Crypto

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)

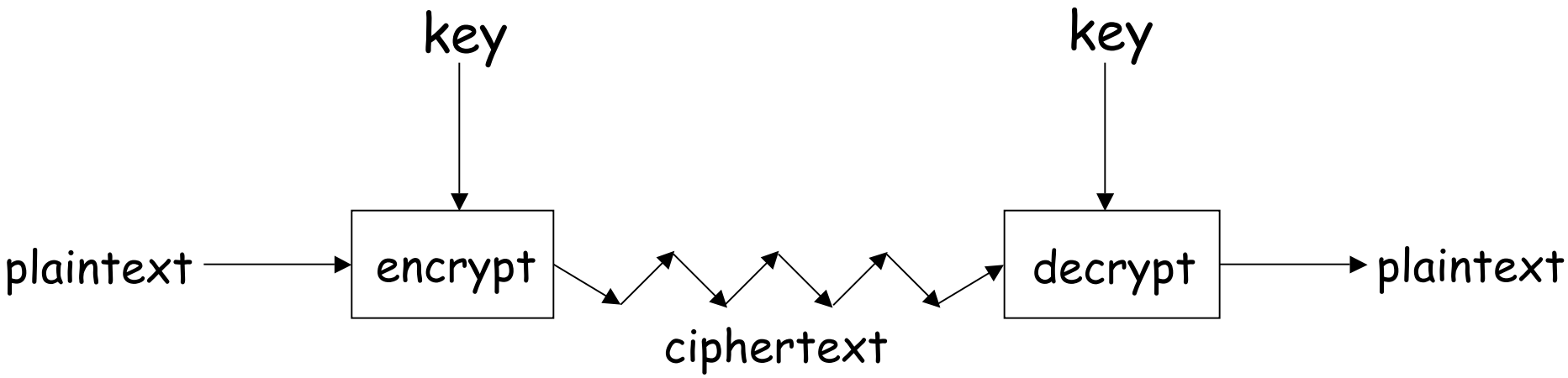
How to Speak Crypto

- 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 (sign)

Crypto

- Basic assumption
  - The system is completely known to the attacker
  - Only the key is secret
- Also known as **Kerckhoffs Principle**
  - Crypto algorithms are not secret
- Why do we make this assumption?
  - Experience has shown that secret algorithms are weak when exposed
  - Secret algorithms never remain secret
  - Better to find weaknesses beforehand

Crypto as Black Box



A generic use of 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:
  IRXUVFRUHDAGVHYHABHDUVDIR
- Shift by 3 is "Caesar's cipher"

Ceasar's Cipher Decryption

- Suppose we know a Ceasar's cipher is being used
- Ciphertext: VSRQJHEREVTXDUHSDQWU

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

- Plaintext: spongebobsquarepants

Not-so-Simple Substitution

- Shift by n for some n ∈ {0,1,2,…,25}
- Then key is n
- Example: key = 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 = 4

Even-less-Simple Substitution

- Key is some permutation of letters
- Need not be a 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

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

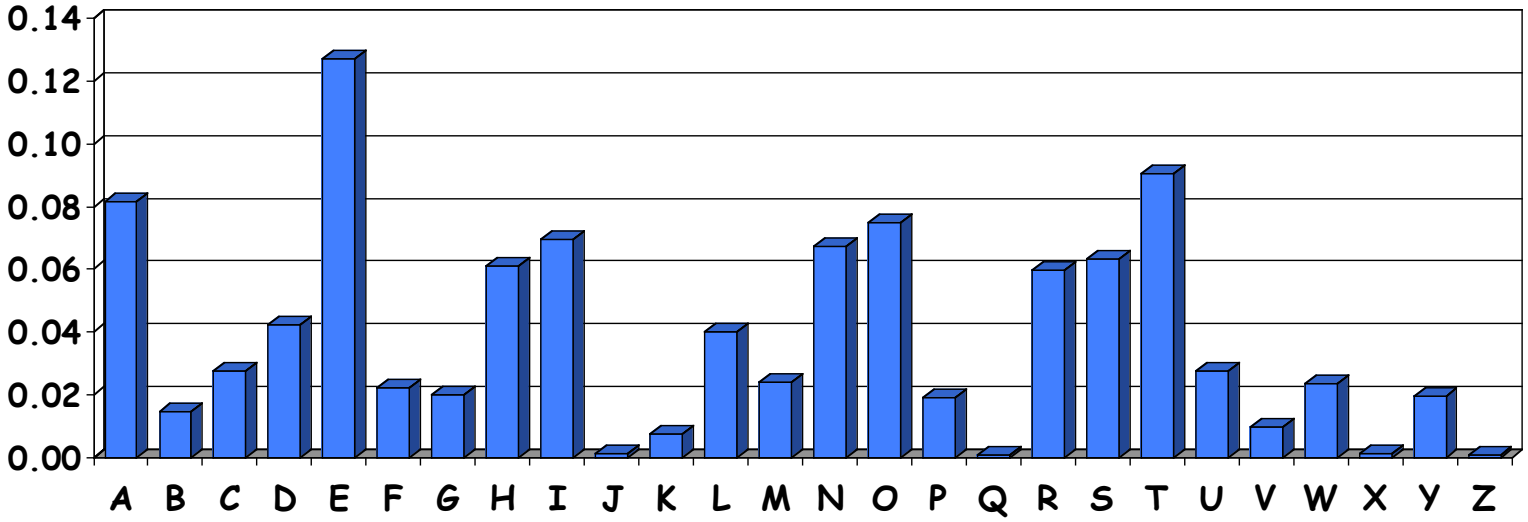
Cryptanalysis II: Be Clever

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

PBFPVYFBQXZTYFPBFEQJHDXXQVAPTPQJKTOYQWIPBVWLXTOXBT
FXQWAXBVCXQWAXFQJVWLEQNTOZQGGQLFXQWAKVWLXQW
AEBIPBFXFQVXGTVJVWLBTPQWAEBFPBFHCVLXBQUFEVWLXGDP
EQVPQGVPPBFTIXPFHXZHVFAGFOTHFEFBQUFTDHZBQPOTHXTY
FTODXQHFTDPTOGHFQPBQWAQJJTODXQHFOQPWTBDHHIXQV
APBFZQHCFWPFHPBFIPBQWKFABVYYDZBOTHPBQPQJTQOTOGH
FQAPBFEQJHDXXQVAVXEBQPEFZBVFOJIWFFACFCCFHQWAUVW
FLQHGFXVAFXQHFUFHILTTAVWAFFAWTEVOITDHFHFQAITIXP
FHXAFQHEFZQWGFLVWPTOFFA

Cryptanalysis II

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



Cryptanalysis II

- Ciphertext:

PBFPVYFBQXZTYFPBFEQJHDXXQVAPTPQJKTOYQWIPBVWLXTOXBTFXQWA
XBVCXQWAXFQJVWLEQNTOZQGGQLFXQWAKVWLXQWAEBIPBFXFQVX
GTVJVWLBTPQWAEBFPBFHCVLXBQUFEVWLXGDPEQVPQGVPPBFTIXPFHXZ
HVFAGFOTHFEFBQUFTDHZBQPOTHXTYFTODXQHFTDPTOGHFQPBQWAQ
JJTODXQHFOQPWTBDHHIXQVAPBFZQHCFWPFHPBFIPBQWKFABVYYDZB
OTHPBQPQJTQOTOGHFQAPBFEQJHDXXQVAVXEBQPEFZBVFOJIWFFACF
CCFHQWAUVWFLQHGFXVAFXQHFUFHILTTAVWAFFAWTEVOITDHFHFQ
AITIXPFHXAFQHEFZQWGFLVWPTOFFA

- Decrypt this message using info 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

Cryptanalysis: Terminology

- Cryptosystem is **secure** if best know attack is to try all keys
- Cryptosystem is **insecure** if any shortcut attack is known
- By this definition, an insecure system might be harder to break than a secure system!

