM جلوگیری بکنیم؟

- امضای دیجیتالی

- با کلید متقارن و کلید عمومی اینو رمزش کنیم

اگ ما کلید داشتیم چه کلید متقارن چه کلید عمومی و خصوصی دیگه چه کاری بود بیایم تبادل کلید بکنیم بخاطر همین دیفی هلمن رو می ذاریم کنار ولی نه کامل ??

# *Elliptic Curve Cryptography*

---

# Elliptic Curve Crypto (ECC)

- “Elliptic curve” is **not** a cryptosystem
- Elliptic curves provide different way to do the math in public key system
- Elliptic curve versions of DH, RSA, ...
- Elliptic curves are more efficient
  - Fewer bits needed for same security
  - But the operations are more complex, yet it is a big “win” overall

ECC یا رمز خم بیضوی یک سیستم رمزنگاری نیست بلکه یک روش ریاضی است و ازش استفاده میکنیم تا سیستم های رمزنگاری کلید عمومی و در حال حاضر استاندارد هایی که امریکا داره برای رمز عمومی ECC پس روش خیلی مهمیه این روش به نسبت بقیه روش ها خیلی کارآمد است و با طول بیت کمتری با ما همون امنیتی رو میدن که مثلا اگر ما RSA ساده رو استفاده بکنیم بهمون امنیت میده مثلا 256 بیت تا ECC معادل سه هزار خورده ای بیت تا RSA معمولیه و... توی کلید عمومی عدامون خیلی بزرگ است ینی اون  $p$  و  $q$  خیلی بزرگ است پس تا این حالت می ریم سراغ ECC با طول بیت کمتر که بهمون همون امنیت رو بده اینجا ما طول بیت کمتری داریم ولی عملیات هایی که داریم اینجا پیچیده تر است ولی بازم می ارزه

# What is an Elliptic Curve?

- An elliptic curve  $E$  is the graph of an equation of the form

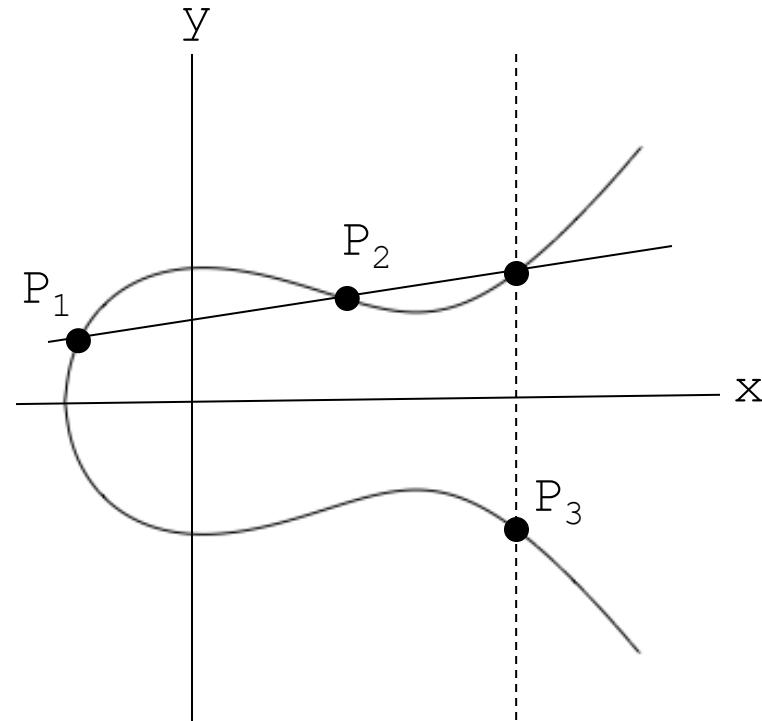
$$y^2 = x^3 + ax + b$$

- Also includes a “point at infinity”
- What do elliptic curves look like?
- See the next slide!

- فرمول خم بیضوری:

$y^2$  ینی نسبت به محور  $x$  ها متقارن است و یک نقطه ای هم در بی نهایت داریم

# Elliptic Curve Picture



- Consider elliptic curve  
 $E: y^2 = x^3 - x + 1$
- If  $P_1$  and  $P_2$  are on  $E$ , we can define addition,  
 $P_3 = P_1 + P_2$   
as shown in picture
- Addition is all we need...

- اون نقطه بى نهايٰت توى بى نهايٰت محور  $X$

اين منحنى برميگرده و بى نهايٰتمون روی محور  $y$  ميشه

نکته: يك نقطه رو که با بى نهايٰت جمع کنيم ميشه خودش اين ميشه قانونمون

فرض ميکنیم يك خم بيضوی داريم که ميخوايم دوتا نقطه رو توش جمع بکنيم اگر  $p_1, p_2$ ,

رو داشته باشيم و بخوايم  $p_3$  رو بسازيم ميایم روی منحنى  $p_2, p_1$  رو مشخص ميکنیم و

يك خطی از اينا می کشيم تا منحنى رو قطع بکنه اون نقطه ميشه  $p_3$ - پس  $p_3$  ميشه قرينه

پس محاسبات ما اينجا پيچide تر است

نکته: جمع يك نقطه با خودش ميشه مamas

نکته: اگر دوتا نقطه عمودی باشند جوابش ميشه بى نهايٰت ينى  $p_2, p_1$  توى يك راستا باشند

# Points on Elliptic Curve

- Consider  $y^2 = x^3 + 2x + 3 \pmod{5}$

$$x = 0 \Rightarrow y^2 = 3 \Rightarrow \text{no solution } \pmod{5}$$

$$x = 1 \Rightarrow y^2 = 6 = 1 \Rightarrow y = 1, 4 \pmod{5}$$

$$x = 2 \Rightarrow y^2 = 15 = 0 \Rightarrow y = 0 \pmod{5}$$

$$x = 3 \Rightarrow y^2 = 36 = 1 \Rightarrow y = 1, 4 \pmod{5}$$

$$x = 4 \Rightarrow y^2 = 75 = 0 \Rightarrow y = 0 \pmod{5}$$

- Then points on the elliptic curve are

$(1, 1) \quad (1, 4) \quad (2, 0) \quad (3, 1) \quad (3, 4) \quad (4, 0)$

and the point at infinity:  $\infty$

مثال:

این خم بیضوی رو داریم توی مد 5  
پس X ها میشه از 0 تا 4

X می تونه هر عددی باشه نهایتاً متش رو میگیریم و می ره توی مد صفر تا 4  
اینجا نقاط رو به دست میاریم + نقطه بی نهایت

# Elliptic Curve Math

□ Addition on:  $y^2 = x^3 + ax + b \pmod{p}$

$$P_1 = (x_1, y_1), P_2 = (x_2, y_2)$$

$P_1 + P_2 = P_3 = (x_3, y_3)$  where

$$x_3 = m^2 - x_1 - x_2 \pmod{p}$$

$$y_3 = m(x_1 - x_3) - y_1 \pmod{p}$$

And  $m = (y_2 - y_1) * (x_2 - x_1)^{-1} \pmod{p}$ , if  $P_1 \neq P_2$

$$m = (3x_1^2 + a) * (2y_1)^{-1} \pmod{p}, \text{ if } P_1 = P_2$$

Special cases: If  $m$  is infinite,  $P_3 = \infty$ , and  
 $\infty + P = P$  for all  $P$

بعد:

نکته: برای اینکه عدداً مون، اعداد گسته ای باش که ما بتوانیم اینو توی یک رشته‌ی بیتی تعریف بکنیم و بهش برسیم می‌ایم پیمانه ای عمل می‌کنیم

برای جمعش می‌ایم شیب این خطه رو حساب می‌کنیم این شیب رو به دست می‌اریم و بعد  $x_1, x_2$ ,  
هم داریم و  $y_1, y_2$  هم داریم و می‌ایم  $x_3, y_3$  به دست می‌اریم  
دوتا حالت خاص:

اگر شیب بی نهایت باشه  $p_3$  می‌شه بی نهایت  
و اگر یک چیزی با بی نهایت جمع بشه می‌شه خودش

# Elliptic Curve Addition

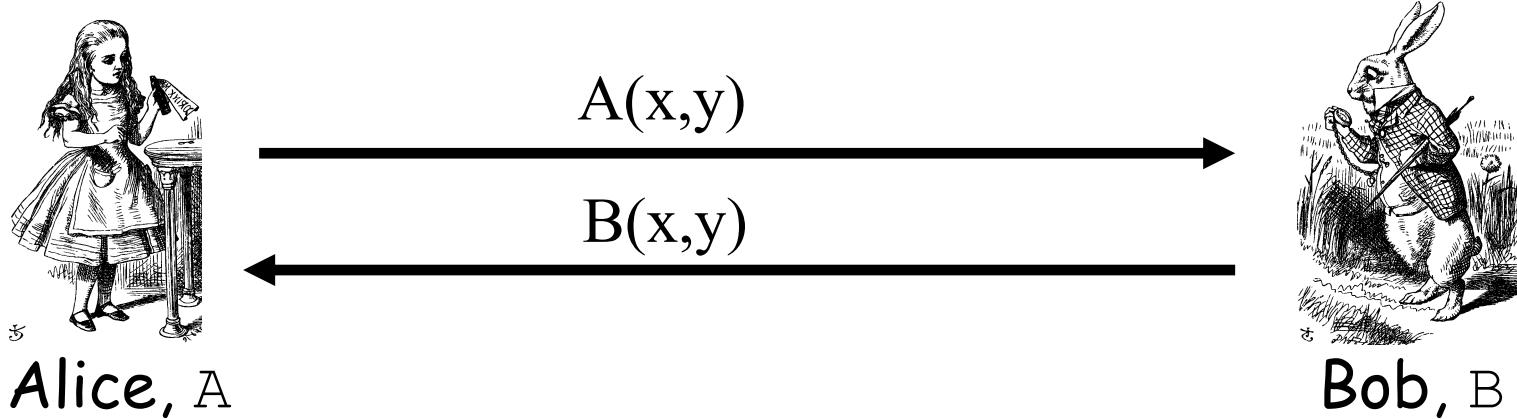
- Consider  $y^2 = x^3 + 2x + 3 \pmod{5}$ .  
Points on the curve are  $(1, 1)$   $(1, 4)$   
 $(2, 0)$   $(3, 1)$   $(3, 4)$   $(4, 0)$  and  $\infty$
- What is  $(1, 4) + (3, 1) = P_3 = (x_3, y_3)$ ?  
$$m = (1-4) * (3-1)^{-1} = -3 * 2^{-1}$$
$$= 2(3) = 6 = 1 \pmod{5}$$
$$x_3 = 1 - 1 - 3 = 2 \pmod{5}$$
$$y_3 = 1(1-2) - 4 = 0 \pmod{5}$$
- On this curve,  $(1, 4) + (3, 1) = (2, 0)$

