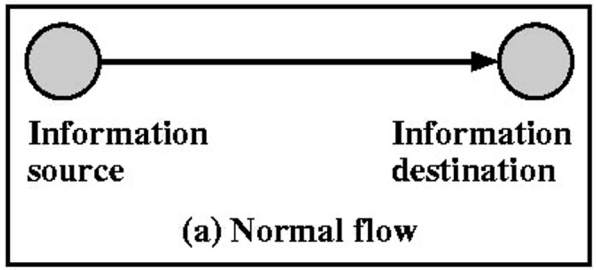
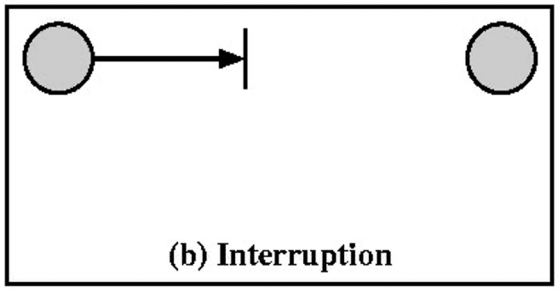
CSE509 : Computer System Security

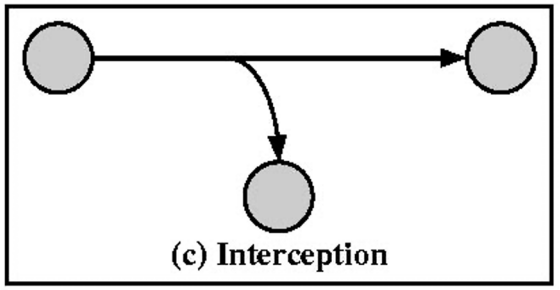
Security

- Communication security
  - security of data channel
  - typical assumption: adversary has access to the physical link over which data is transmitted
  - cryptographic separation is necessary
- System Security
  - security at the end points
  - information cannot be encrypted, as it needs to be accessed by applications on the end system
  - logical separation is typically the basis

Security Concerns

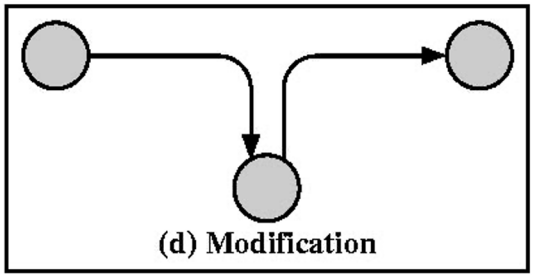


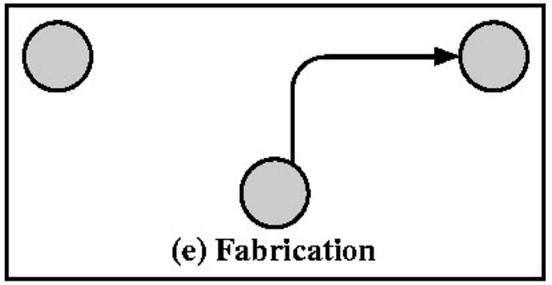




Availability

Confidentiality





Authenticity, integrity

Nonrepudiability

How to achieve security

- Basis is separation
  - Separate adversarial entities
- How to separate adversaries?
  - Physical separation
  - Temporal separation
  - Cryptographic separation
  - Logical separation
- Security vs Functionality
  - Controlled sharing

Cryptography

- Encode the data in a manner that makes it accessible only to authorized parties
  - Encryption algorithm
  - Encryption key
- Why it is not a good idea to rely on secrecy of algorithm
  - Hard to develop good encryption algorithm
  - Does not scale beyond a few users
  - Security by obscurity
- Key point: need to preserve secrecy of key

Crypto

- **Cryptology** — The art and science of making and breaking "secret codes"
- **Cryptography** — making "secret codes"
  - **Encryption** ($E_k(X)$) Vs **Decryption** ($D_k(X)$)
  - **Key** Vs ***Algorithm***
- **Cryptanalysis** — breaking "secret codes" - Discover k, X or both
- **Crypto** — all of the above (and more)

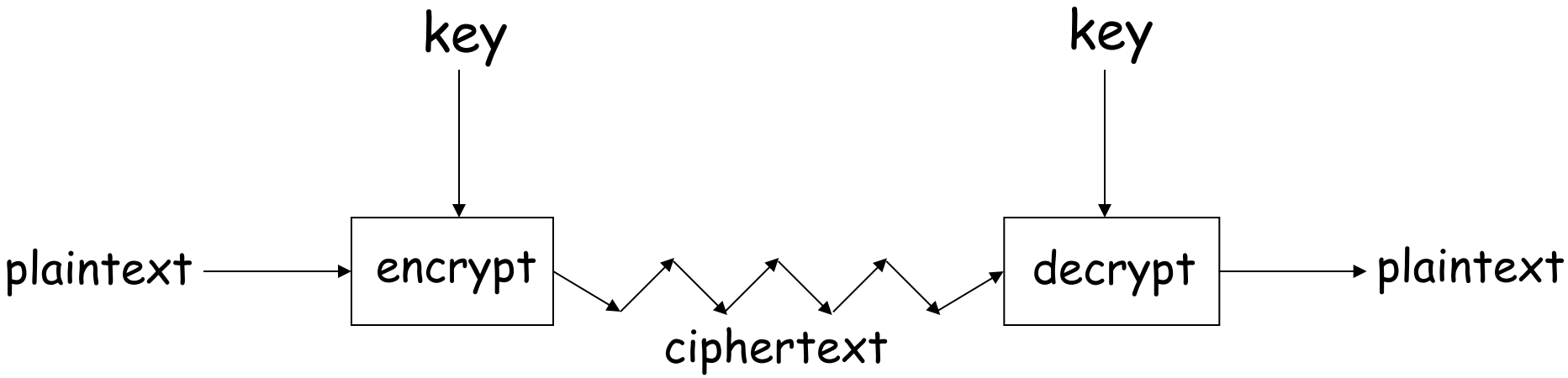
How to Speak Crypto

- 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

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

Crypto as Black Box



A generic view of symmetric key crypto

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:
  IRXUVFRUHDQGVHYHQBHDUVDJR
- Shift by 3 is "Caesar's cipher"

Ceasar'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

Not-so-Simple Substitution

- Shift by n for some $n \in \{0,1,2,\ldots,25\}$
- 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
Ciphertext	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

Cryptanalysis I: Try Them All

- A simple substitution (shift by n) is used
  - But the key is unknown
- Given ciphertext: CSYEVIXIVQMREXIH
- How to find 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 shift of the alphabet
- 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