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: matrix size and permutations (3,5,1,4,2) and (1,3,2)

One-time Pad Encryption

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

Encryption: Plaintext ⊕ 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

One-time Pad Decryption

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

Decryption: Ciphertext ⊕ Key = Plaintext

	s	r	l	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	l	h	i	t	l	e	r

One-time Pad

Double agent claims sender used "**key**":

	s	r	l	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	l	l	h	i	t	l	e	r

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

One-time Pad

Sender is captured and claims the key is:

	s	r	l	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	l	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, when used correctly
  - Ciphertext provides no info about plaintext
  - All plaintexts are equally likely
  - Pad must be random, used only once
  - Pad is known only by sender and receiver
  - Pad is same size as message
  - No assurance of message integrity
- Why not distribute message the same way as the pad?

Real-world One-time Pad

- Project VENONA
  - Soviet spy messages from U.S. in 1940's
  - Nuclear espionage, etc.
  - Thousands of messaged
- Spy carried one-time pad into U.S.
- Spy used pad to encrypt secret messages
- Repeats within the "one-time" pads made cryptanalysis possible

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

Codebook

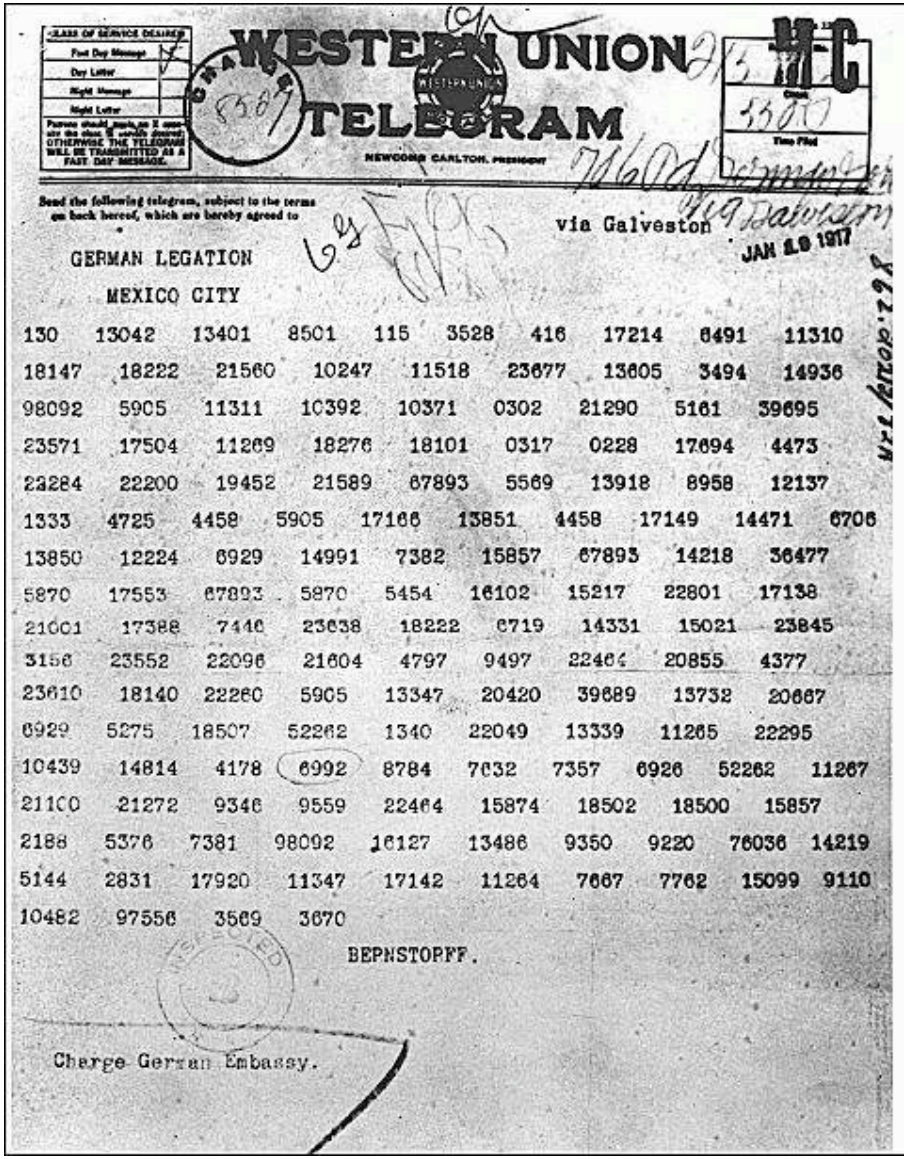
- Literally, a book filled with "codewords"
- Zimmerman Telegram encrypted via codebook

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

- Modern block ciphers are codebooks!
- More on this later…

Zimmerman Telegram

- One of most famous codebook ciphers ever
- Led to US entry in WWI
- Ciphertext shown here…