اینجا نقاط رو داره می خواد جمع بکنه:

نکته: اون کاری که میکنیم توی خم بیضوی دیگه اون منحنی نیست و داریم با نقاط کار میکنیم  
بنی یه عالمه نقطه داریم توی صفحه و بین این نقاط داره جمع اتفاق می افته

# ECC Diffie-Hellman

- **Public:** Elliptic curve and point  $(x,y)$  on curve
- **Private:** Alice's A and Bob's B



- Alice computes  $A(B(x,y))$
- Bob computes  $B(A(x,y))$
- These are the same since  $AB = BA$

اینجا ایس و باب هر کدوم یک مقداری رو انتخاب میکنند  
ایس میاد A بار  $(x,y)$  با خودش جمع میکنه:  $A(x,y)$   
و باب هم میاد B رو ضربدر  $(x,y)$  میکنه

پس ایس A رو میدونه و  $y(x,B)$  هم به دستش رسیده پس ایس نقطه ای که به دستش رسید  
از طرف باب رو میاد A بار جمع میکنه و باب هم میاد B بار نقطه که از طرف ایس به  
دستش رسیده رو با هم جمع میکنه  
پس هر دو طرف دارند نقطه  $(x,y)$  رو AB بار با هم جمع میکنند --> پس هر دو به یک  
مقدار مشترک اینجا می رسد

چیزی که سخته اینه که مثلا trudy شنوده کرده  $A(x,y)$  رو و  $(x,y)$  هم عمومی بود و اینکه  
بیاد A رو به دست بیاره ینی بفهمه  $(x,y)$  چند بار با خودش جمع بشه تا  $y(x,A)$  ساخته بشه  
این مسئله میشه سخت  
پس سختیش شد این که ما A,B رو راحت از اون نقطه ای که داریم نمیتونیم به دست بیاریم

# ECC Diffie-Hellman

- **Public:** Curve  $y^2 = x^3 + 7x + b \pmod{37}$   
and point  $(2, 5) \Rightarrow b = 3$
- **Private:** Alice:  $A = 4$ , Bob:  $B = 7$
- Alice sends Bob:  $4(2, 5) = (7, 32)$
- Bob sends Alice:  $7(2, 5) = (18, 35)$
- Alice computes:  $4(18, 35) = (22, 1)$
- Bob computes:  $7(7, 32) = (22, 1)$

:مثال

# Larger ECC Example

- Example from Certicom ECCp-109
  - Challenge problem, solved in 2002
- Curve E:  $y^2 = x^3 + ax + b \pmod{p}$
- Where

$p = 564538252084441556247016902735257$   
 $a = 321094768129147601892514872825668$   
 $b = 430782315140218274262276694323197$

□ Now what?

- عدهای بزرگتر:

اگر ما رفتیم روی اعداد بزرگتر مثل اینجا که چالشمن 109 بیت بوده و توی سال 2002 شکسته شد

# ECC Example

- The following point P is on the curve E  
 $(x,y) = (97339010987059066523156133908935,$   
 $149670372846169285760682371978898)$
- Let  $k = 281183840311601949668207954530684$
- The  $kP$  is given by  
 $(x,y) = (44646769697405861057630861884284,$   
 $522968098895785888047540374779097)$
- And this point is also on the curve E

---

# Really Big Numbers!

- Numbers are big, but not big enough
  - ECCp-109 bit (32 digit) solved in 2002
- Today, ECC DH needs bigger numbers
- But RSA needs **way** bigger numbers
  - Minimum RSA modulus today is 1024 bits
  - That is, more than 300 decimal digits
  - That's about 10x the size of ECC example
  - And 2048 bit RSA modulus is common...

---

# Uses for Public Key Crypto

برای رمزنگاری کلید عمومی استفاده می شود

# Uses for Public Key Crypto

- Confidentiality
  - Transmitting data over insecure channel
  - Secure storage on insecure media
- Authentication protocols (later)
- Digital signature
  - Provides integrity and non-repudiation
  - No non-repudiation with symmetric keys

- استفاده هایی که از کلید عمومی میکنیم:  
هر چیزی که با استفاده از کلید متقارن بھش می رسیم برای کلید عمومی هم جوابه

# Non-non-repudiation

- Alice orders 100 shares of stock from Bob
- Alice computes **MAC** using symmetric key
- Stock drops, Alice claims she did *not* order
- Can Bob prove that Alice placed the order?
- **No!** Bob also knows the symmetric key, so he could have forged the **MAC**
- **Problem:** Bob knows Alice placed the order, but he can't prove it

- مثال:

الیس 100 تا سهم از باب سفارش داده و مکش هم با استفاده از رمز متقارن ساخته  
سهام می ریزه و الیس ادعا میکنه که همچین سفارشی نداده ایا باب می تونه ثابت کنه؟  
خیر چون باب هم خودش اون کلید متقارنه رو داشته و می تونسته این مکو بسازه و جایی نمی  
تونه ادعا کنه چون اون مکو هم الیس و هم باب می تونن بسازن

# Non-repudiation

- Alice orders 100 shares of stock from Bob
- Alice **signs** order with her private key
- Stock drops, Alice claims she did not order
- Can Bob prove that Alice placed the order?
- **Yes!** Alice's private key used to sign the order — only Alice knows her private key
- This assumes Alice's private key has not been lost/stolen

الیس 100 تا سهم سفارش داده و سفارش خودشو با کلید خصوصی امضا میکنه و وقتی سهام ریخت الیس میگه من همچین سفارشی نداشتم و در این حالت باب می تونه اثبات کنه پس اگر کسی کلید خصوصیش لو رفت اینو باید اعلام کنه که دیگه از این کلید استفاده نمیکنه و اگه سفارشی بعد از اون اعلام بشه می فهمن این مال الیس نیس مسئله non-repudiation رو با استفاده از کلید متقارن نداشتیم ولی اینجا داریم

# Public Key Notation

- **Sign message M with Alice's private key:**  $[M]_{Alice}$
- **Encrypt message M with Alice's public key:**  $\{M\}_{Alice}$
- Then

$$\{\{M\}_{Alice}\}_{Alice} = M$$

$$[\{M\}_{Alice}]_{Alice} = M$$

مسیح رو با M نشون میدیم  
و اسلاید روبه رو بخون  
نکته:

مثلا اگر الیس یک پیامی رو با کلید خصوصی خودش امضا کنه بعد یکی بیاد با کلید عمومی الیس او نو رمز کنه به اون پیام می رسه  
اگر یکی اول بیاد با کلید عمومی الیس رمز کنه و بعد الیس بیاد اینو امضا کنه دوباره به همون می رسه

# Sign and Encrypt vs Encrypt and Sign

---

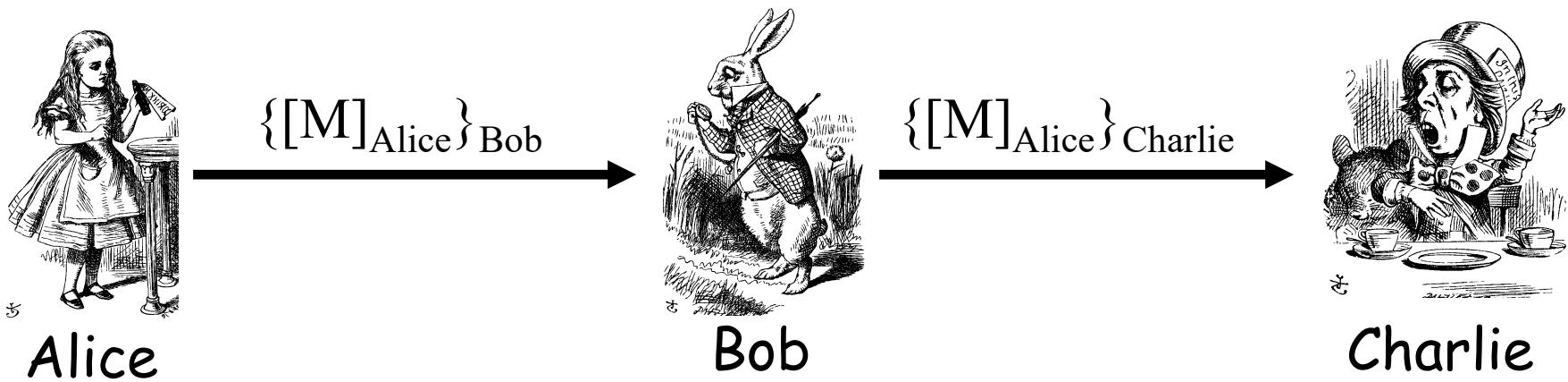
# Confidentiality and Non-repudiation?

- Suppose that we want confidentiality and integrity/non-repudiation
- Can public key crypto achieve both?
- Alice sends message to Bob
  - **Sign and encrypt:**  $\{[M]_{Alice}\}_{Bob}$
  - **Encrypt and sign:**  $[ \{M\}_{Bob} ]_{Alice}$
- Can the order possibly matter?

-  
اینجا میخوایم هم confidentiality , integrity را داشته باشیم و می خوایم جواب این سوالو بدیم که ایا کلید عمومی می تونه هر دوی اینا رو برآورده کنه؟  
اول رمز بعد امضا یا اول امضا بعد رمز?  
ترتیبیش مهمه ایا؟ ص بعدی

# Sign and Encrypt

- $M = \text{"I love you"}$



- **Q:** What's the problem?
- **A:** No problem — public key is public

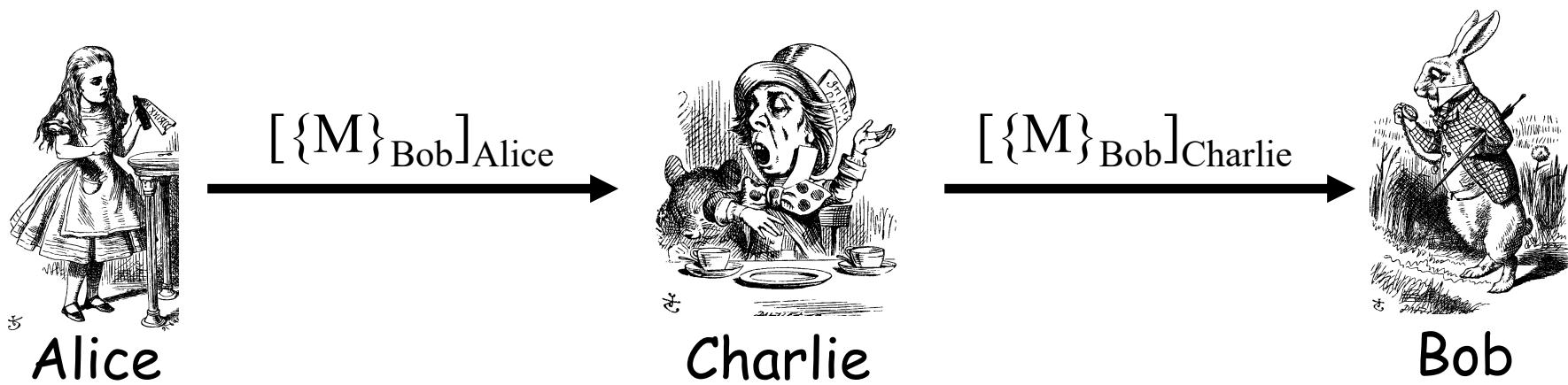
اینجا فرض شده الیس یه همچین پیامی رو امضا کرده و بعد هم رمزش میکنه و می فرسته  
برای باب