- Then $26! > 2^{88}$ possible keys

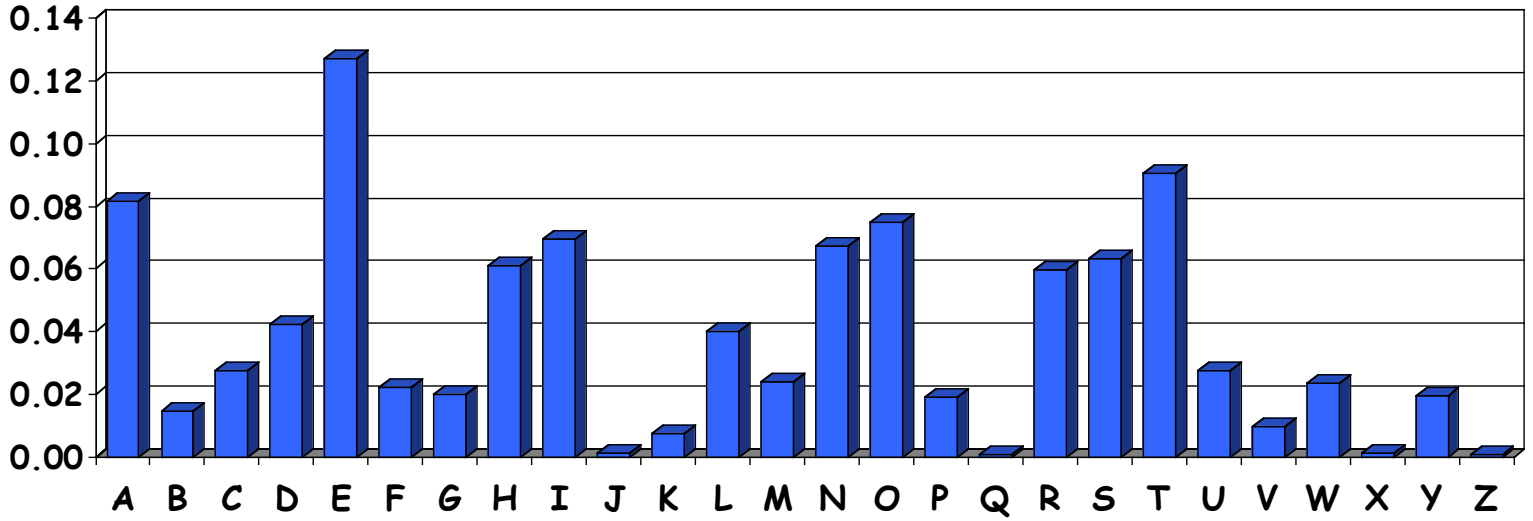
Cryptanalysis II: Be Clever

- We know that a simple substitution is used
- But not necessarily a shift by n
- Find the key given the ciphertext:

PBFPVYFBQXZTYFPBFEQJHDXXQVAPTPQJKTOYQWIPBVWLXTOX
BTFXQWAXBVCXQWAXFQJVWLEQNTOZQGGQLFXQWAKVWLXQ
WAEBIPBFXFQVXGTVJVWLBTPQWAEBFPBFHCVLXBQUFEVWLXGD
PEQVPQGVPPBFTIXPFHXZHVFAGFOTHFEFBQUFTDHZBQPOTHXTY
FTODXQHFTDPTOGHFQPBQWAQJJTODXQHFOQPWTBDHHIXQV
APBFZQHCFWPFHPBFIPBQWKFABVYYDZBOTHPBQPQJTQOTOGHF
QAPBFEQJHDXXQVAVXEBQPEFZBVFOJIWFFACFCCFHQWAUVWF
LQHGFXVAFXQHFUFHILTTAVWAFFAWTEVOITDHFHFQAITIXPFH
XAFQHEFZQWGFLVWPTOFFA

Cryptanalysis II

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



Cryptanalysis II

- Ciphertext:

PBFPVYFBQXZTYFPBFEQJHDXXQVAPTPQJKTOYQWIPBVWLXTOXBTFXQ
WAXBVCXQWAXFQJVWLEQNTOZQGGQLFXQWAKVWLXQWAEBIPBFXFQ
VXGTVJVWLBTPQWAEBFPBFHCVLXBQUFEVWLXGDPEQVPQGVPPBFTIXPFH
XZHVFAGFOTHFEFBQUFTDHZBQPOTHXTYFTODXQHFTDPTOGHFQPBQW
AQJJTODXQHFOQPWTBDHHIXQVAPBFZQHCFWPFHPBFIPBQWKFABVYY
DZBOTHPBQPQJTQOTOGHFQAPBFEQJHDXXQVAVXEBQPEFZBVFOJIWFF
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

What if…

- We encrypted 'Gadsby'? A 50,000 word novel WITHOUT the letter 'e'?
- https://en.wikipedia.org/wiki/Gadsby_(novel)

If Youth, throughout all history, had had a champion to stand up for it; to show a doubting world that a child can think; and, possibly, do it practically; you wouldn't constantly run across folks today who claim that "a child don't know anything." A child's brain starts functioning at birth; and has, amongst its many infant convolutions, thousands of dormant atoms, into which God has put a mystic possibility for noticing an adult's act, and figuring out its purport.

Cryptanalysis: Terminology

- Cryptosystem is **secure** if best know 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 … ?

Type of Attack	Known to Cryptanalyst
Ciphertext only	• Encryption algorithm • Ciphertext to be decoded
Known plaintext	• Encryption algorithm • Ciphertext to be decoded • One or more plaintext-ciphertext pairs formed with the secret key
Chosen plaintext	• Encryption algorithm • Ciphertext to be decoded • Plaintext message chosen by cryptanalyst, together with its corresponding ciphertext generated with the secret key
Chosen ciphertext	• Encryption algorithm • Ciphertext to be decoded • Purported ciphertext chosen by cryptanalyst, together with its corresponding decrypted plaintext generated with the secret key
Chosen text	• Encryption algorithm • Ciphertext to be decoded • Plaintext message chosen by cryptanalyst, together with its corresponding ciphertext generated with the secret key • Purported ciphertext chosen by cryptanalyst, together with its corresponding decrypted plaintext generated with the secret key

Claude Shannon

- The founder of Information Theory
- 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

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

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
  - Forward search (public key crypto)
  - And others…

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"