WESTERN UNION TELEGRAM

NEWCOMB CARLTON, PRESIDENT

via Galveston

JAN 19 1917

GERMAN LEGATION

MEXICO CITY

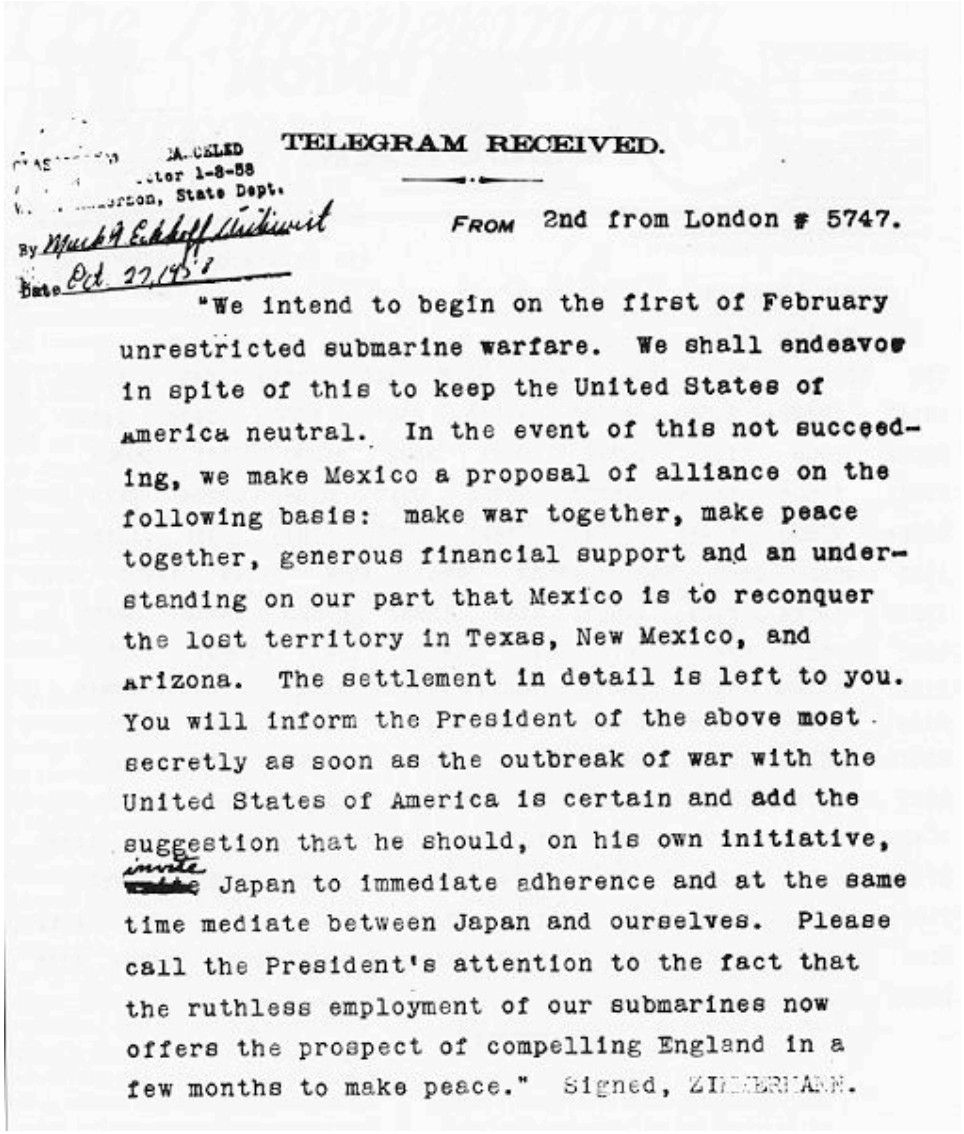
130 13042 13401 8501 115 3528 416 17214 6491 11310
18147 18222 21560 10247 11518 23677 13605 3494 14936
98092 5905 11311 10392 10371 0302 21290 5161 39695
23571 17504 11269 18276 18101 0317 0228 17694 4473
23284 22200 19452 21589 67893 5569 13918 8958 12137
1333 4725 4458 5905 17166 13851 4458 17149 14471 6706
13850 12224 6929 14991 7382 15857 67893 14218 36477
5870 17553 67893 5870 5454 16102 15217 22801 17138
21001 17388 7446 23638 18222 6719 14331 15021 23845
3156 23552 22096 21604 4797 9497 22464 20855 4377
23610 18140 22260 5905 13347 20420 39689 13732 20667
6929 5275 18507 52262 1340 22049 13339 11265 22295
10439 14814 4178 6992 8784 7632 7357 6926 52262 11267
21100 21272 9346 9559 22464 15874 18502 18500 15857
2188 5376 7381 98092 16127 13486 9350 9220 76036 14219
5144 2831 17920 11347 17142 11264 7667 7762 15099 9110
10482 97556 3569 3670

BEPNSTORFF.

Charge German Embassy.

Zimmerman Telegram Decrypted

- British had recovered partial codebook
- Able to fill in missing parts



TELEGRAM RECEIVED.

FROM 2nd from London # 5747.

"We intend to begin on the first of February unrestricted submarine warfare. We shall endeavor in spite of this to keep the United States of America neutral. In the event of this not succeeding, we make Mexico a proposal of alliance on the following basis: make war together, make peace together, generous financial support and an understanding on our part that Mexico is to reconquer the lost territory in Texas, New Mexico, and Arizona. The settlement in detail is left to you. You will inform the President of the above most secretly as soon as the outbreak of war with the United States of America is certain and add the suggestion that he should, on his own initiative, invite Japan to immediate adherence and at the same time mediate between Japan and ourselves. Please call the President's attention to the fact that the ruthless employment of our submarines now offers the prospect of compelling England in a few months to make peace." Signed, ZIMMERMANN.

A Few Historical Items

- Crypto timeline
- Spartan Scytale — transposition cipher
- Caesar's cipher
- Poe's *The Gold Bug*
- Election of 1876

Election of 1876

- "Rutherfraud" Hayes vs "Swindling" Tilden
  - Popular vote was virtual tie
- Electoral college delegations for 4 states (including Florida) in dispute
- Commission: All 4 states to Hayes
- Tilden accused Hayes of bribery
  - Was it true?

Election of 1876

- Encrypted messages by Tilden supporters later emerged
- Cipher: Partial codebook, plus transposition
- Codebook substitution for important words

ciphertext	plaintext
Copenhagen	Greenbacks
Greece	Hayes
Rochester	votes
Russia	Tilden
Warsaw	telegram
:	:

Election of 1876

- Apply codebook to original message
- Pad message to multiple of 5 words (total length, 10,15,20,25 or 30 words)
- For each length, a fixed permutation applied to resulting message
- Permutations found by comparing many messages of same length
- Note that the **same key** is applied to all messages of a given length

Election of 1876

- Ciphertext: **Warsaw they read all unchanged last are idiots can't situation**
- Codebook: Warsaw == telegram
- Transposition: 9,3,6,1,10,5,2,7,4,8
- Plaintext: **Can't read last telegram. Situation unchanged. They are all idiots.**
- A weak cipher made worse by reuse of key
- **Lesson: Don't reuse/overuse keys!**

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
- Data Encryption Standard (DES), 70's
- Public Key cryptography, 70's
- CRYPTO conferences, 80's
- Advanced Encryption Standard (AES), 90's
- Crypto moved out of classified world

Claude Shannon

- The founder of Information Theory
- 1949 paper: *Comm. Thy. of Secrecy Systems*
- Confusion and diffusion
  - **Confusion** — obscure relationship between plaintext and ciphertext
  - **Diffusion** — spread plaintext statistics through the ciphertext
  - Proved that one-time pad is secure
  - One-time pad only uses confusion, while double transposition only uses diffusion

Taxonomy of Cryptography

- **Symmetric Key**
  - Same key for encryption as for decryption
  - Stream ciphers
  - Block ciphers
- **Public Key**
  - Two keys, one for encryption (public), and one for decryption (private)
  - Digital signatures — nothing comparable in symmetric key crypto
- **Hash algorithms**

Taxonomy of Cryptanalysis

- Ciphertext only
- Known plaintext
- Chosen plaintext
  - "Lunchtime attack"
  - Protocols might encrypt chosen text
- Adaptively chosen plaintext
- Related key
- Forward search (public key crypto only)
- Etc., etc.