و باب هم اوون پیامو با کلید خودش رمزگشایی می کنه و با کلید عمومی چارلی رمزش میکنه و  
می فرسته برای چارلی

این وسط باب می تونه بگه الیس این پیامو فرستاده پس اینجا مشکل ما حل نشد  
توی این حالت باب و چارلی می تونن به اوون پیام برسند

# Encrypt and Sign

- $M = \text{"My theory, which is mine...."}$



- **Note** that Charlie cannot decrypt M
- **Q:** What is the problem?
- **A:** No problem — public key is public

- توى اين حالت اول رمز و بعد امضا:

اليس مى خواهد يه قضيه رو و اسه باب بفرسته پس اول رمز ميكنه و بعد امضا ميكنه

اين وسط چارلى نشسته مثل حمله MitM و ادعا مكنه اين قضيه مال اون بوده

توى اين حالت چارلى نمى تونه پيام رو بخونه ينى از نظر Confidentiality خيالمون راحته ولی integrity مختل شده

نکته: امضا که برای integrity با کلید عمومی می زنیم و عملیات کلید عمومی یک عملیات پیچیده ای است و کند است و هرچقدر هم این پیامی که میخوایم امضا کنیم کوچکتر باشه بهتر است

# Public Key Infrastructure

---

# Public Key Certificate

- ❑ Digital **certificate** contains name of user and user's public key (possibly other info too)
- ❑ It is *signed* by the issuer, a *Certificate Authority (CA)*, such as VeriSign

$M = (Alice, Alice's \text{ public key}), S = [M]_{CA}$

**Alice's Certificate** = (M, S)

- ❑ Signature on certificate is verified using CA's public key

Must verify that  $M = \{S\}_{CA}$

ما یه سری certificate یا گواهی دیجیتال داریم که شامل اسم کاربر و کلید عمومی است  
ما یه سری certificate داریم که به صورت همه پخشی منتشر میشه که هر کاربری چه کلید  
عمومی داره

چجوری اثبات کنیم هر کلید عمومی مال یک شخص است؟ خب این کلید باید از طرف یک  
کسی امضا شده باشه و مطمئن باشه یا certificate authority or ca  
پس یک کسی که قدرت اینو داره که گواهی دیجیتال به بقیه بده اینو باید امضا کرده باشه  
در کنار این که M رو می فرستیم باید S رو بفرستیم ینی هر کسی که بخواهد بدونه کلید  
عمومی الیس چیه اول چک میکنه پیامی که CA امضا کرده معتبر است یا نه پس امضای  
CA رو verify میکنه

الیس رسید دست باب و باب تازه اولین باره که کلید عمومی الیس رو میخواهد  
داشته باشه پس میاد S با کلید عمومی CA رمزش میکنه تا به M برسه و میاد چک میکنه  
اون M که بهش رسیده با این M که الیس توی گواهی دیجیتالش داده یکیه یا نه  
حالا ما چه شکلی به این CA اطمینان کنیم؟

یه مثال گفت توی ص 14 جزوه نوشتم

# Certificate Authority

- Certificate authority (CA) is a trusted 3rd party (TTP) — creates and signs certificates
- Verify signature to verify **integrity** & identity of **owner of corresponding private key**
  - Does **not** tell us anything about the identity of the **sender** of certificate — certificates are public!
- Big problem if CA makes a mistake
  - CA once issued Microsoft cert. to someone else
- A common format for certificates is X.509

- اگر CA اشتباه بکنه خیلی فاجعه است مثال: یک بار certificate مایکروسافت رو به یکی دیگه داده بود و این خودش یه فاجعه بزرگ است و یکی از فرمت های رایج هم که داریم واسه certificate هم X.509 است

# PKI

- Public Key Infrastructure (PKI): the stuff needed to securely use public key crypto
  - Key generation and management
  - Certificate authority (CA) or authorities
  - Certificate revocation lists (CRLs), etc.
- No general standard for PKI
- We mention 3 generic “trust models”
  - We only discuss the CA (or CAs)

یک سری چیزا است برای زیر ساخت کلید عمومی ینی چیزایی که لازمه که ما این تبادلات اینارو داشته باشیم:

یکی بحث تولید و مدیریت کلید است  
به CA یا CA ها نیاز داریم

یک recocation list هم نیاز داریم که اونایی که منقضی شدند رو باید توی یک لیست داشته باشیم که اینا به صورت عمومی اعلام بشه و مردم بدونند که دیگه با این کلید عمومی کار نکنند

یه سری از حملات هم روی همین موضوع است ینی مثلا فولان برنامه نمی ره چک بکنه که ایا این کلید منقضی شده است یا نه پس روی این موضوع می تونیم حمله بکنیم

یک استاندارد مشخصی اینجا نداریم که بگیم همچین سازوکاری داریم ولی مدل های اعتماد زیادی داریم که سه تاشو مطرح میکنیم--> بحث همون CA است که این CA از کجا میاد

# PKI Trust Models

## □ Monopoly model

- One universally trusted organization is the CA for the known universe
- Big problems if CA is ever compromised
- Who will act as CA ???
  - System is useless if you don't trust the CA!

-  
یک مدل‌مون **monopoly** است

این مدل یک CA جهانی داره و همه بهش اعتماد دارند و همه میشناسشن و مسئله این است  
اگر این CA یک روزی تسخیر شد چی کار باید بکنیم؟ ینی کی جاشو بگیره و اصلا ما از  
کجا بفهمیم این CA از دست رفت این مسئله خیلی مهمیه که توی این مدل داریم

# PKI Trust Models

## ❑ Oligarchy

- Multiple (as in, "a few") trusted CAs
- This approach is used in browsers today
- Browser may have 80 or more CA certificates, just to verify certificates!
- User can decide which CA or CAs to trust

مدل الیگارشی یا ینی ما یک تعداد محدودی ثروتمند داشته باشیم که به بقیه دارند حکومت می‌کنند و مدل الیگارشی هم همین طور است ینی ما تعداد محدودی CA داریم که اینا دارند به بقیه حکومت می‌کنند

معمولًا هم سلسله مراتبی است ینی ما یک CA داریم که به بقیه CA ها certificate میده و او نا به بقیه می‌دن و به همین صورت ادامه می‌یابه توی مرورگرا از این حالت استفاده می‌شه

خودمون تصمیم می‌گیریم که به کدام CA یا CA ها اعتماد کنیم و از کی certificate قبول می‌کنیم که اینو توی اپلیکیشن های مختلف دیدیم که مثلاً می‌گه ما این certificate رو قبول می‌کنیم مثل همین مرورگرا که وقتی این سایت باز می‌شه مثلاً می‌گه این سایت ایمین نیست چون اون مرورگره این certificate که بهش داده رو قبول نداره و بهش اعتماد نداره

# PKI Trust Models

- Anarchy model
  - Everyone is a CA...
  - Users must decide who to trust
  - This approach used in PGP: "Web of trust"
- Why is it anarchy?
  - Suppose certificate is signed by Frank and you don't know Frank, but you do trust Bob and Bob says Alice is trustworthy and Alice vouches for Frank. Should you accept the certificate?
- Many other trust models/PKI issues

- مدل انارشی یا بی نظمی

توی این حالت همه برای خودشون certificate صادر میکنند و اتفاقی که می افته اینه که کاربران تصمیم میگیرند که به این Certificate اعتماد بکنند یا نه اینجا حالت زنجیره ای داریم مثلا ما به دنی اعتماد نداریم ولی به حوری اعتماد داریم و حوری، پرتو رو قبول داره و پرتو هم نساج رو قبول داره و نساج هم دنی رو قبول داره پس ما هم تهش دنی رو قبول میکنیم

# Confidentiality in the Real World

---

# Symmetric Key vs Public Key

- Symmetric key +'s
  - **Speed**
  - No public key infrastructure (PKI) needed  
(but have to generate/distribute keys)
- Public Key +'s
  - **Signatures** (non-repudiation)
  - No **shared secret** (but, do have to get private keys to the right user...)

توی بحث کلید متقارن اون چیزی که نقطه قوت است سرعت است چون سرعت خیلی بالا می باشد مثل AES چون توی سخت افزار است و راحت به ما جواب میده و سرعت توی بحث کلید متقارن مزیت خیلی مهمی است

اینجا نیازی به زیرساخت اینا نداریم و داستان اعتماد اینا اینجا نیاز نیست

توی کلید عمومی امضا رو داریم که خیلی مزیت مهمی است چون بحث عدم انکارو میاره و توی اپلیکیشن ها خیلی لازمش داریم و اینجا نیازی نیست که قبل برای اینکه مثلا بخوایم یک ارتباطی رو شروع بکنیم قبلش یه جوری از طریق کانال امن رسونده باشم به طرف مقابل به همچین چیزی نیاز نداریم

# Notation Reminder

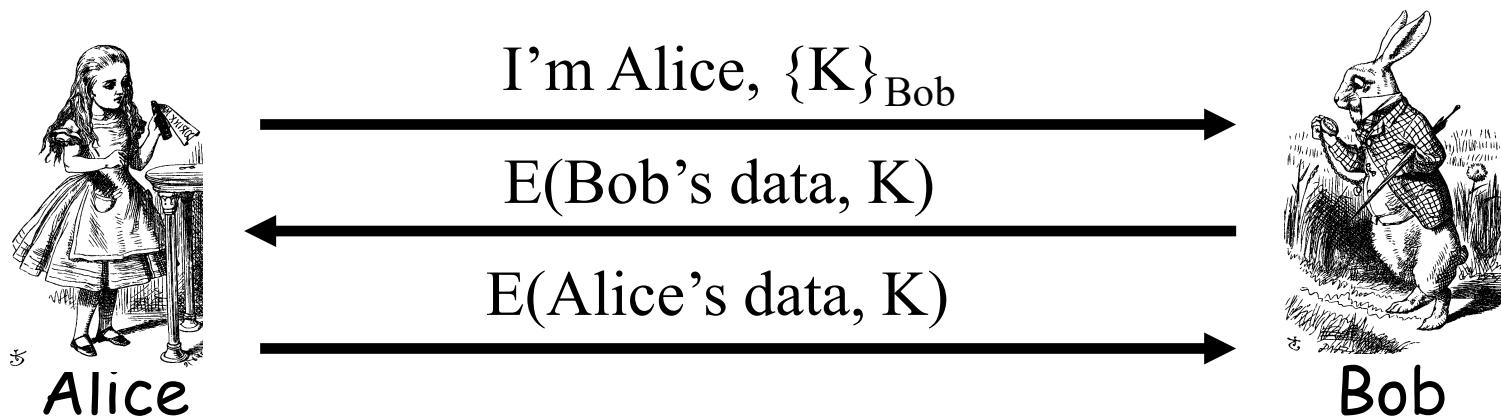
- Public key notation
  - Sign M with Alice's **private key**  
 $[M]_{\text{Alice}}$
  - Encrypt M with Alice's **public key**  
 $\{M\}_{\text{Alice}}$
- Symmetric key notation
  - Encrypt P with **symmetric key** K  
 $C = E(P, K)$
  - Decrypt C with **symmetric key** K  
 $P = D(C, K)$

---

# Real World Confidentiality

## □ Hybrid cryptosystem

- Public key crypto to establish a key
- Symmetric key crypto to encrypt data...