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

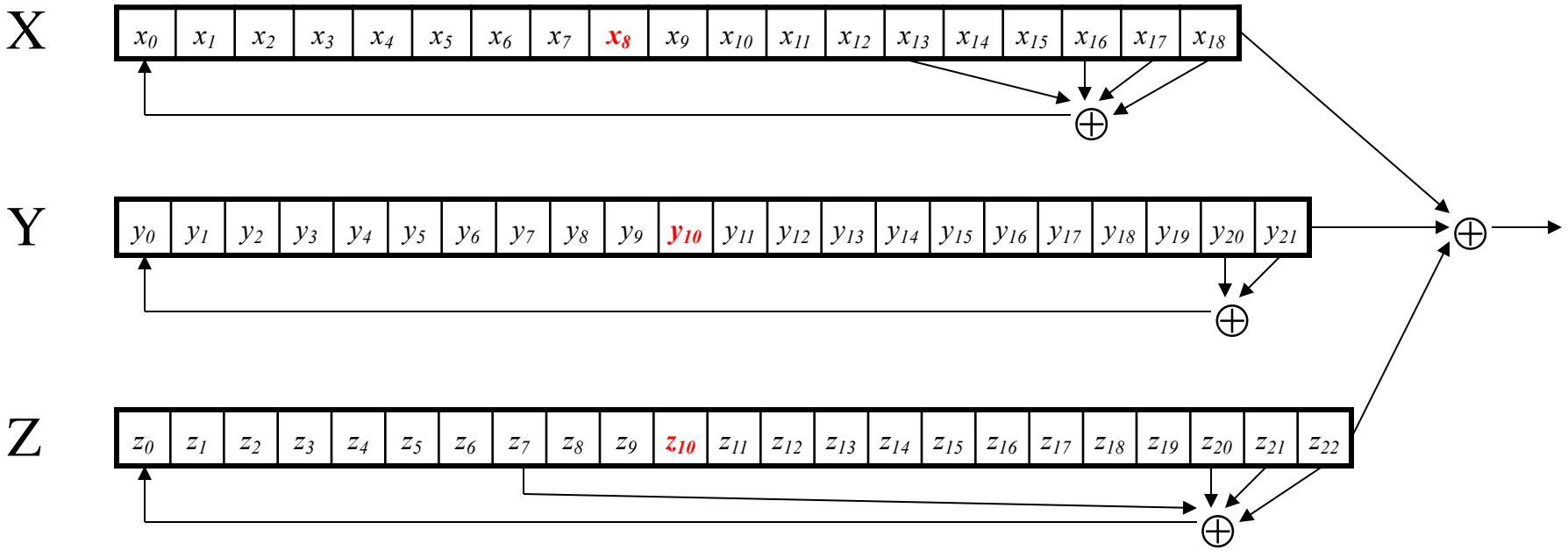
A5/1: Shift Registers

- A5/1 uses 3 *shift registers*
  - X: 19 bits $(x_0,x_1,x_2, \ldots,x_{18})$
  - Y: 22 bits $(y_0,y_1,y_2, \ldots,y_{21})$
  - Z: 23 bits $(z_0,z_1,z_2, \ldots,z_{22})$

A5/1: Keystream

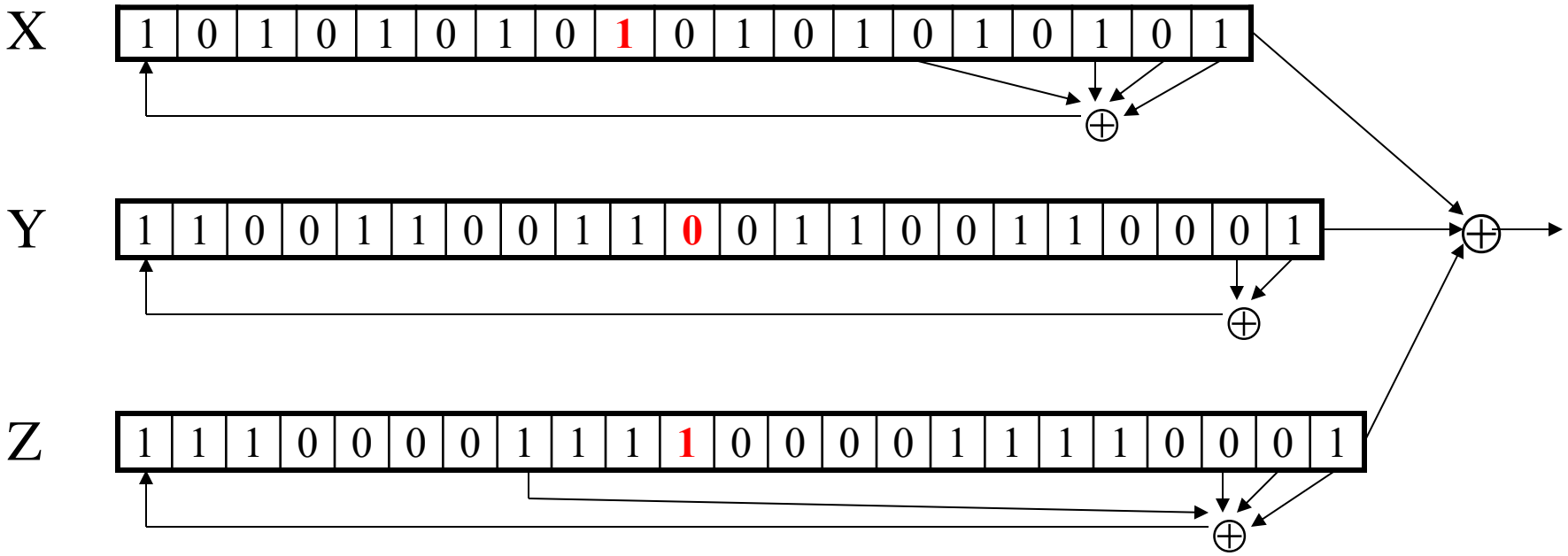
- At each iteration: $m = \text{maj}(x_8, y_{10}, z_{10})$
  - Examples: $\text{maj}(0,1,0) = 0$ and $\text{maj}(1,1,0) = 1$
- If $x_8 = m$ then X *steps*
  - $t = x_{13} \oplus x_{16} \oplus x_{17} \oplus x_{18}$
  - $x_i = x_{i-1}$ for $i = 18,17,\ldots,1$ and $x_0 = t$
- If $y_{10} = m$ then Y *steps*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

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

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

RC4

- A self-modifying lookup table
- Table always contains a permutation of the byte values $0,1,\ldots,255$
- 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

RC4 Initialization

- S[] is permutation of 0,1,...,255
- key[] contains N bytes of key

```
for i = 0 to 255
      S[i] = i
      K[i] = key[i (mod 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
```

RC4 Keystream

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

```
i = (i + 1) mod 256
j = (j + S[i]) mod 256
swap(S[i], S[j])
t = (S[i] + S[j]) mod 256
keystreamByte = S[t]
```

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

Stream Ciphers

- Stream ciphers were popular in the past
  - Efficient in hardware
  - Speed was needed to keep up with voice, 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…

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

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 $i = 1, 2, ..., n$, 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: $C = (L_n, R_n)$