Symmetric Key Crypto

Symmetric Key Crypto

- Stream cipher — like a one-time pad
  - Key is relatively short
  - Key is stretched into a long **keystream**
  - Keystream is then used like a one-time pad
- Block cipher — based on codebook concept
  - Block cipher key determines a codebook
  - Each key yields a different codebook
  - Employ both "confusion" and "diffusion"

Stream Ciphers

Stream Ciphers

- Not as popular today as block ciphers
- We'll discuss two examples
- A5/1
  - Based on shift registers
  - Used in GSM mobile phone system
- RC4
  - Based on a changing lookup table
  - Used many places

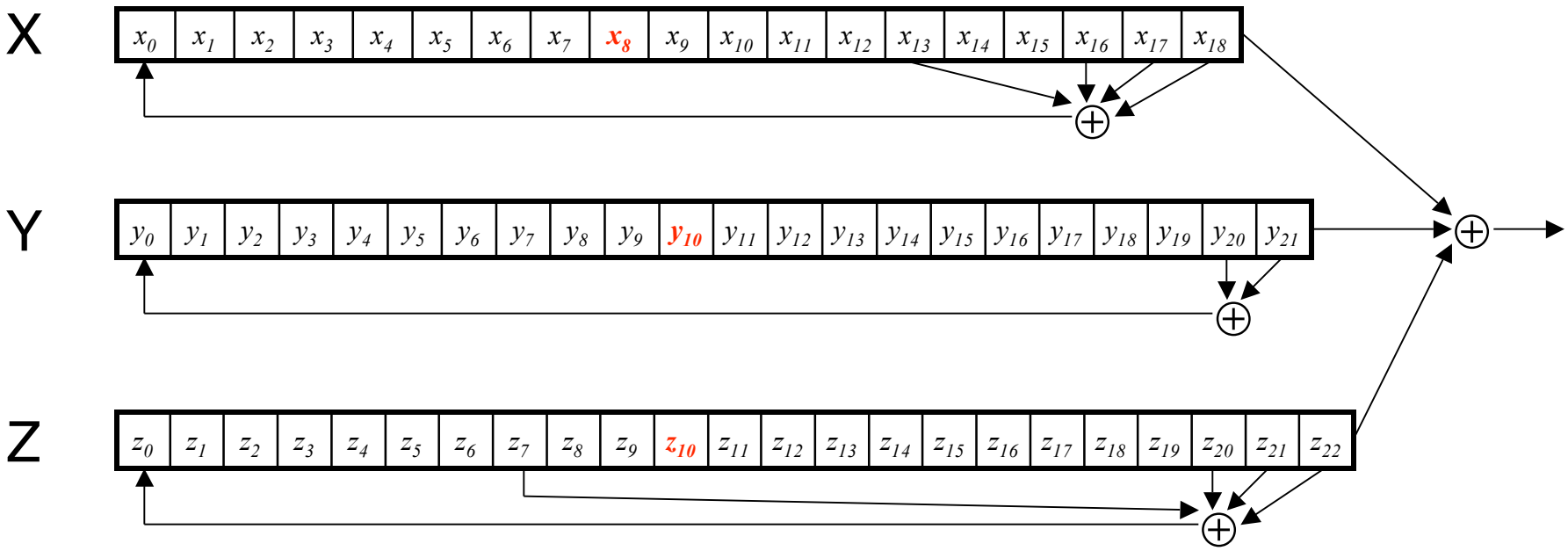
A5/1

- A5/1 consists of 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

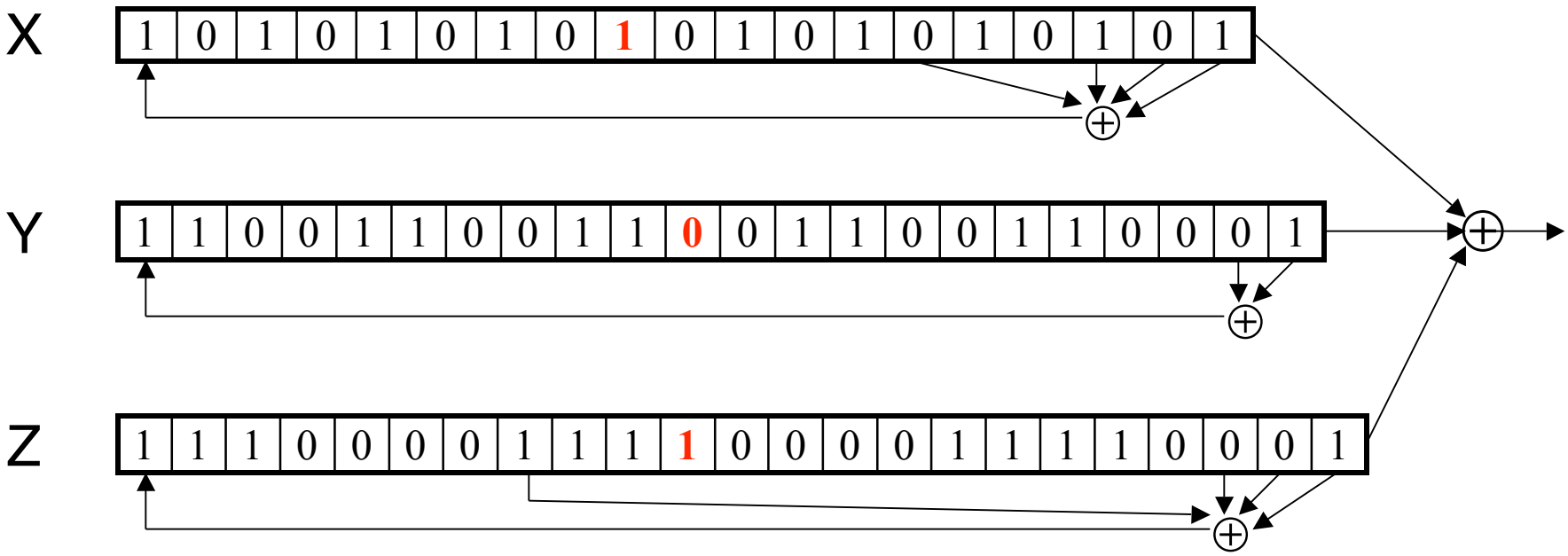
- At each step: $m = \text{maj}(x_8, y_{10}, z_{10})$
  - Examples: $\text{maj}(0,1,0) = 0$ and $\text{maj}(1,1,0) = 1$
- If $x_8 = m$ then X steps
  - $t = x_{13} \oplus x_{16} \oplus x_{17} \oplus x_{18}$
  - $x_i = x_{i-1}$ for $i = 18,17,\ldots,1$ and $x_0 = t$
- If $y_{10} = m$ then Y steps
  - $t = y_{20} \oplus y_{21}$
  - $y_i = y_{i-1}$ 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 value is a single bit
- Key is used as initial fill of registers
- Each register steps or not, based on (x_8, y_{10}, z_{10})
- Keystream bit is XOR of right bits of registers

A5/1



- In this example, $m = \text{maj}(x_8, y_{10}, z_{10}) = \text{maj}(1,0,1) = 1$
- 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-based crypto is efficient in hardware
- Harder to implement in software
- In the past, very popular
- Today, more is done in software due to faster processors
- Shift register crypto still used some