## □ Can Bob be sure he's talking to Alice?

- سیستم رمز ترکیبی:

الیس میخواهد با باب پیام مبادله کنند و برای اینکه بحث سرعت رو داشته باشه بهتره از الگوریتم رمز کلید متقارن استفاده بکنه

قبلش میخواهد کلیدشو به اشتراک بذاره پس از کلید عمومی استفاده میکنه

پس توی شکل میگه من الیسم و این کلیدمونو و با کلید عمومی باب رمز میکنه، کلید رسید به دست باب مسئله ای که داشتیم توی رمزهای متقارن کلید بود که قبلش باید دو طرف کلید رو به اشتراک می ذاشتند اینجا کلید به دست باب و پیامشو با کلید رمز میکنه و میفرسته برای الیس و الیس هم همینطور اشکالش: از کجا می دونه باب، که الیس واقعا خود همون الیس است --> توی پروتکل ها میگیم اینو

# Chapter 5: Hash Functions++

“I'm sure [my memory] only works one way.” Alice remarked.

“I can't remember things before they happen.”

“It's a poor sort of memory that only works backwards,”  
the Queen remarked.

“What sort of things do you remember best?” Alice ventured to ask.

“Oh, things that happened the week after next,”  
the Queen replied in a careless tone.

— Lewis Carroll, *Through the Looking Glass*

# Chapter 5: Hash Functions++

A boat, beneath a sunny sky  
Lingering onward dreamily  
    In an evening of July —  
Children three that nestle near,  
    Eager eye and willing ear,

...

— Lewis Carroll, *Through the Looking Glass*

# Hash Function Motivation

- Suppose Alice signs  $M$ 
  - Alice sends  $M$  and  $S = [M]_{\text{Alice}}$  to Bob
  - Bob verifies that  $M = \{S\}_{\text{Alice}}$
  - Can Alice just send  $S$ ?
- If  $M$  is big,  $[M]_{\text{Alice}}$  costly to **compute & send**
- Suppose instead, Alice signs  $h(M)$ , where  $h(M)$  is a much smaller “fingerprint” of  $M$ 
  - Alice sends  $M$  and  $S = [h(M)]_{\text{Alice}}$  to Bob
  - Bob verifies that  $h(M) = \{S\}_{\text{Alice}}$

الیس میخواست یک پیامی رو امضا کنه و بفرسته برای باب  
حالا باب باید **verifiy** بکنه اینجا که اون  $M$  با رمزشده اون امضا یکی است یا نه  
و سوال اینه الیس فقط می تونه  $S$  رو بفرسته؟

بله میشه ولی اگر  $M$  خیلی بزرگ شد و اینکه بیایم این امضا رو حساب کنیم این کار از نظر محاسباتی  
سنگینه و هم اینکه ارسالش هم هزینه بر است پس به جاش الیس ببیاد یه چیزی رو امضا کنه مثل  $(h(M))$   
که  $(h(M))$  یه جوابی اثر انگشت  $M$  باشه یعنی خاص  $M$  باشه و ببیاد  $M$  رو بفرسته و امضای که روی  
اون اثر انگشت زده

این  $h$  رو باید باب هم بتونه حساب کنه پس اینجا یک تابعی داریم که حساب کردنش برای دو طرف راحته  
و اینکه کلید دار است یا نه رو بعدا می بینیم و بعد میاد  $(h(M))$  رو امضا میکنه  
ویژگی های  $(h(M))$ :

باید کوچک باشه  
تابع یک طرفه است چون فشرده میشه وقتی که کوچیکش کردیم پس چون از یه تعداد بیت بزرگی رفتیم  
به تعداد بیت کوچکتر میشه یه طرفه و برگشت پذیر نیست ولی برای این موضوع صادق است و همیشه  
این موضوع صادق نیست

به ازای پیام های مختلف هش های مختلفی داریم  
محاسباتش راحتراست و سبک است--> برای هر دو طرف میشه : باب هم که بخواست **verifiy** کنه  $(h(M))$   
به دست میاد و مقایسه میکنه با رمز  $S$  که الیس فرستاده

# Hash Function Motivation

- So, Alice signs  $h(M)$ 
  - That is, Alice computes  $S = [h(M)]_{\text{Alice}}$
  - Alice then sends  $(M, S)$  to Bob
  - Bob verifies that  $h(M) = \{S\}_{\text{Alice}}$
- What properties must  $h(M)$  satisfy?
  - Suppose Trudy finds  $M'$  so that  $h(M) = h(M')$
  - Then Trudy can replace  $(M, S)$  with  $(M', S)$
- Does Bob detect this tampering?
  - No, since  $h(M') = h(M) = \{S\}_{\text{Alice}}$

این پیامی که اینجا الیس می فرسته بینی  $M$  می تونه رمز شده باشه  
 $(h(M))$  باید چه ویژگی هایی داشته باشد؟

فرض کنید  $trudy$  بتونه یک پیامی رو پیدا کنه که  $h(M)$  او ن هم با  $h(M')$  الیس یکی باشه  
می تونه اینو جا بزنه بینی به جای اینکه  $M, S$  بفرسته میاد  $M', S'$  می فرسته --> حمله MitM  
و اون بحث رمزنگاری اینا بخاطر همینه که  $trudy$  بتونه همچین کاری بکنه  
یکی از ویژگی های مهمی که  $h$  باید برای ما برآورده بکنه اینه که ما نتونیم راحت یک  $M'$   
پیدا بکنیم که بتونیم  $h$  اش با  $(M)$  یکی باشه --> قاعدها یک همچین چیزی وجود داره چون  
ما داریم فضایی که ازش پیام ها رو انتخاب میکنیم یک فضای بی نهایتی است و از یک طرف  
گفتیم کوچیکش میکنیم فضارو --> جلوتر میگیم که اگر فضارو کوچیک کردیم چی کار کنیم  
که  $trudy$  بتونه همچین کاری بکنه

# Crypto Hash Function

- Crypto hash function  $h(x)$  must provide
  - **Compression** — output length is small
  - **Efficiency** —  $h(x)$  easy to compute for any  $x$
  - **One-way** — given a value  $y$  it is infeasible to find an  $x$  such that  $h(x) = y$
  - **Weak collision resistance** — given  $x$  and  $h(x)$ , infeasible to find  $y \neq x$  such that  $h(y) = h(x)$
  - **Strong collision resistance** — infeasible to find **any**  $x$  and  $y$ , with  $x \neq y$  such that  $h(x) = h(y)$
- Lots of collisions exist, but hard to find **any**

-تابع  $x(h)$  چه چیز ای باید داشته باشه:

-1- فشردگی

-2- efficiency ینی می خواستیم از شر محاسبات سنگین امضا روی یک پیام بزرگ خلاص بشیم

-3- یک طرفه بودن ینی اگر  $y$  داشتیم که اگر  $y$  با  $x(h)$  برابر بود راحت نتونیم بهش برسیم

-4- مقاومت در برابر برخورد ضعیف ینی اگر ما یک  $x$  داشتیم که  $x(h)$  نظریش هم داریم و نتونیم  $u$  رو پیدا کنیم که با  $x$  برابر نباشه ولی  $h$  هاشون با هم برابر باشد

-5- مقاومت در برابر برخورد قوی ینی کلا  $y, x$  کلا نتونیم همینطوری پیدا بکنیم که  $x(h)$  با  $y$  با هم برابر باشد --> اینجا یک  $x$  مشخصی نداریم و موضوع برای هر  $y, x$  که پیدا کردیم هست

فرق 4 و 5 در این است که ؟؟ نفهمیدم دو دقیقه اخر فیلم ؟؟

نکته: حتما collision رو داریم ولی پیدا کردنش سخته

# Pre-Birthday Problem

- Suppose  $N$  people in a room
- How large must  $N$  be before the probability someone has same birthday as me is  $\geq 1/2$  ?
  - Solve:  $1/2 = 1 - (364/365)^N$  for  $N$
  - We find  $N = 253$

- فرض کنید  $n$  نفر توی یک اتاق هستند  
چیزی که میخوایم پیدا کنیم اینه که چقدر باید چیز باشه که ما یکی رو پیدا کنیم که تولدش مثل  
ما باشه؟

بن مثل Weak collision resistance است چون ما یک مقدار مشخصی داریم و میخوایم  
بدونیم که ایا کسی هست که شبیه تاریخ تولد ما باشه یا نه؟  
مقدار Weak collision resistance محتمل است  
 $n$  رو چگونه حساب کنیم؟

$n$  ینی احتمال اینکه یک نفری همون تولد مارو داشته باشه؟  
 $n = \frac{364}{365}$  احتمال داره که اون افراد تاریخ تولدشون مثل ما نباشه بعد این عدد به توان همون  
 $n$  نفر میشه کسایی که تاریخ تولدشون مثل ما نیست و چون می خوایم یه نفرو پیدا کنیم که مثل  
ما است پس 1 منهای مقدار بالایی میکنیم  
ما میخواستیم که احتمالمنون بیشتر از  $\frac{1}{2}$  باشه و اسه همین کل معادله بالا رو برابر با  $\frac{1}{2}$  می  
ذاریم و حلش میکنیم و  $n$  رو بدست میاریم  
پس حداقل 253 نفر باید توی اتاق باشند که ما این احتمال رو بدیم که یکی پیدا بشه که حداقل  
تولدش با ما توی یه روز باشه  
این هنوز مسئله تولد نیست

# Birthday Problem

- How many people must be in a room before probability is  $\geq 1/2$  that any two (or more) have same birthday?
  - $1 - 365/365 \cdot 364/365 \cdots (365-N+1)/365$
  - Set equal to  $1/2$  and solve: **N = 23**
- Surprising? A paradox?
- Maybe not: “Should be” about  $\sqrt{365}$  since we compare all **pairs** x and y
  - And there are 365 possible birthdays

این مسئله تولد است که میگه توی یک اتاق چند نفر باشند که احتمال اینکه دو نفر شانسی تولدشون توی یه روز باشه بیشتر از  $1/2$  باشه

این مثل **Strong collision resistance** است

میخوایم هیچکس با هیچکسی تولدش توی یه روز نباشه:

خب نفر اول میتونه هر روزی باشه پس احتمالش میشه  $365/365$

نفر دوم میشه  $364/365$

و به همین ترتیب می ره جلو

و میشه اینکه هیچکس با هیچکس دیگه روز تولدش یکی نباشه

حالا ما برای اینکه حداقل دو نفر رو پیدا کنیم که روز تولدشون یکی باشه میشه مکمل بالایی

توی مسئله قبل ما یه روز خاص داریم و همه نباید اون یه روز باشند ولی توی این مسئله

میگیم هیچ دونفری مثل هم نباشه

جواب میشه 23 که خیلی کوچیک شد که به این پارادوکس مسئله تولد هم گفته میشه

# Of Hashes and Birthdays

- If  $h(x)$  is  $N$  bits, then  $2^N$  different hash values are possible
- So, if you hash about  $\sqrt{2^N} = 2^{N/2}$  values then you expect to find a collision
- **Implication?** “Exhaustive search” attack...
  - Secure  $N$ -bit hash requires  $2^{N/2}$  work to “break”
  - Recall that secure  $N$ -bit symmetric cipher has work factor of  $2^{N-1}$
- Hash output length vs cipher key length?