Feistel Cipher: Decryption

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

  $R_{i-1} = L_i$

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

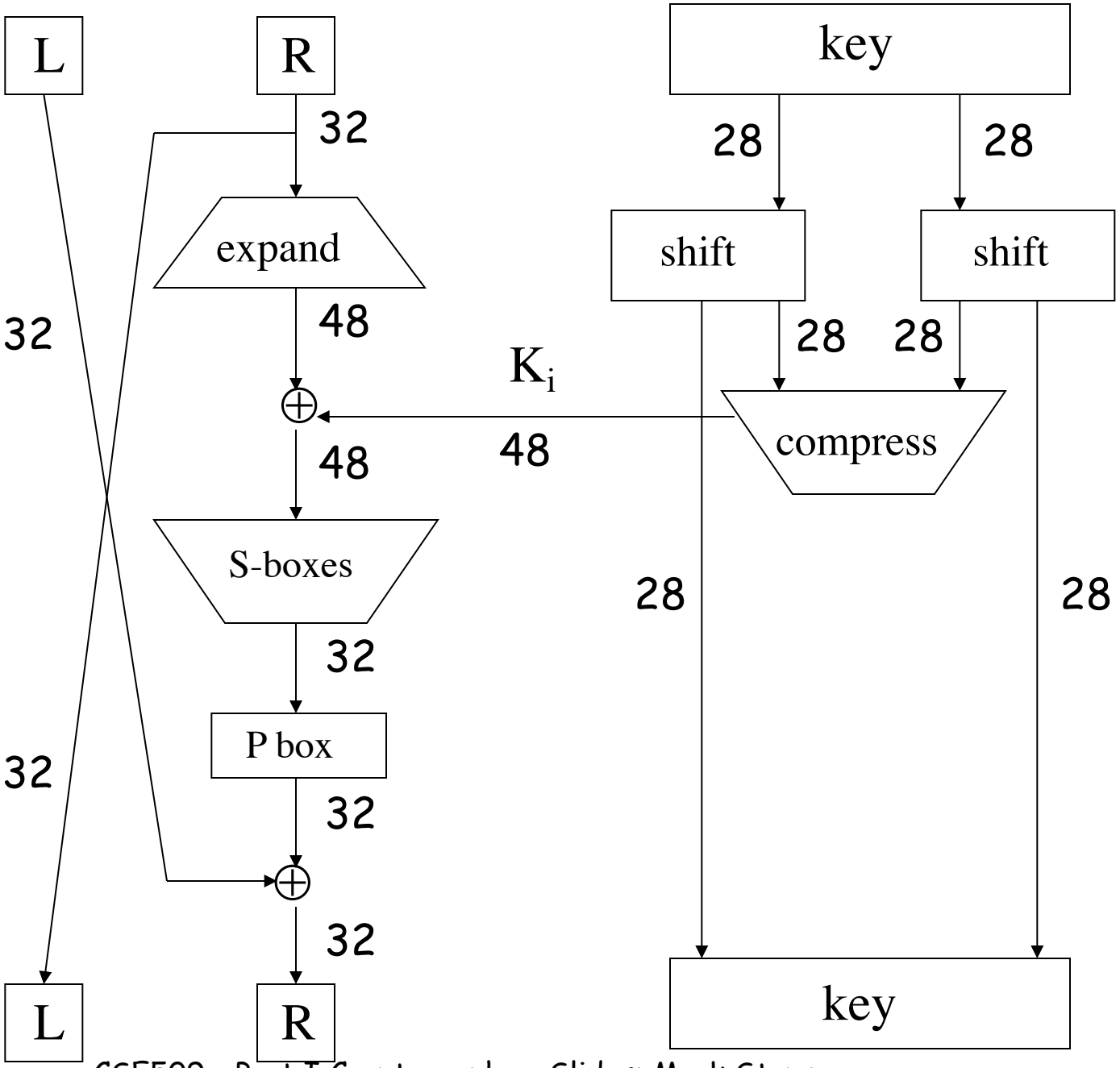
  where F is round function and K_i is subkey
- Plaintext: $P = (L_0, R_0)$
- Decryption works for any function F
  - But only secure for certain functions F

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 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



One Round of DES

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
```

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

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

DES Subkey

- 56 bit DES key, numbered 0,1,2,…,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
```

DES Subkey

- For rounds `i=1,2,...,16`
  - Let $LK = (LK$ circular shift left by $r_i)$
  - Let $RK = (RK$ 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 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

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

Security of DES

- Security depends heavily on S-boxes
  - Everything else in DES is linear
- 35+ years of intense analysis has 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)

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 $K_1 \neq K_2$

Triple DES

- Today, 56 bit DES key is too small
  - Exhaustive key search is feasible
- But DES is 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

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

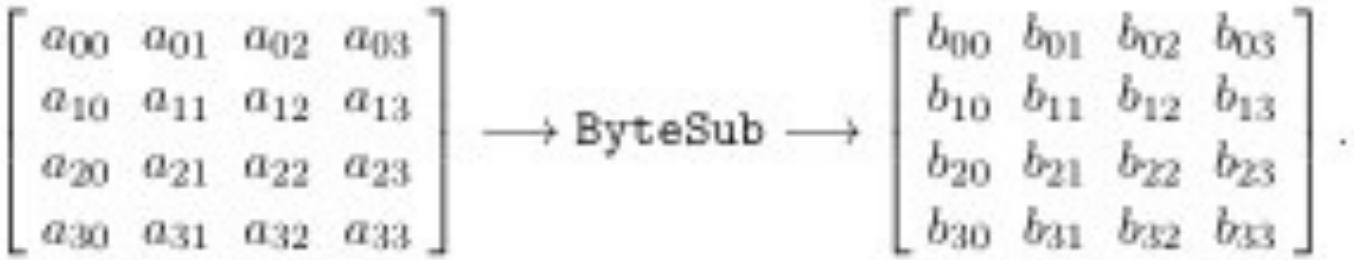
- Replacement for DES
- AES competition (late 90's)
  - NSA openly involved
  - Transparent selection process
  - Many strong algorithms proposed
  - Rijndael Algorithm ultimately selected (pronounced like "Rain Doll" or "Rhine Doll")
- Iterated block cipher (like DES)
- Not a Feistel cipher (unlike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

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)

AES ByteSub

- Treat 128 bit block as 4x4 byte array



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

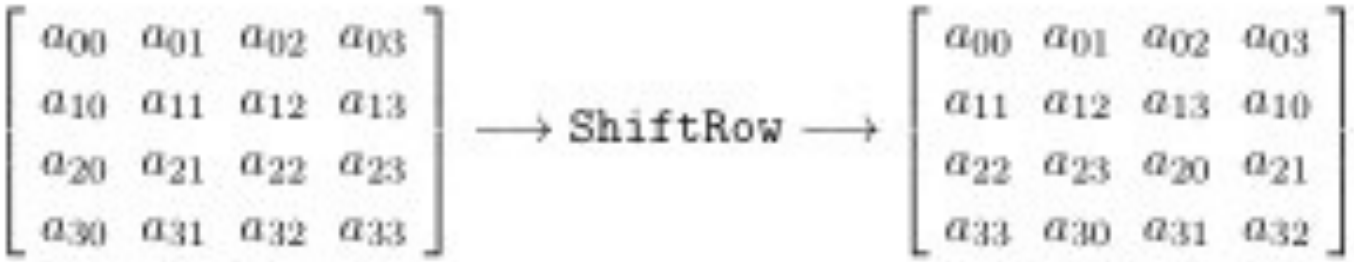
AES "S-box"