RC4

- A self-modifying lookup table
- Table always contains some permutation of 0,1,...,255
- Initialize the permutation using key
- At each step, RC4
  - 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 j
i = j = 0
```

RC4 Keystream

- For each keystream byte, swap table elements and select 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 must be discarded
  - Otherwise attacker may be able to recover key

Stream Ciphers

- Stream ciphers were big in the past
  - Efficient in hardware
  - Speed needed to keep up with voice, etc.
  - Today, processors are fast, so software-based crypto is fast enough
- Future of stream ciphers?
  - Shamir: "the death of stream ciphers"
  - May be exaggerated…

Block Ciphers

(Iterated) Block Cipher

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

Feistel Cipher

- **Feistel cipher** refers to a type of block cipher design, not a specific cipher
- Split plaintext block into left and right halves: Plaintext = (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 = (L_n,R_n)

Feistel Cipher

- Decryption: Ciphertext = (L_n, R_n)
- For each round i=n,n–1,…,1, compute

  $R_{i-1} = L_i$

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

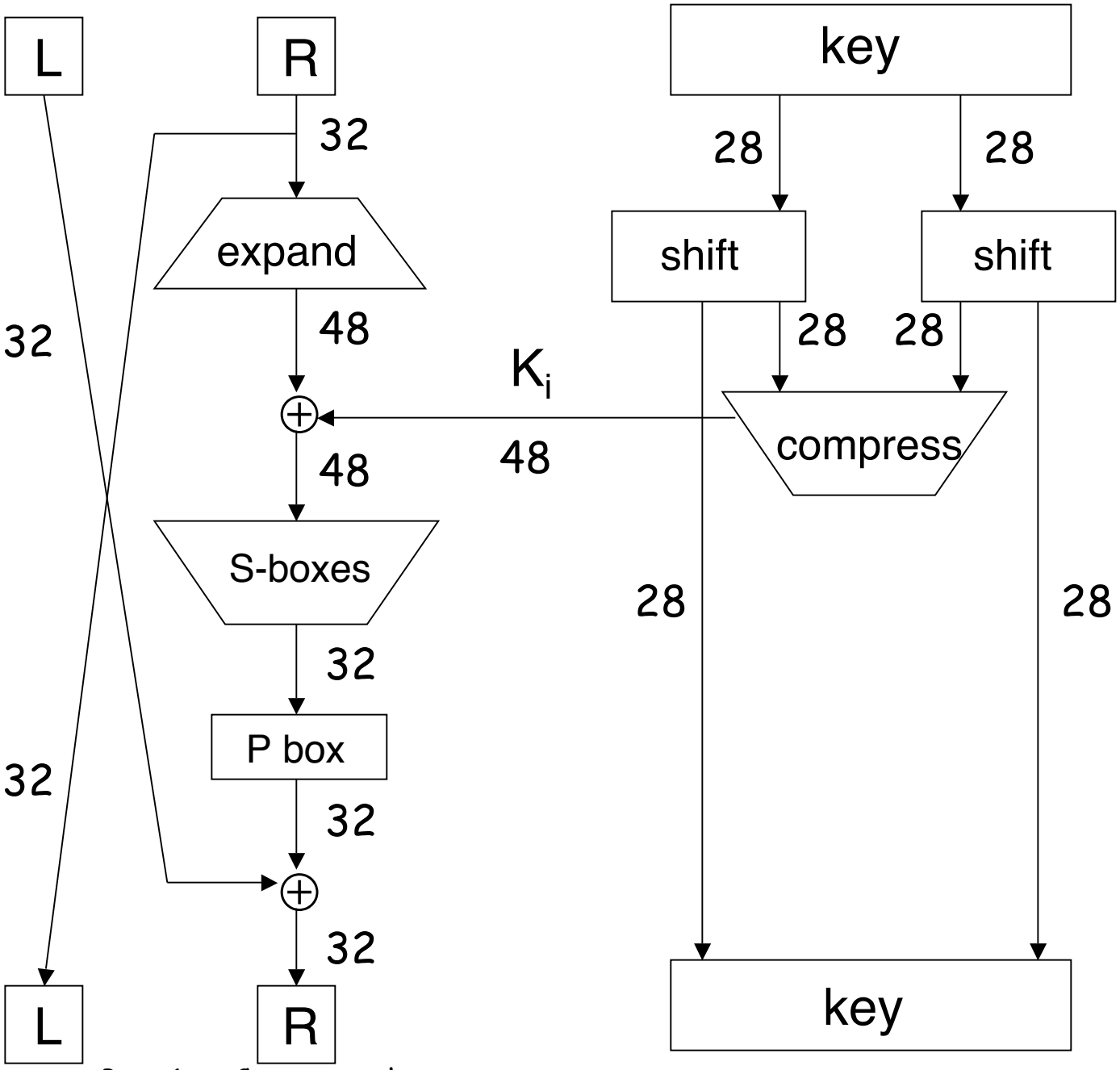
  where F is round function and K_i is subkey
- Plaintext = (L_0, R_0)
- Formula "works" for any function F
- But only secure for certain functions F

Data Encryption Standard

- DES developed in 1970's
- Based on IBM Lucifer cipher
- U.S. government standard
- DES development was controversial
  - NSA was secretly involved
  - Design process not open
  - Key length was reduced
  - Subtle changes to Lucifer algorithm

DES Numerology

- DES is a Feistel cipher
  - 64 bit block length
  - 56 bit key length
  - 16 rounds
  - 48 bits of key used each round (subkey)
- Each round is simple (for a block cipher)
- Security depends primarily on "S-boxes"
  - Each S-boxes 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
- 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 perm P before round 1
- Halves are swapped after last round
- A final permutation (inverse of P) is applied to (R_{16},L_{16}) to yield ciphertext
- None of these serve any security purpose

Security of DES

- Security of DES depends a lot on S-boxes
  - Everything else in DES is linear
- Thirty years of intense analysis has revealed no "back door"
- Attacks today use exhaustive key search
- **Inescapable conclusions**
  - Designers of DES knew what they were doing
  - Designers of DES were ahead of their time

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 that
  - P = D(E(P, K), K) and C = E(D(C, K), K)

Triple DES

- Today, 56 bit DES key is too small
- But DES is everywhere: 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 use Encrypt-Decrypt-Encrypt (EDE) with 2 keys?
  - Backward compatible: $E(D(E(P,K),K),K) = E(P,K)$
  - And 112 bits is enough

3DES

- Why not C = E(E(P,K),K) ?
  - Still just 56 bit key
- Why not C = E(E(P,K_1),K_2) ?
- A (semi-practical) known plaintext attack
  - Precompute 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 is keys: C = E(E(P,K_1),K_2)

Advanced Encryption Standard

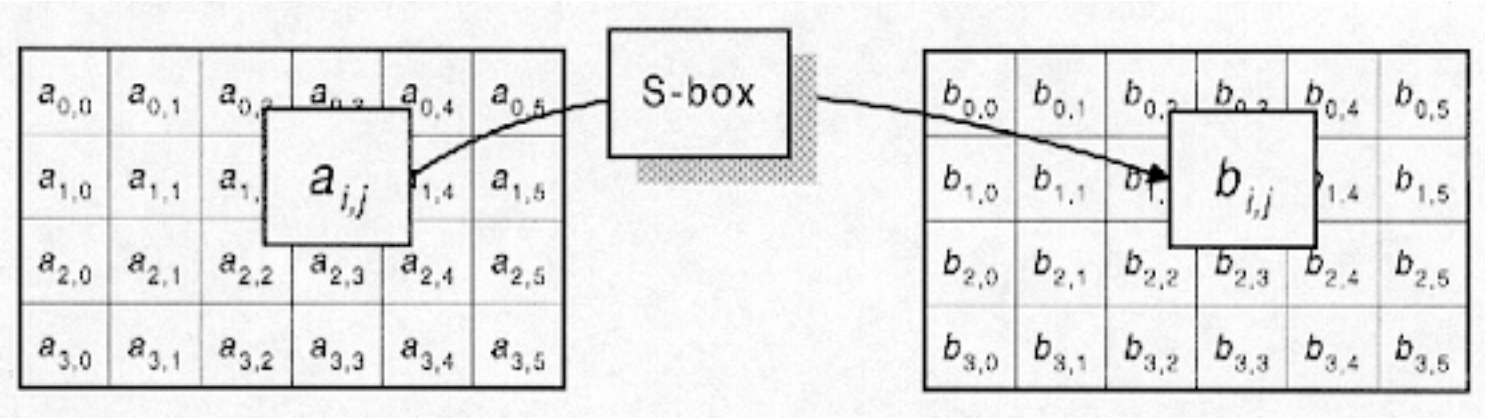
- Replacement for DES
- AES competition (late 90's)
  - NSA openly involved
  - Transparent process
  - Many strong algorithms proposed
  - Rijndael Algorithm ultimately selected
    - Pronounced like "Rain Doll" or "Rhine Doll"
- Iterated block cipher (like DES)
- Not a Feistel cipher (unlike DES)

AES Overview

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

AES ByteSub

- Assume 192 bit block, 4x6 bytes