- حالا ربطش چیه؟

گفتیم که  $h$  ما  $n$  بیت خروجی تولید میکنه پس فضایی که داریم و اسه خروجی  $h$  هست دو به توان  $n$  ینی ما دو به توان  $n$  مقدار متفاوت برای هش میتوانیم داشته باشیم سوال برای مسئله تولد این بود که **collision** داشته باشم توی مسئله هش فانکشن ها وقتی بخوایم امنیت رو حساب کنیم اون **Strong collision resistance** برای ما مهمه اینکه **collision** داشته باشیم

پس اینجا از دو به توان  $n$  یک رادیکال میگیریم برای اینکه امنیت هش فانکشن رو حساب کنیم مثلا اگر  $n=128$  باشه در بدترین حالت برای مهاجم این میشه که 2 به توان 64 پیدا کنه یک **collision**

# Non-crypto Hash (1)

- Data  $X = (X_1, X_2, X_3, \dots, X_n)$ , each  $X_i$  is a byte
- Define  $h(X) = (X_1 + X_2 + X_3 + \dots + X_n) \text{ mod } 256$
- Is this a secure cryptographic hash?
- Example:  $X = (10101010, 00001111)$
- Hash is  $h(X) = 10111001$
- If  $Y = (00001111, 10101010)$  then  $h(X) = h(Y)$
- Easy to find collisions, so **not** secure...

- هش های غیر رمز:

فرض کنید داده خودمون رو به صورت بایت بایت مشخص میکنیم  
بعدش هش ما میاد اینارو جمع میزنه و مدشو می گیره  
اینجا ورودی میتونه هر چیزی باشه ینی  $n$  مان محدودیت نداره نهايیتا اينه که اگر اخرين بایت  
ما کم داشت ینی داده ما تقسیم پذیر بر مد نبود میایم پد میکنیم

بشردگی داره و کار امد است و یک طرفه هم هست و **collision** هم براش خیلی زیاد پیدا میشه  
بیشترین سختی که داره برای مهاجم 2 به توان 4 است ینی 16 تا مقدار رندوم تولید کنه می  
رسه به **collision**  
8 بیت خروجی داریم و دو به توان  $8/2$  میشه دو به توان 4

# Non-crypto Hash (2)

- Data  $X = (X_0, X_1, X_2, \dots, X_{n-1})$
- Suppose hash is defined as

$$h(X) = (nX_1 + (n-1)X_2 + (n-2)X_3 + \dots + 2 \cdot X_{n-1} + X_n) \bmod 256$$

- Is this a secure cryptographic hash?
- Note that  
$$h(10101010, 00001111) \neq h(00001111, 10101010)$$
- But hash of (00000001, 00001111) is same as hash of (00000000, 00010001)
- Not “secure”, but this hash is used in the (non-crypto) application [rsync](#)

حالت 2:

دیتا مثل همون حالت قبل است و اینجا هم باقی است

و اینجا هم سخترین حالت مثل قبل است که 16 میشد ولی توی این روش پیدا کردن اون چشمی  
که مثلا می تونستیم بفهمیم اینجا دیگه نباشه  
پیدا کردن و اشن سخت نیست **collision**

# Non-crypto Hash (3)

- ❑ Cyclic redundancy check (CRC)
  - ❑ Essentially, CRC is the remainder in a long division calculation
  - ❑ Good for detecting burst **errors**
    - Such errors are unlikely to yield a collision
  - ❑ But easy to **construct** collisions
    - In crypto, Trudy is the enemy, not "random"
  - ❑ CRC has been mistakenly used where crypto integrity check is required (e.g., WEP)

-  
روش 3: CRC بقی مانده تقسیمان است با این تفاوت به جای اینکه منها بشه xor میشه برای این استفاده میکردیم: که توی شبکه که داشتیم استفاده میکردیم اگه نویز داشتیم و خطایی اوmd این نشون بد

مسئله این است که اون خطایی که به وجود میاد براثر نویز یک خطای غیر عمدی است ولی ما می دونیم هوشمند است و collision برای چیزی مثل CRC برای trudy سخت نباشه

# Popular Crypto Hashes

- **MD5** — invented by Rivest (of course...)
  - 128 bit output
  - MD5 collisions easy to find, so it's broken
- **SHA-1** — A U.S. government standard, inner workings similar to MD5
  - 160 bit output
- Many other hashes, but MD5 and SHA-1 are the most widely used
- Hashes work by hashing message in blocks

- هش های رمزی:

بیشترین روش هایی که ازش استفاده شده MD5 و SHA1 میگن اصلا ازش استفاده نکنید چون خیلی سریع collision برash پیدا میشه MD5 128 بیت خروجی و سخترین میشه 2 به توان 64 و خیلی راحت شکسته میشه

MD5 , SHA1 هر دو تاشون شکسته شدند و این باعث شد هی ورژن های بعدی بیاد

# Crypto Hash Design

- Desired property: **avalanche effect**
  - Change to 1 bit of input should affect about half of output bits
- Crypto hash functions consist of some number of rounds
- Want security and speed
  - "Avalanche effect" after few rounds
  - But simple rounds
- Analogous to design of block ciphers

یکی از مهمترین ویژگی هایی که می خوایم هش فانکشن داشته باشند و توی طراحی بهش دقت  
میکنیم اثر بهمنی است و میگه اگر ما یک بیت رو تغییر دادیم این تغییر ما توی خروجی خیلی  
خودشو نشون میده

و معمولا یک تعدادی راند داریم

و هم ازشون امنیت می خوایم و هم سرعت

دلم میخواهد توی دورهای کمتری اون اثر بهمنی اتفاق بیوفته

چرا AES استفاده نمیکنیم و اسه هش فانکشن؟ بحث محاسباتی ینی توی راندها ما خیلی  
عملیات سنگین داشتیم ولی باید راند ها اینجا ساده باشه  
و طراحیشون هم شبیه بلاک سایفر ها است

# SHA-3

- Secure hash algorithm 3 (SHA-3)
  - SHA-1 was a long-time U.S. standard
  - SHA-2 is a family of related algorithms
- SHA-3 developed via a competition
  - Analogous to AES competition
- Prior to SHA-3, most popular hash algorithms were SHA-1 and MD5
  - MD5 is broken by 2000, SHA-1 is similar

-  
توى جزوه اينو نوشتم

# SHA-3

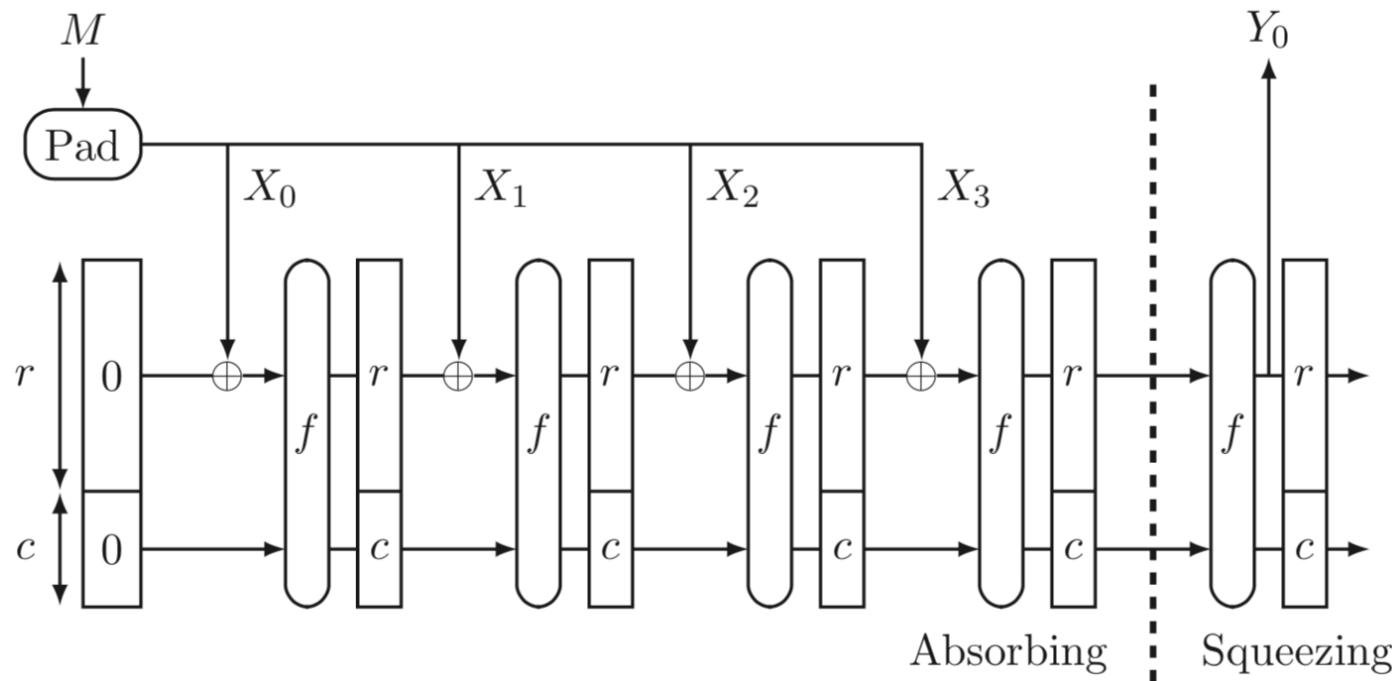
- SHA-3 based on Keccak algorithm
- Prior to SHA-3, most crypto hashes based on Merkle-Damgard method
  - Comparable to Feistel for block cipher
  - Blocks, rounds, etc., but no key
- Keccak also based on blocks & rounds
  - M-D uses compression to combine bits
  - Keccak use a “sponge” construction