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

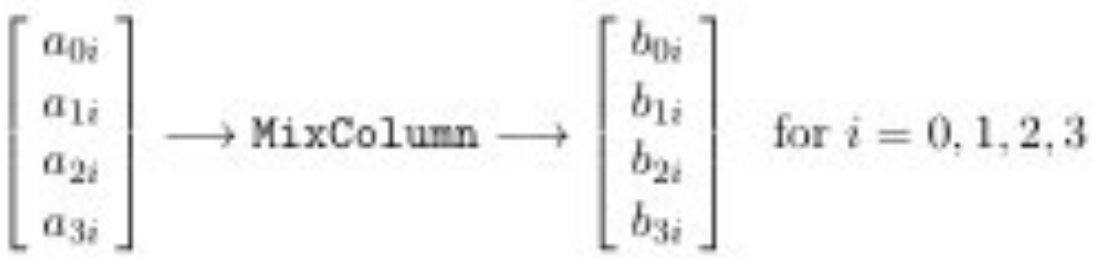
AES ShiftRow

- Cyclic shift rows



AES MixColumn

- Invertible, linear operation applied to each column



- Implemented as a (big) lookup table

AES MixColumn

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \quad \text{for } 0 \le c < \boldsymbol{Nb}. \qquad (5.6)$$

As a result of this multiplication, the four bytes in a column are replaced by the following:

$$s'_{0,c} = (\{02\} \bullet s_{0,c}) \oplus (\{03\} \bullet s_{1,c}) \oplus s_{2,c} \oplus s_{3,c}$$

$$s'_{1,c} = s_{0,c} \oplus (\{02\} \bullet s_{1,c}) \oplus (\{03\} \bullet s_{2,c}) \oplus s_{3,c}$$

$$s'_{2,c} = s_{0,c} \oplus s_{1,c} \oplus (\{02\} \bullet s_{2,c}) \oplus (\{03\} \bullet s_{3,c})$$

$$s'_{3,c} = (\{03\} \bullet s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (\{02\} \bullet s_{3,c}).$$

AES AddRoundKey

- XOR subkey with block

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix} \oplus \begin{bmatrix} k_{00} & k_{01} & k_{02} & k_{03} \\ k_{10} & k_{11} & k_{12} & k_{13} \\ k_{20} & k_{21} & k_{22} & k_{23} \\ k_{30} & k_{31} & k_{32} & k_{33} \end{bmatrix} = \begin{bmatrix} b_{00} & b_{01} & b_{02} & b_{03} \\ b_{10} & b_{11} & b_{12} & b_{13} \\ b_{20} & b_{21} & b_{22} & b_{23} \\ b_{30} & b_{31} & b_{32} & b_{33} \end{bmatrix}$$

Block Subkey

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

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)

A Few Other Block Ciphers

- Briefly…
  - IDEA
  - Blowfish
  - RC6

IDEA

- Invented by James Massey
  - One of the giants of modern crypto
- IDEA has 64-bit block, 128-bit key
- IDEA uses mixed-mode arithmetic
- Combine different math operations
  - IDEA the first to use this approach
  - Frequently used today

Blowfish

- Blowfish encrypts 64-bit blocks
- Key is variable length, up to 448 bits
- Invented by Bruce Schneier
- Almost a Feistel cipher

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

- The round function F uses 4 S-boxes
  - Each S-box maps 8 bits to 32 bits
- Key-dependent S-boxes
  - S-boxes determined by the key

RC6

- Invented by Ron Rivest
- Variables
  - Block size
  - Key size
  - Number of rounds
- An AES finalist
- Uses data dependent rotations
  - Unusual for algorithm to depend on plaintext

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 issue

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

ECB Mode

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

Encrypt	Decrypt
$C_0 = E(P_0, K)$	$P_0 = D(C_0, K)$
$C_1 = E(P_1, K)$	$P_1 = D(C_1, K)$
$C_2 = E(P_2, K) \ \ldots$	$P_2 = D(C_2, K) \ \ldots$

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

ECB Cut and Paste

- Suppose plaintext is

  `Alice digs Bob. Trudy digs Tom.`

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

  $P_0 = \text{“}\texttt{Alice di}\text{”}, P_1 = \text{“}\texttt{gs Bob. }\text{”},$

  $P_2 = \text{“}\texttt{Trudy di}\text{”}, P_3 = \text{“}\texttt{gs Tom. }\text{”}$

- Ciphertext: C_0, C_1, C_2, C_3
- Trudy cuts and pastes: C_0, C_3, C_2, C_1
- Decrypts as

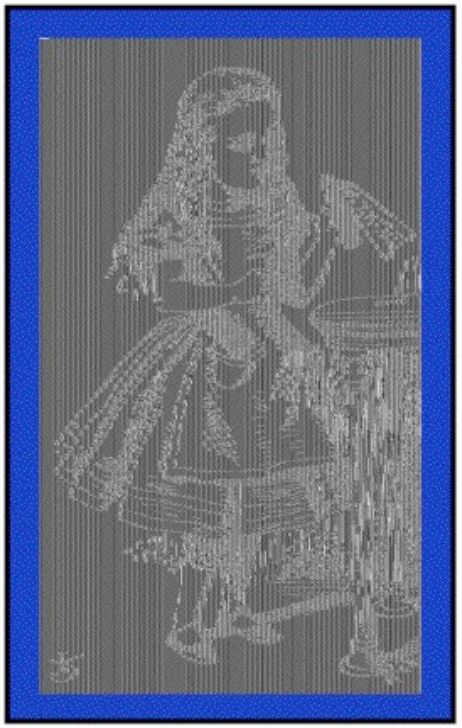
  `Alice digs Tom. Trudy digs Bob.`

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!