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

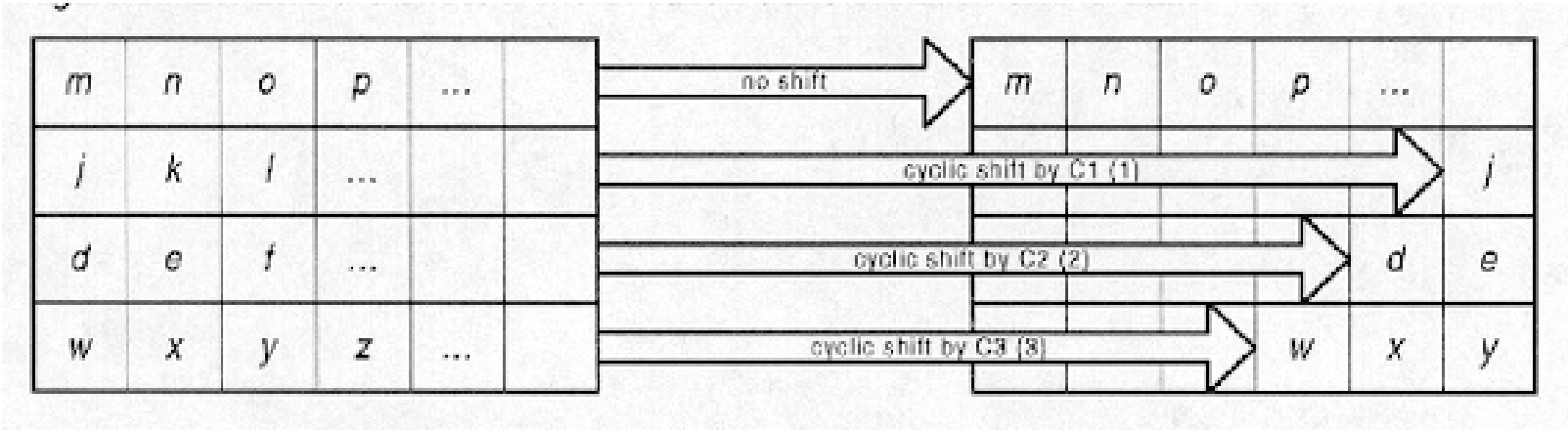
AES "S-box"

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

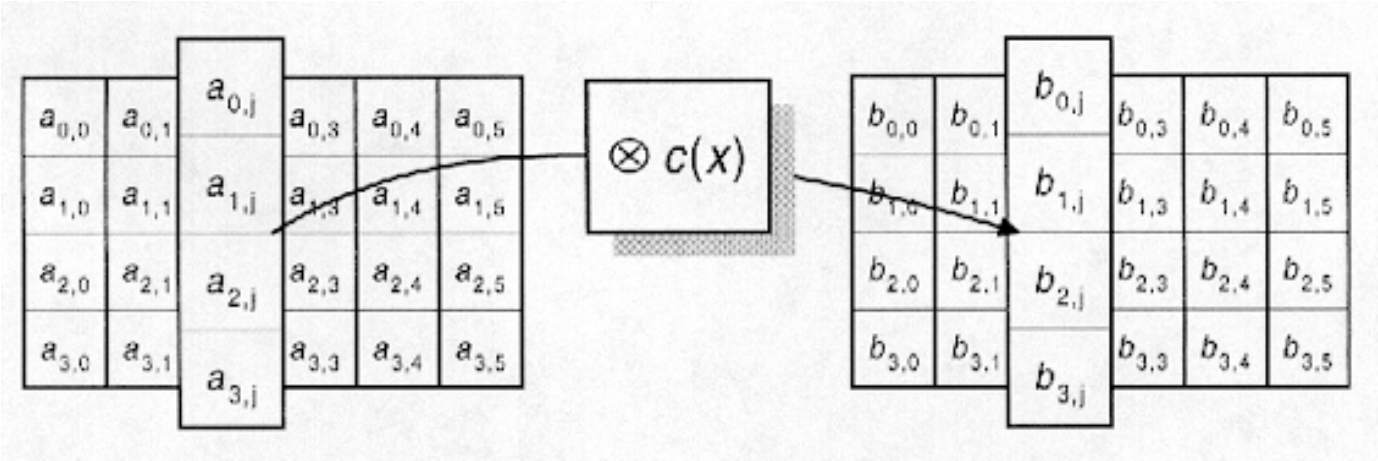
AES ShiftRow

- Cyclic shift rows



AES MixColumn

- Nonlinear, invertible operation applied to each column



- Implemented as a (big) lookup table

AES AddRoundKey

- XOR subkey with block

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} & a_{04} & a_{05} \\ a_{10} & a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{20} & a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{30} & a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \end{bmatrix} \oplus \begin{bmatrix} k_{00} & k_{01} & k_{02} & k_{03} & k_{04} & k_{05} \\ k_{10} & k_{11} & k_{12} & k_{13} & k_{14} & k_{15} \\ k_{20} & k_{21} & k_{22} & k_{23} & k_{24} & k_{25} \\ k_{30} & k_{31} & k_{32} & k_{33} & k_{34} & k_{35} \end{bmatrix} = \begin{bmatrix} b_{00} & b_{01} & b_{02} & b_{03} & b_{04} & b_{05} \\ b_{10} & b_{11} & b_{12} & b_{13} & b_{14} & b_{15} \\ b_{20} & b_{21} & b_{22} & b_{23} & b_{24} & b_{25} \\ b_{30} & b_{31} & b_{32} & b_{33} & b_{34} & b_{35} \end{bmatrix}$$

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
- More detailed…
  - TEA

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 to rely on data as part of algorithm

Tiny Encryption Algorithm

- 64 bit block, 128 bit key
- Assumes 32-bit arithmetic
- Number of rounds is variable (32 is considered secure)
- Uses "weak" round function, so large number rounds required

TEA Encryption

Assuming 32 rounds:

(K[0],K[1],K[2],K[3]) = 128 bit key
(L,R) = plaintext (64-bit block)
delta = 0x9e3779b9
sum = 0
for i = 1 to 32
    sum += delta
    L += ((R<<4)+K[0])^(R+sum)^((R>>5)+K[1])
    R += ((L<<4)+K[2])^(L+sum)^((L>>5)+K[3])
next i
ciphertext = (L,R)

TEA Decryption

Assuming 32 rounds:

(K[0],K[1],K[2],K[3]) = 128 bit key
(L,R) = ciphertext (64-bit block)
delta = 0x9e3779b9
sum = delta << 5
for i = 1 to 32
    R –= ((L<<4)+K[2])^(L+sum)^((L>>5)+K[3])
    L –= ((R<<4)+K[0])^(R+sum)^((R>>5)+K[1])
    sum –= delta
next i
plaintext = (L,R)

TEA comments

- **Almost** a Feistel cipher
  - Uses + and - instead of ⊕ (XOR)
- Simple, easy to implement, fast, low memory requirement, etc.
- Possibly a related key attack
- eXtended TEA (XTEA) eliminates related key attack (slightly more complex)
- Simplified TEA (STEA) — insecure version used as an example for cryptanalysis

Block Cipher Modes

Multiple Blocks

- How to encrypt multiple blocks?
- A new key for each block?
  - As bad as (or worse than) a one-time pad!
- Encrypt each block independently?
- Make encryption depend on previous block(s), i.e., "chain" the blocks together?
- How to handle partial blocks?

Modes of Operation

- Many modes of operation — we discuss three
- Electronic Codebook (**ECB**) mode
  - Obvious thing to do
  - Encrypt each block independently
  - There is a serious weakness
- Cipher Block Chaining (**CBC**) mode
  - Chain the blocks together
  - More secure than ECB, virtually no extra work
- Counter Mode (**CTR**) mode
  - Acts like a stream cipher
  - Popular for random access

ECB Mode

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

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 a fixed key K, this is an electronic version of a codebook cipher
- A new codebook for each key

ECB Cut and Paste Attack

- Suppose plaintext is

  `Alice digs Bob. Trudy digs Tom.`

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

  P_0 = "`Alice di`", P_1 = "`gs Bob. `",

  P_2 = "`Trudy di`", P_3 = "`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