# SHA-3 Sponge

- “Absorbing” phase
  - Each new block XORed with permutation of current internal state
- “Squeezing” phase
  - After entire message has been absorbed
  - Extract bits of state for hash value
- Two important parameter
  - Let  $r$  be the “rate” and  $c$  the “capacity”

# SHA-3 Sponge

- Here, assuming message  $M$  is 4 blocks
  - $M = (X_0, X_1, X_2, X_3)$
- Output of SHA-3 is one block  $Y_0$



# SHA-3 Sponge

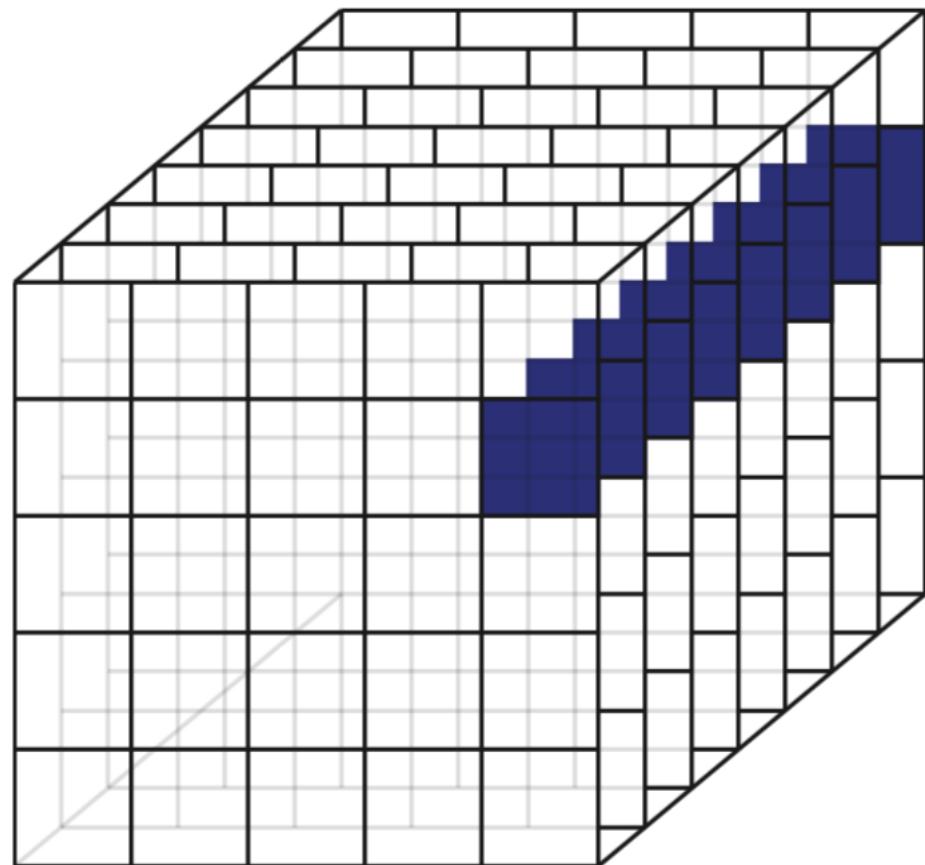
- For SHA-3,  $r + c = 1600$ 
  - $r$  is 1344 or 1088 (so  $c$  is 256 or 512)
- Output length  $y_0$  in {224, 256, 384, 512}
- Function  $f$  is a permutation
- Since  $f$  is a permutation and XOR used for combining, absorbing is reversible
  - But, hash must be one-way
  - So overall, SHA-3 must be one-way

# SHA-3 Inner Workings

- Security and efficiency of SHA-3 depends on choice of function  $f$ 
  - In this sense, like S-boxes in DES
- In SHA-3,  $f$  implemented in 24 rounds
  - Each round, 5 steps denoted  $\theta, \rho, \pi, X, I$
- Recall, block size is 1600
  - For efficiency, internally block is treated as  $5 \times 5$  array of 64-bit words

# SHA-3 Lane

- We have  $5 \times 5$  grid of 64-bit words
- Refer to each  $A[i,j]$  as a “lane”
  - Here lane  $A[4,3]$  is highlighted



# SHA-3 Notation

- XOR:  $\oplus$
- Left cyclic shift: <<<
- Negation: ~
- AND: &
- Specified rotation:  $r[x,y]$
- Specified round constants:  $R_i$

# SHA-3 Round

- Round i
- Recall
  - 24 rounds for each 1600 bit block

```
// θ step
for x = 0 to 4
    C[x] = A[x, 0] ⊕ A[x, 1] ⊕ A[x, 2] ⊕ A[x, 3] ⊕ A[x, 4]
next x
for x = 0 to 4
    D[x] = C[x - 1] ⊕ (C[x + 1] ≪ 1)
next x
for x = 0 to 4
    for y = 0 to 4
        A[x, y] = A[x, y] ⊕ D[x]
    next y
next x
// ρ and π steps
for x = 0 to 4
    for y = 0 to 4
        B[y, 2x + 3y] = (A[x, y] ≪ r[x, y])
    next y
next x
// χ step
for x = 0 to 4
    for y = 0 to 4
        A[x, y] = B[x, y] ⊕ ((~B[x + 1, y]) & B[x + 2, y])
    next y
next x
// υ step
A[0, 0] = A[0, 0] ⊕ Ri
```

# SHA-3 Bottom Line

- ❑ Elegant and efficient design
- ❑ Secure replacement for MD5, SHA-1
  - MD5 broken by 2000
  - SHA-1 similar to MD5, and considered "fully" broken in 2020
- ❑ However, SHA-1 and MD5 still used
- ❑ SHA-3 has not (yet) wildly popular

# HMAC

- Can compute a MAC of the message  $M$  with key  $K$  using a “hashed MAC” or **HMAC**
- HMAC is a **keyed hash**
  - Why would we need a key?
- How to compute HMAC?
- Two obvious choices:  $h(K,M)$  and  $h(M,K)$
- Which is better?

# HMAC

- Should we compute HMAC as  $h(K,M)$  ?
- Hashes computed in blocks
  - $h(B_1, B_2) = F(F(A, B_1), B_2)$  for some  $F$  and constant  $A$
  - Then  $h(B_1, B_2) = F(h(B_1), B_2)$
- Let  $M' = (M, X)$ 
  - Then  $h(K, M') = F(h(K, M), X)$
  - Attacker can compute HMAC of  $M'$  without  $K$
- Is  $h(M, K)$  better?
  - Yes, but... if  $h(M') = h(M)$  then we might have  $h(M, K) = F(h(M), K) = F(h(M'), K) = h(M', K)$

# Correct Way to HMAC

- Described in RFC 2104
- Let  $B$  be the block length of hash, in bytes
  - $B = 64$  for MD5 and SHA-1 and Tiger
- $\text{ipad} = 0x36$  repeated  $B$  times
- $\text{opad} = 0x5C$  repeated  $B$  times
- Then

$$\text{HMAC}(M, K) = h(K \oplus \text{opad}, h(K \oplus \text{ipad}, M))$$

# Hash Uses

- Authentication (HMAC)
- Message integrity (HMAC)
- Message fingerprint
- Data corruption detection
- Digital signature efficiency
- Anything you can do with symmetric crypto
- Also, many, many clever/surprising uses...

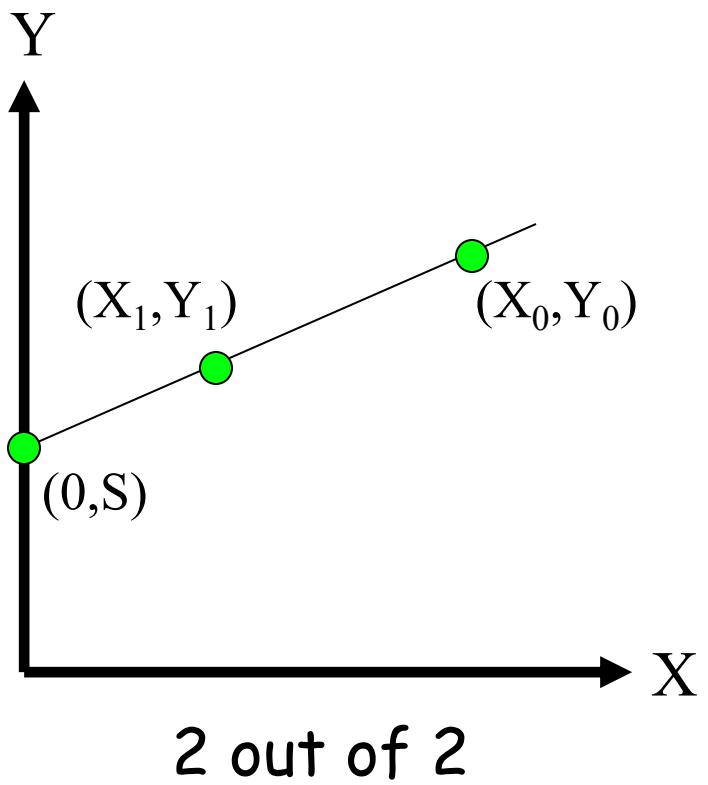
# Online Bids

- Suppose Alice, Bob and Charlie are bidders
- Alice plans to bid A, Bob B and Charlie C
- They don't trust that bids will stay secret
- A possible solution?
  - Alice, Bob, Charlie submit **hashes**  $h(A)$ ,  $h(B)$ ,  $h(C)$
  - All hashes received and posted online
  - Then bids A, B, and C submitted and revealed
- Hashes don't reveal bids (one way)
- Can't change bid after hash sent (collision)
- But there is a serious flaw here...