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

$$C_0 = E(IV \oplus P_0, K),$$
$$C_1 = E(C_0 \oplus P_1, K),$$
$$C_2 = E(C_1 \oplus P_2, K), \ldots$$

Decryption

$$P_0 = IV \oplus D(C_0, K),$$
$$P_1 = C_0 \oplus D(C_1, K),$$
$$P_2 = C_1 \oplus D(C_2, K), \ldots$$

- Analogous to classic codebook *with additive*

CBC Mode

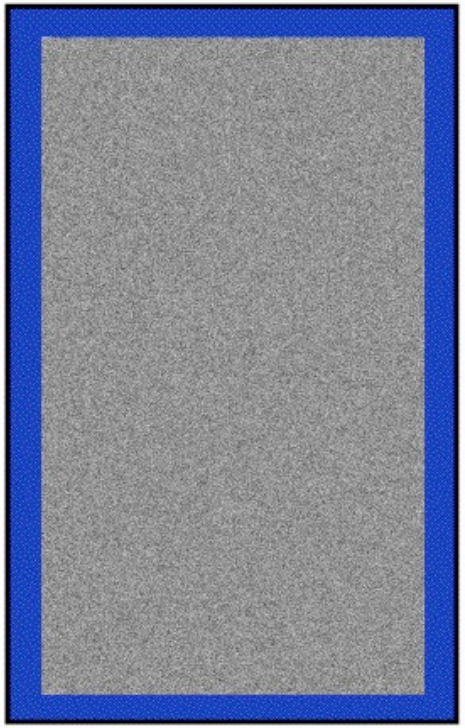
- Identical plaintext blocks yield different ciphertext blocks — this is very good!
- But what about errors in transmission?
  - If C_1 is garbled to, say, G then

    $$P_1 \neq C_0 \oplus D(G, K), P_2 \neq G \oplus D(C_2, K)$$

  - But $P_3 = C_2 \oplus D(C_3, K), P_4 = C_3 \oplus D(C_4, K), \ldots$
  - Automatically recovers from errors!
- Cut and paste is still possible, but more complex (and will cause garbles)

Alice Likes CBC Mode

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



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

Counter Mode (CTR)

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

Encryption

$$C_0 = P_0 \oplus E(IV, K),$$
$$C_1 = P_1 \oplus E(IV+1, K),$$
$$C_2 = P_2 \oplus E(IV+2, K), \ldots$$

Decryption

$$P_0 = C_0 \oplus E(IV, K),$$
$$P_1 = C_1 \oplus E(IV+1, K),$$
$$P_2 = C_2 \oplus E(IV+2, K), \ldots$$

- Note: CBC also works for random access
  - But there is a significant limitation…

Data Integrity

Data Integrity

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

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

MAC Computation

- MAC computation (assuming N blocks)

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

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

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

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

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

Does a MAC work?

- Suppose Alice has 4 plaintext blocks
- Alice computes

  $C_0 = E(IV \oplus P_0, K)$, $C_1 = E(C_0 \oplus P_1, K)$,

  $C_2 = E(C_1 \oplus P_2, K)$, $C_3 = E(C_2 \oplus P_3, K) = MAC$
- Alice sends IV, P_0, P_1, P_2, P_3 and MAC to Bob
- Suppose Trudy changes P_1 to X
- Bob computes

  $C_0 = E(IV \oplus P_0, K)$, $C_1 = E(C_0 \oplus X, K)$,

  $C_2 = E(C_1 \oplus P_2, K)$, $C_3 = E(C_2 \oplus P_3, K) = MAC \neq MAC$
- It works since error propagates into MAC
- Trudy can't make $MAC == MAC$ without K

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

Uses for Symmetric Crypto

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

Public Key Cryptography

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

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…

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 $ed = 1 \bmod (p-1)(q-1)$
- **Public key** is (N,e)
- **Private key** is d

RSA

- Message M is treated as a number
- To encrypt M we compute

  $$C = M^e \bmod N$$

- To decrypt ciphertext C, we compute

  $$M = C^d \bmod N$$

- Recall that e and N are public
- If Trudy can factor $N = pq$, she can use e to easily find d since $ed = 1 \bmod (p-1)(q-1)$
- So, **factoring the modulus breaks RSA**
  - Is factoring the only way to break RSA?

Does RSA Really Work?

- Given $C = M^e \bmod N$ we want to show that
  $M = C^d \bmod N = M^{ed} \bmod N$
- We'll need **Euler's Theorem**:
  If x is relatively prime to n then $x^{\varphi(n)} = 1 \bmod n$
- Facts:
  1) $ed = 1 \bmod (p-1)(q-1)$
  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, $C^d = M^{ed} = M^{(ed-1)+1} = M \cdot M^{ed-1} = M \cdot M^{k\varphi(N)}$
  $= M \cdot (M^{\varphi(N)})^k \bmod N = M \cdot 1^k \bmod N = M \bmod N$

Simple RSA Example

- Example of *textbook* RSA
  - 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 $ed = 1 \bmod 20$
    - We find that $d = 7$ works
- Public key: $(N, e) = (33, 3)$
- Private key: $d = 7$

Simple 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

$$M = C^d \bmod N = 17^7 = 410{,}338{,}673$$
$$= 12{,}434{,}505 * 33 + 8 = 8 \bmod 33$$

More Efficient RSA (1)

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

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 $e = 3$)

Diffie-Hellman

Diffie-Hellman Key Exchange

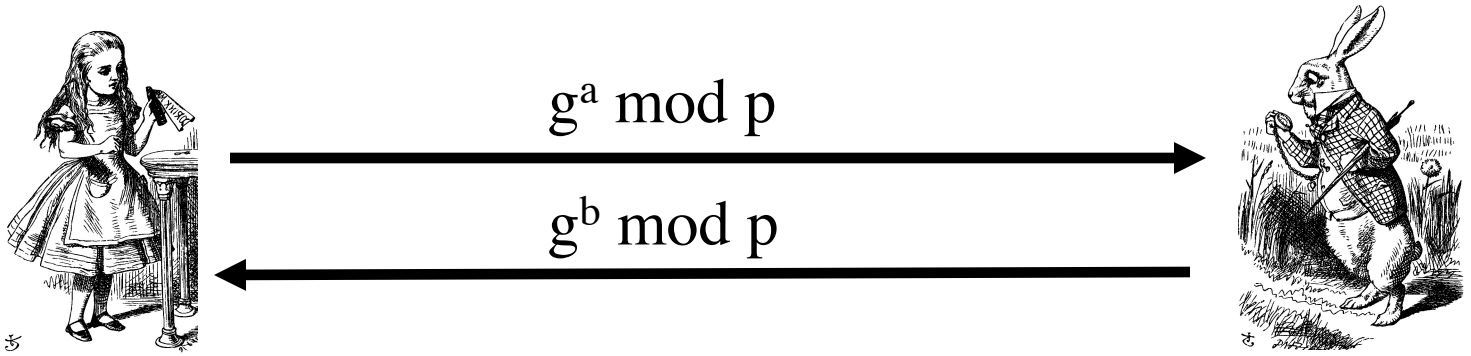
- Invented by Williamson (GCHQ) and, independently, by D and H (Stanford)
- A "key exchange" algorithm
  - Used to establish a shared symmetric key
  - *Not* for encrypting or signing
- Based on discrete log problem
  - Given: g, p, and $g^k \bmod p$
  - Find: exponent k