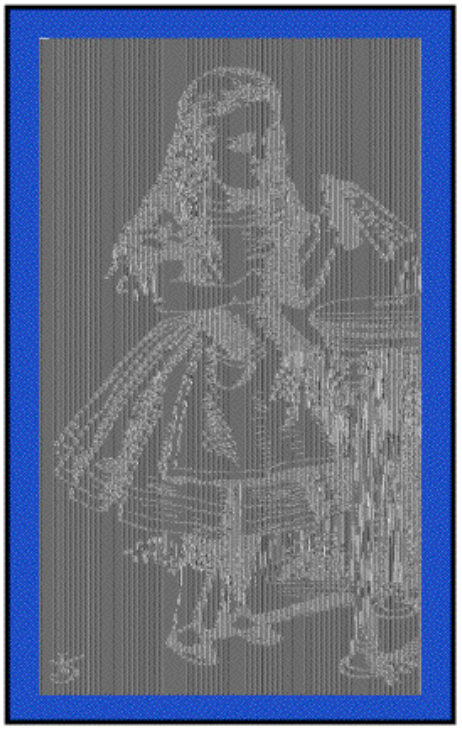
  `Alice digs Tom. Trudy digs Bob.`

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, Alice ECB encrypted (TEA)



- Why does this happen?
- Same plaintext block ⇒ same ciphertext!

CBC Mode

- Blocks are "chained" together
- A random initialization vector, or IV, is required to initialize CBC mode
- IV is random, but need not be secret

Encryption

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

Decryption

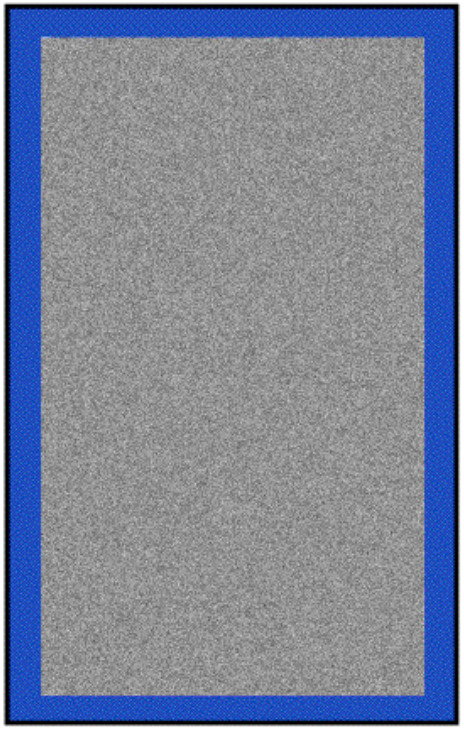
$P_0 = IV \oplus D(C_0, K)$,
$P_1 = C_0 \oplus D(C_1, K)$,
$P_2 = C_1 \oplus D(C_2, K)$,…

CBC Mode

- Identical plaintext blocks yield different ciphertext blocks
- Cut and paste is still possible, but more complex (and will cause garbles)
- 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 $P_3 = C_2 \oplus D(C_3, K)$, $P_4 = C_3 \oplus D(C_4, K)$,…
- Automatically recovers from errors!

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 stream cipher

Encryption

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

Decryption

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

- CBC can also be used for random access!!!

Integrity

Data Integrity

- **Integrity** — prevent (or at least detect) unauthorized modification of data
- Example: Inter-bank fund transfers
  - Confidentiality is nice, but integrity is critical
- Encryption provides **confidentiality** (prevents unauthorized disclosure)
- Encryption alone does **not** assure integrity (recall one-time pad and attack on ECB)

MAC

- Message Authentication Code (MAC)
  - Used for data **integrity**
  - Integrity **not** the same as confidentiality
- MAC is computed as **CBC residue**
  - Compute CBC encryption, but only save the final ciphertext block

MAC Computation

- MAC computation (assuming N blocks)

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

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

  $C_2 = E(C_1 \oplus P_2, K)$,…

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

- MAC sent along with plaintext
- Receiver does same computation and verifies that result agrees with MAC
- Receiver must also know the key K

Why 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

  $C_0 = E(IV \oplus P_0, K)$, $\mathbf{C_1} = E(C_0 \oplus X, K)$,
  $\mathbf{C_2} = E(\mathbf{C_1} \oplus P_2, K)$, $\mathbf{C_3} = E(\mathbf{C_2} \oplus P_3, K) = \mathbf{MAC} \neq MAC$
- Error **propagates** into **MAC** (unlike CBC decryption)
- Trudy can't change **MAC** to MAC without key K

Confidentiality and Integrity

- Encrypt with one key, compute MAC with another
- Why not use the same key?
  - Send last encrypted block (MAC) twice?
  - Can't add any security!
- Using different keys to encrypt and compute MAC works, even if keys are related
  - But still twice as much work as encryption alone
- Confidentiality and integrity with 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 (later…)
- Anything you can do with a hash function (upcoming chapter…)

Public Key Cryptography

Public Key Cryptography

- Two keys
  - Sender uses recipient's **public key** to encrypt
  - Receiver uses his **private key** to decrypt
- Based on **trap door, one way function**
  - Easy to compute in one direction
  - Hard to compute in other direction
  - "Trap door" used to create keys
  - Example: Given p and q, product N=pq is easy to compute, but given N, it is hard to find p and q

Public Key Cryptography

- Encryption
  - Suppose we encrypt M with Bob's public key
  - Only Bob's private key can decrypt to find M
- Digital Signature
  - **Sign** by "encrypting" with private key
  - Anyone can **verify** signature by "decrypting" with public key