# Blockchain (separate ppt slides)

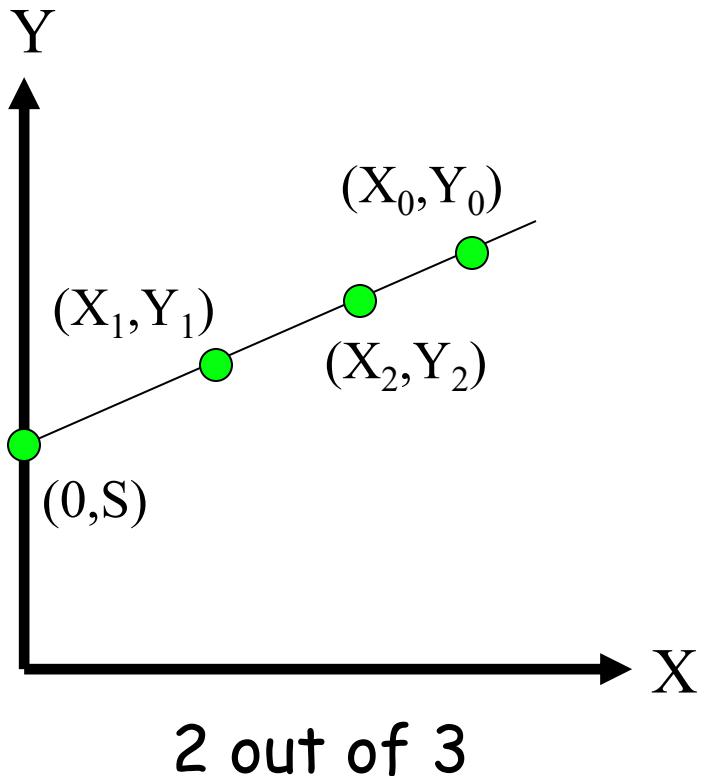
# Secret Sharing

# Shamir's Secret Sharing



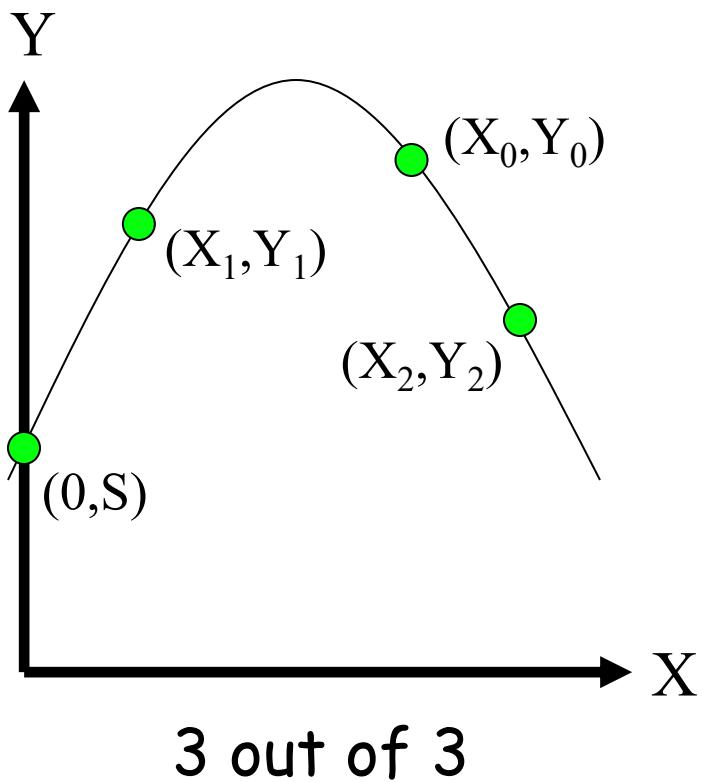
- Two points determine a line
- Give  $(X_0, Y_0)$  to Alice
- Give  $(X_1, Y_1)$  to Bob
- Then Alice and Bob must cooperate to find secret  $S$
- Also works in discrete case
- Easy to make "m out of n" scheme for any  $m \leq n$

# Shamir's Secret Sharing



- Give  $(X_0, Y_0)$  to Alice
- Give  $(X_1, Y_1)$  to Bob
- Give  $(X_2, Y_2)$  to Charlie
- Then any **two** can cooperate to find secret  $S$
- No **one** can determine  $S$
- A "2 out of 3" scheme

# Shamir's Secret Sharing

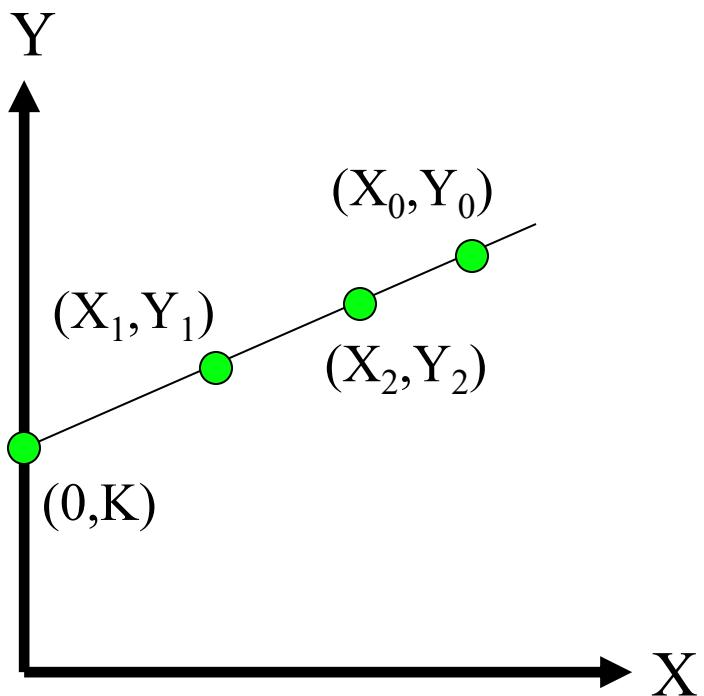


- Give (X<sub>0</sub>, Y<sub>0</sub>) to Alice
- Give (X<sub>1</sub>, Y<sub>1</sub>) to Bob
- Give (X<sub>2</sub>, Y<sub>2</sub>) to Charlie
- 3 pts determine parabola
- Alice, Bob, and Charlie must cooperate to find S
- A "3 out of 3" scheme
- What about "3 out of 4"?

# Secret Sharing Use?

- **Key escrow** — suppose it's required that your key be stored somewhere
- Key can be "recovered" with court order
- But you don't trust FBI to store your keys
- We can use secret sharing
  - Say, three different government agencies
  - Two must cooperate to recover the key

# Secret Sharing Example



- Your symmetric key is K
- Point  $(X_0, Y_0)$  to FBI
- Point  $(X_1, Y_1)$  to DoJ
- Point  $(X_2, Y_2)$  to DoC
- To recover your key K, two of the three agencies must cooperate
- No one agency can get K

# Visual Cryptography

- Another form of secret sharing...
- Alice and Bob want to “share” an image
- Both must cooperate to reveal the image
- Nobody can learn anything about image from Alice’s share or Bob’s share alone
  - That is, both shares are required
- Is this possible?

# Visual Cryptography

- How to “share” a pixel?
- Suppose image is black and white
- Then each pixel is either black or white
- We split pixels as shown

	Pixel	Share 1	Share 2	Overlay
a.				
b.				
c.				
d.				

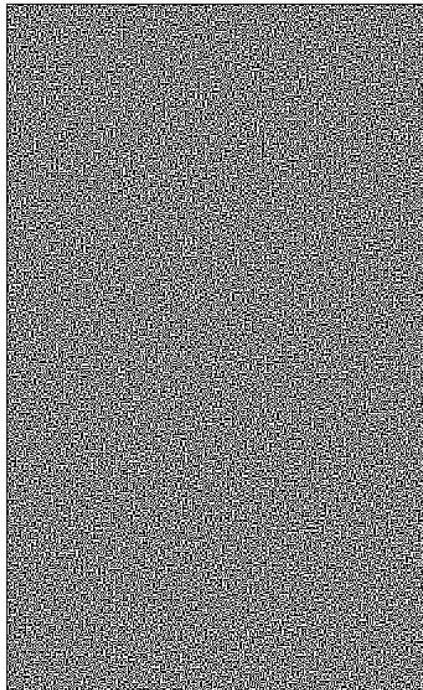
# Sharing Black & White Image

- If pixel is white, randomly choose a or b for Alice's/Bob's shares
- If pixel is black, randomly choose c or d
- **No information in one "share"**

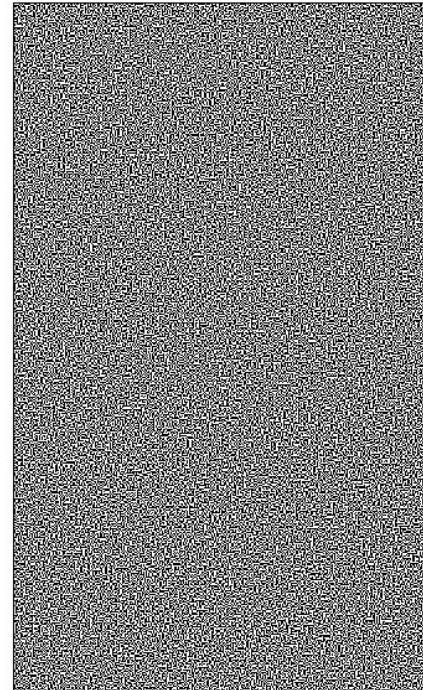
	Pixel	Share 1	Share 2	Overlay
a.				
b.				
c.				
d.				

# Visual Crypto Example

Alice's share



Bob's share



Overlaid shares



# Visual Crypto

- ❑ How does visual “crypto” compare to regular crypto?
- ❑ In visual crypto, no key...
  - Or, maybe both images are the key?
- ❑ With encryption, exhaustive search
  - Except for the one-time pad
- ❑ Exhaustive search on visual crypto?
  - No exhaustive search is possible!

# Visual Crypto

- Visual crypto — no exhaustive search...
- How does visual crypto compare to crypto?
  - Visual crypto is “information theoretically” secure — also true of secret sharing schemes
  - With regular encryption, goal is to make cryptanalysis computationally infeasible
- Visual crypto an example of **secret sharing**
  - Not really a form of crypto, in the usual sense

# Random Numbers in Cryptography

# Random Numbers

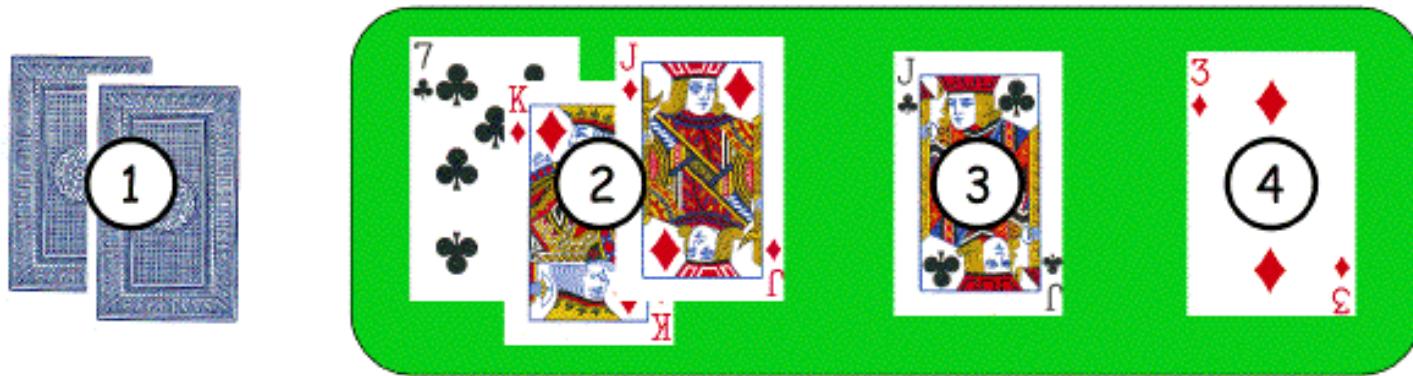
- Random numbers used to generate **keys**
  - Symmetric keys
  - RSA: Prime numbers
  - Diffie Hellman: secret values
- Random numbers used for nonces
  - Sometimes a sequence is OK
  - But sometimes nonces must be random
- Random numbers also used in simulations, statistics, etc.
  - In such apps, need “statistically” random numbers

# Random Numbers

- *Cryptographic random* numbers must be statistically random and **unpredictable**
- Suppose server generates symmetric keys
  - Alice:  $K_A$
  - Bob:  $K_B$
  - Charlie:  $K_C$
  - Dave:  $K_D$
- Alice, Bob, and Charlie don't like Dave...
- Alice, Bob, and Charlie, working together, must not be able to determine  $K_D$

# Non-random Random Numbers

- Online version of Texas Hold 'em Poker
  - ASF Software, Inc.