Diffie-Hellman

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

Diffie-Hellman

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



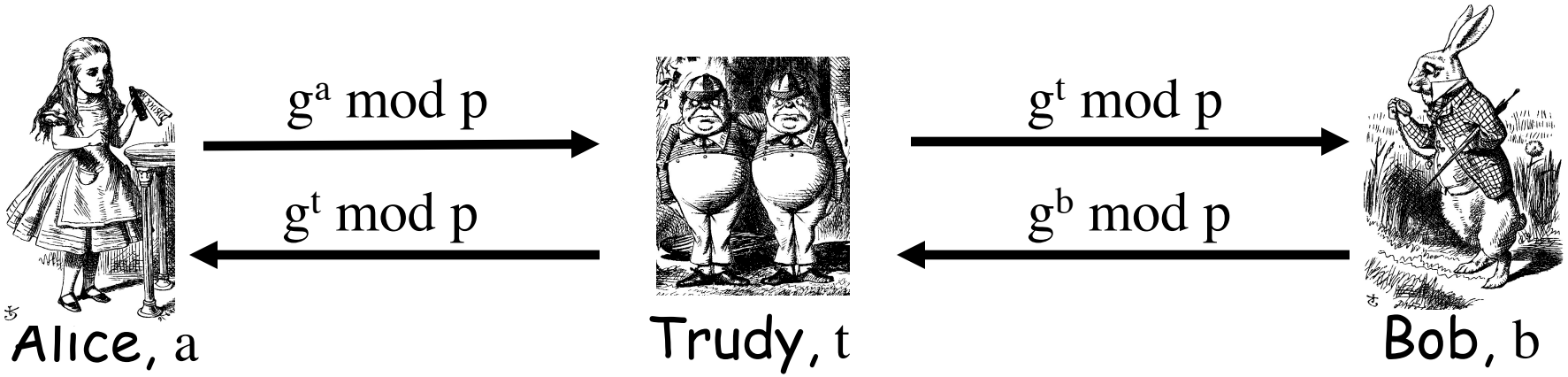
- Alice computes $(g^b)^a = g^{ba} = g^{ab} \bmod p$
- Bob computes $(g^a)^b = g^{ab} \bmod p$
- They can use $K = g^{ab} \bmod p$ as symmetric key

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

Diffie-Hellman

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



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

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

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

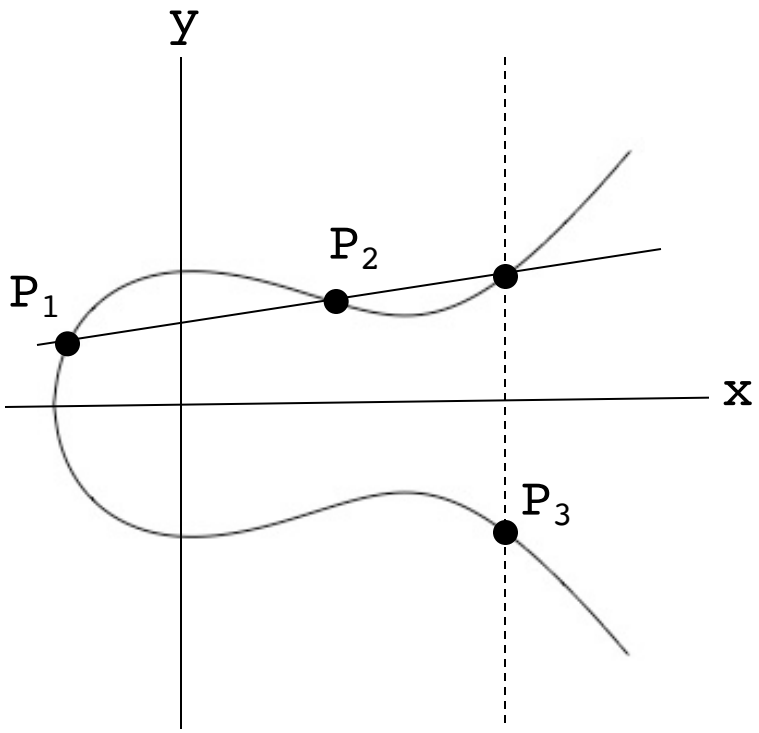
What is an Elliptic Curve?

- 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!

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…

Points on Elliptic Curve

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

  $x = 0 \Rightarrow y^2 = 3 \Rightarrow$ 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

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

  and the point at infinity: ∞

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

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

  Special cases: If m is infinite, $P_3 = \infty$, and

  $\infty + P = P$ for all P

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 ∞
- 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, $(1,4) + (3,1) = (2,0)$

ECC Diffie-Hellman

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

$A(x,y)$

$B(x,y)$

Alice, A Bob, B

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

ECC Diffie-Hellman

- Public: Curve $y^2 = x^3 + 7x + b \pmod{37}$ and point $(2,5) \Rightarrow b = 3$
- Alice's private: $A = 4$
- Bob's private: $B = 7$
- 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: $7(7,32) = (22,1)$

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?

ECC Example

- The following point P is on the curve E

(x,y) = (9733901098705906652315613390893 5, 14967037284616928576068237197889 8)

- Let k = 28118384031160194966820795453068 4
- The kP is given by

(x,y) = (4464676969740586105763086188428 4, 52296809889578588047540374779097)

- 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 in ECC example
  - And 2048 bit RSA modulus is common…

Uses for Public Key Crypto

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

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

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

Public Key Notation

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

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

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

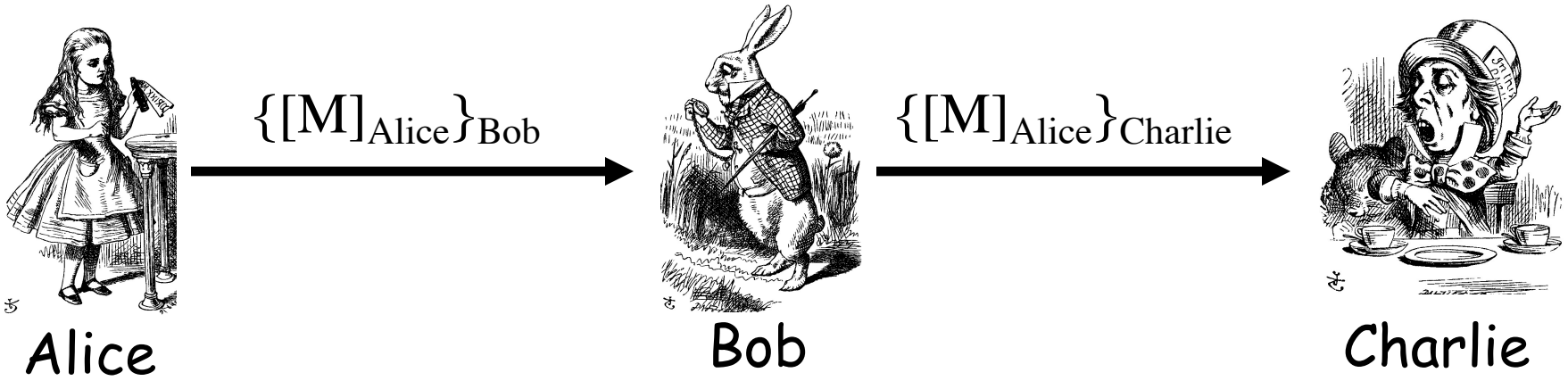
Sign and Encrypt
vs
Encrypt and Sign

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?