Player's hand

Community cards in center of the table

- Random numbers used to shuffle the deck
- Program did not produce a random shuffle
- A serious problem, or not?

# Card Shuffle

- There are  $52! > 2^{225}$  possible shuffles
- The poker program used “random” 32-bit integer to determine the shuffle
  - So, only  $2^{32}$  distinct shuffles could occur
- Code used Pascal pseudo-random number generator (PRNG): Randomize()
- Seed value for PRNG was function of number of milliseconds since midnight
- Less than  $2^{27}$  milliseconds in a day
  - So, less than  $2^{27}$  possible shuffles

# Card Shuffle

- Seed based on milliseconds since midnight
- PRNG re-seeded with each shuffle
- By synchronizing clock with server, number of shuffles that need to be tested  $< 2^{18}$
- Could then test all  $2^{18}$  in real time
  - Test each possible shuffle against "up" cards
- Attacker knows **every card** after the first of five rounds of betting!

# Poker Example

- Poker program is an extreme example
  - But common PRNGs are predictable
  - Only a question of how many outputs must be observed before determining the sequence
- Crypto random sequences not predictable
  - For example, keystream from RC4 cipher
  - But "seed" (or key) selection is still an issue!
- How to generate initial **random** values?
  - Keys (and, in some cases, seed values)

# What is Random?

- ❑ True “random” is hard to define
- ❑ **Entropy** is a measure of randomness
- ❑ Good sources of “true” randomness
  - Radioactive decay — but, radioactive computers are not too popular
  - Hardware devices — many good ones on the market
  - Lava lamp — relies on chaotic behavior

# Randomness

- Sources of randomness via software
  - Software is supposed to be deterministic
  - So, must rely on external “random” events
  - Mouse movements, keyboard dynamics, network activity, etc., etc.
- Can get **quality** random bits by such methods
- But **quantity** of bits is very limited
- Bottom line: “The use of pseudo-random processes to generate secret quantities can result in pseudo-security”

# Information Hiding

# Information Hiding

## ❑ Digital Watermarks

- Example: Add “invisible” info to data
- Defense against music/software piracy

## ❑ Steganography

- “Secret” communication channel
- Similar to a **covert channel** (more later)
- Example: Hide data in an image file

# Watermark

- ❑ Add a “mark” to data
- ❑ Visibility (or not) of watermarks
  - Invisible — Watermark is not obvious
  - Visible — Such as **TOP SECRET**
- ❑ Strength (or not) of watermarks
  - Robust — Readable even if attacked
  - Fragile — Damaged if attacked

# Watermark Examples

- Add **robust invisible** mark to digital music
  - If pirated music appears on Internet, can trace it back to original source of the leak
- Add **fragile invisible** mark to audio file
  - If watermark is unreadable, recipient knows that audio has been tampered with (integrity)
- Combinations of several types are sometimes used
  - E.g., visible plus robust invisible watermarks

# Watermark Example (1)

- Non-digital watermark: U.S. currency



- Image embedded in paper on rhs
  - Hold bill to light to see embedded info

# Watermark Example (2)

- ❑ Add **invisible** watermark to photo
- ❑ Claim is that 1 inch<sup>2</sup> contains enough info to reconstruct entire photo
- ❑ If photo is damaged, watermark can be used to reconstruct it!

# Steganography

- According to Herodotus (Greece 440 BC)
  - Shaved slave's head
  - Wrote message on head
  - Let hair grow back
  - Send slave to deliver message
  - Shave slave's head to expose a message warning of Persian invasion
- Historically, steganography used by military more often than cryptography

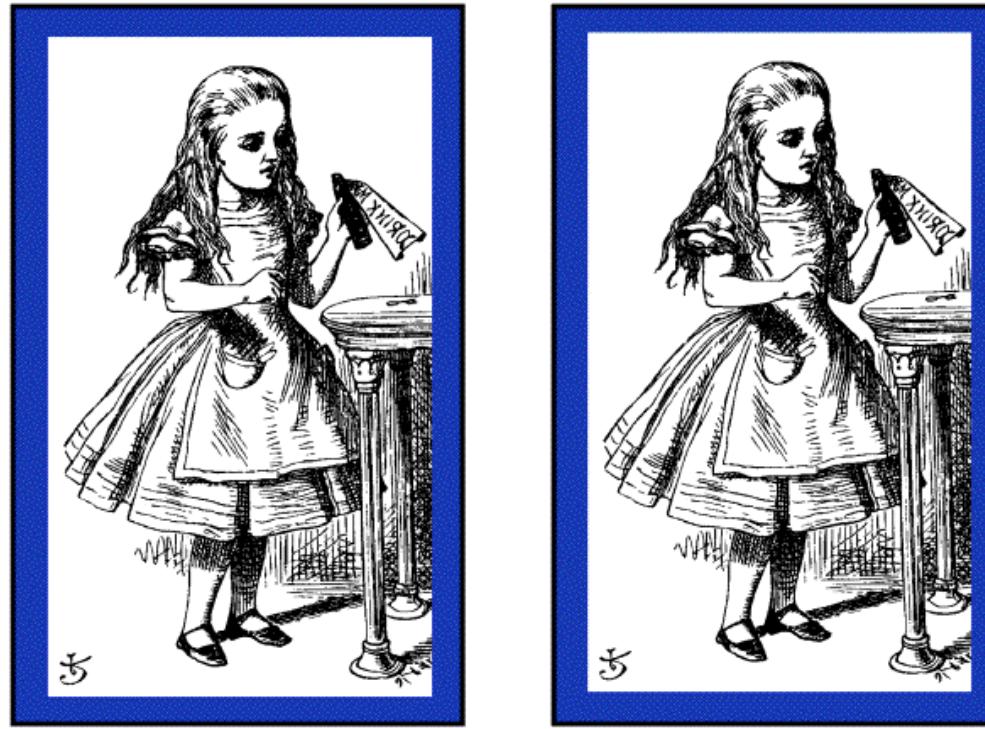
# Images and Steganography

- Images use 24 bits for color: **RGB**
  - 8 bits for **red**, 8 for **green**, 8 for **blue**
- For example
  - **0x7E 0x52 0x90** is **this color**
  - **0xFE 0x52 0x90** is **this color**
- While
  - **0xAB 0x33 0xF0** is **this color**
  - **0xAB 0x33 0xF1** is **this color**
- Low-order bits don't matter...

# Images and Stego

- Given an uncompressed image file...
  - For example, BMP format
- ...we can insert information into low-order RGB bits
- Since low-order RGB bits don't matter, changes will be "invisible" to human eye
  - But, computer program can "see" the bits

# Stego Example 1



- Left side: plain Alice image
- Right side: Alice with entire *Alice in Wonderland* (pdf) "hidden" in the image

# Non-Stego Example

## □ Walrus.html in web browser

"The time has come," the Walrus said,  
"To talk of many things:  
Of shoes and ships and sealing wax  
Of cabbages and kings  
And why the sea is boiling hot  
And whether pigs have wings."

## □ "View source" reveals:

```
<font color="#000000>"The time has come," the Walrus  
    said,</font><br>  
<font color="#000000>"To talk of many things: </font><br>  
<font color="#000000>Of shoes and ships and sealing wax  
    </font><br>  
<font color="#000000>Of cabbages and kings </font><br>  
<font color="#000000>And why the sea is boiling hot  
    </font><br>  
Part 1—Cryptography  
<font color="#000000>And whether pigs have wings."  
    </font><br>
```

# Stego Example 2

- stegoWalrus.html in web browser

"The time has come," the Walrus said,  
"To talk of many things:  
Of shoes and ships and sealing wax  
Of cabbages and kings  
And why the sea is boiling hot  
And whether pigs have wings."

- "View source" reveals:

```
<font color="#000101>"The time has come," the Walrus  
said,</font><br>  
<font color="#000100>"To talk of many things:</font><br>  
<font color="#010000>of shoes and ships and sealing wax  
</font><br>  
<font color="#010000>of cabbages and kings </font><br>  
<font color="#000000>And why the sea is boiling hot  
□ <font color="#000000>"Hidden" message: 011 010 100 100 000 101  
<font color="#010001>And whether pigs have wings."
```

# Steganography

- Some formats (e.g., image files) are more difficult than html for **humans** to read
  - But easy for computer programs to read...
- Easy to hide info in **unimportant bits**
- Easy to damage info in unimportant bits
- To be *robust*, must use **important bits**
  - But stored info must not damage data
  - Collusion attacks are also a concern
- Robust steganography is tricky!

# Information Hiding: The Bottom Line

- Not so easy to hide digital information
  - “Obvious” approach is **not** robust
  - **Stirmark**: tool to make most watermarks in images unreadable without damaging the image
  - Stego/watermarking are active research topics
- If information hiding is suspected
  - Attacker may be able to make information/watermark unreadable
  - Attacker may be able to read the information, given the original document (image, audio, etc.)

# Crypto Summary

- Terminology
- Symmetric key crypto
  - Stream ciphers
    - A5/1 and RC4
  - Block ciphers
    - DES, AES, TEA
    - Modes of operation
    - Integrity

# Crypto Summary

## ❑ Public key crypto

- Knapsack
- RSA
- Diffie-Hellman
- ECC
- Non-repudiation
- PKI, etc.

# Crypto Summary

## ❑ Hashing

- Birthday problem
- SHA-3
- HMAC

## ❑ Blockchain

## ❑ Secret sharing

## ❑ Random numbers

# Crypto Summary

- ❑ Information hiding
  - Steganography
  - Watermarking

# Coming Attractions...

- Access Control
  - Authentication -- who goes there?
  - Authorization -- can you do that?
- We'll see some crypto in next chapter
- We'll see **lots** of crypto in protocol chapters