Sign and Encrypt

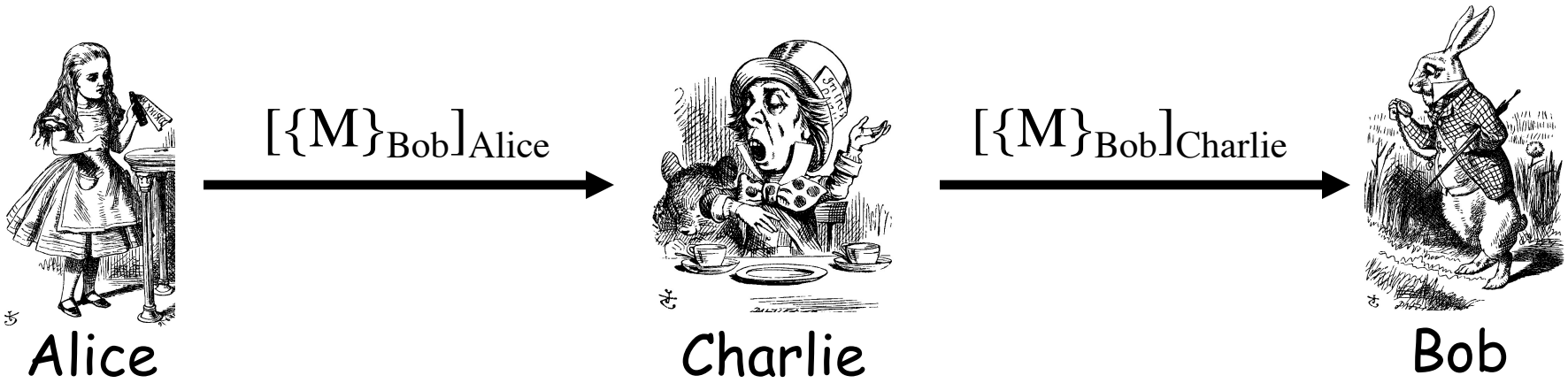
- M = "I love you"



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

Encrypt and Sign

- M = "My theory, which is mine…."



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

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

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

  Alice's Certificate $= (M, S)$
- Signature on certificate is verified using *CA*'s public key

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

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** verify 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

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)

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*!

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

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

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…)

Notation Reminder

- Public key notation
  - Sign M with Alice's private key

    $[M]_{Alice}$
  - Encrypt M with Alice's public key

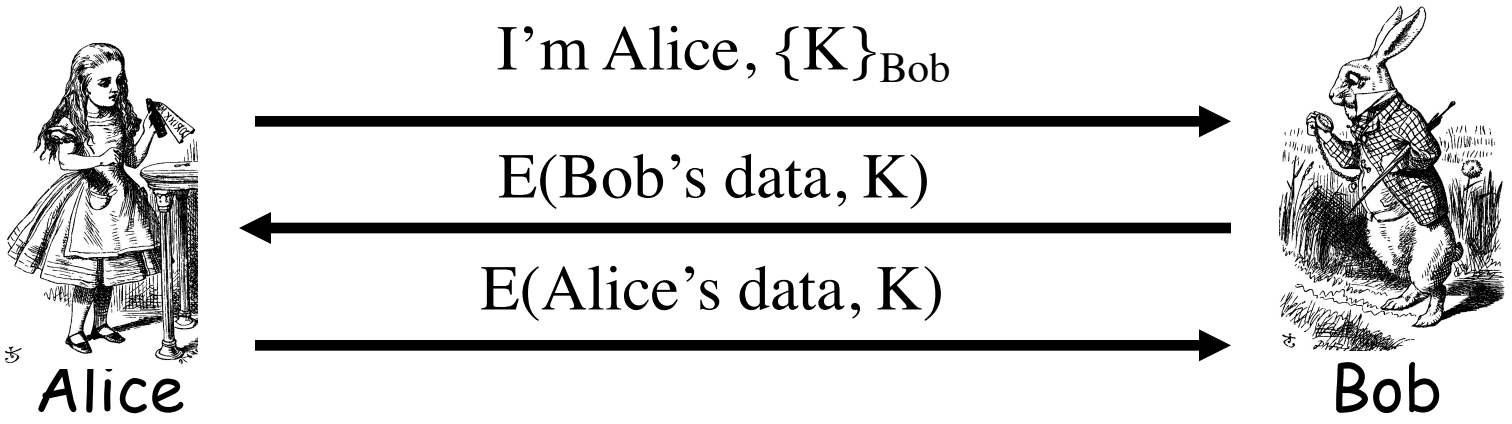
    $\{M\}_{Alice}$
- Symmetric key notation
  - Encrypt P with symmetric key K

    $C = E(P,K)$
  - 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?

Hash Functions

Hash Function Motivation

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

Hash Function Motivation

- So, Alice signs $h(M)$
  - That is, Alice computes $S = [h(M)]_{Alice}$
  - Alice then sends (M, S) to Bob
  - Bob verifies that $h(M) = \{S\}_{Alice}$
- 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 $h(M') = h(M) = \{S\}_{Alice}$

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***

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 $N = 253$

Birthday Problem

- How many people must be in a room before probability is $\geq 1/2$ that any two (or more) have same birthday?
  - $1 - 365/365 \cdot 364/365 \cdot \cdot \cdot (365-N+1)/365$
  - 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

Of Hashes and Birthdays

- If $h(x)$ is N bits, then 2^N different hash values are possible
- So, if you hash about $\text{sqrt}(2^N) = 2^{N/2}$ 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?

Non-crypto Hash (1)

- Data $X = (X_1, X_2, X_3, \ldots, X_n)$, each X_i is a byte
- Define $h(X) = (X_1 + X_2 + X_3 + \ldots + X_n) \bmod 256$
- 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…

Non-crypto Hash (2)

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

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

- 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

Non-crypto Hash (3)

- Cyclic Redundancy Check (CRC)
- Essentially, CRC is the remainder in a long division calculation
- Good for detecting burst **errors**
  - Such random errors 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)

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

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

Tiger Hash

- "Fast and strong"
- Designed by Ross Anderson and Eli Biham — leading cryptographers
- Design criteria
  - Secure
  - Optimized for 64-bit processors
  - Easy replacement for MD5 or SHA-1

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
- $\text{ipad} = 0\text{x}36$ repeated B times
- $\text{opad} = 0\text{x}5\text{C}$ 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…