  - But only private key holder could have signed
  - Like a handwritten signature (and then some)

Knapsack

Knapsack Problem

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

  $$S = a_0W_0 + a_1W_1 + \ldots + a_{n-1}W_{n-1}$$

  (technically, this is "subset sum" problem)
- **Example**
  - Weights (62,93,26,52,166,48,91,141)
  - Problem: Find subset that sums to S=302
  - Answer: 62+26+166+48=302
- The (general) knapsack is NP-complete

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

Knapsack Cryptosystem

1. Generate superincreasing knapsack (SIK)
2. Convert SIK into "general" knapsack (GK)
3. **Public Key:** GK
4. **Private Key:** SIK plus conversion factors

- Easy to encrypt with GK
- With private key, easy to decrypt (convert ciphertext to SIK)
- Without private key, must solve GK (???)

Knapsack Cryptosystem

- Let (2,3,7,14,30,57,120,251) be the SIK
- Choose m = 41 and n = 491 with m, n rel. prime and n greater than sum of elements of SIK
- 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)

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

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
- This knapsack cryptosystem is **insecure**
  - Broken in 1983 with Apple II computer
  - The attack uses **lattice reduction**
- "General knapsack" is not general enough!
- This special knapsack is easy to solve!

RSA

RSA

- Invented by Cocks (GCHQ), independently, by Rivest, Shamir and Adleman (MIT)
- 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 s.t. ed = 1 mod (p–1)(q–1)
- **Public key** is (N,e)
- **Private key** is d

RSA

- To encrypt message M compute
  - $C = M^e \bmod N$
- To decrypt C compute
  - $M = C^d \bmod N$
- Recall that e and N are public
- If attacker can factor N, he can use e to easily find d since $ed = 1 \bmod (p-1)(q-1)$
- Factoring the modulus breaks RSA
- It is not known whether factoring is the only way to break RSA

Does RSA Really Work?

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

Simple RSA Example

- Example of 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 mod 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 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 $2 = 1 \cdot 2$, $5 = 2 \cdot 2 + 1$, $10 = 2 \cdot 5$, $20 = 2 \cdot 10$
  - $5^1 = 5 \bmod 35$
  - $5^2 = (5^1)^2 = 5^2 = 25 \bmod 35$
  - $5^5 = (5^2)^2 \cdot 5^1 = 25^2 \cdot 5 = 3125 = 10 \bmod 35$
  - $5^{10} = (5^5)^2 = 10^2 = 100 = 30 \bmod 35$
  - $5^{20} = (5^{10})^2 = 30^2 = 900 = 25 \bmod 35$
- Never have to deal with huge numbers!

More Efficient RSA (2)

- Let **e = 3** for all users (but not same N or d)
  - Public key operations only require 2 multiplies
  - 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

Diffie-Hellman

Diffie-Hellman

- 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
- Security rests on difficulty of **discrete log** problem: given g, p, and g^k mod p find k

Diffie-Hellman

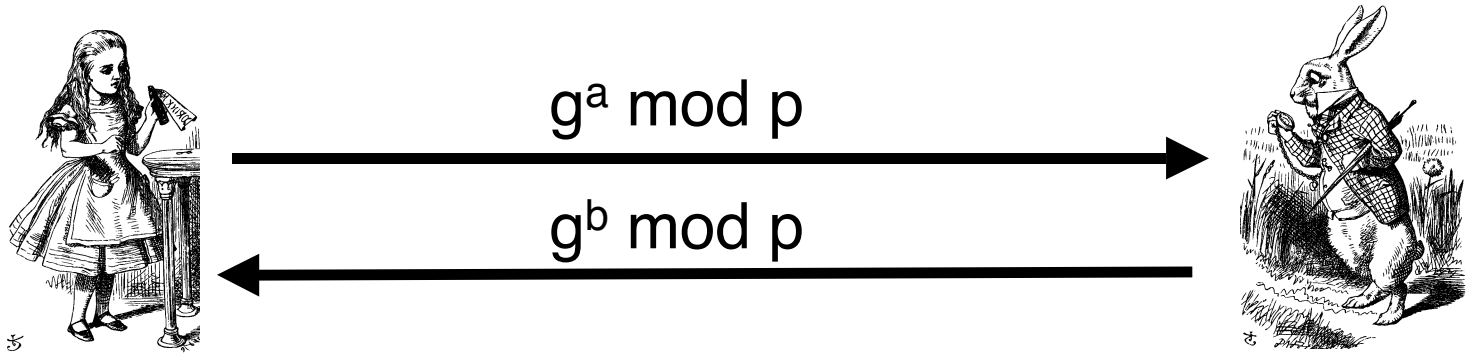
- Let p be prime, let g be a **generator**
  - For any $x \in \{1,2,\ldots,p-1\}$ there is n s.t. $x = g^n \bmod p$
- Alice selects secret value a
- Bob selects secret value b
- Alice sends $g^a \bmod p$ to Bob
- Bob sends $g^b \bmod p$ to Alice
- Both compute shared secret $g^{ab} \bmod p$
- Shared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

Diffie-Hellman

- Suppose that Bob and Alice use $g^{ab} \bmod p$ as a symmetric key
- Trudy can see $g^a \bmod p$ and $g^b \bmod p$
- Note $g^a g^b \bmod p = g^{a+b} \bmod p \neq g^{ab} \bmod p$
- If Trudy can find a or b, system is broken
- If Trudy can solve **discrete log** problem, then she can find a or b

Diffie-Hellman

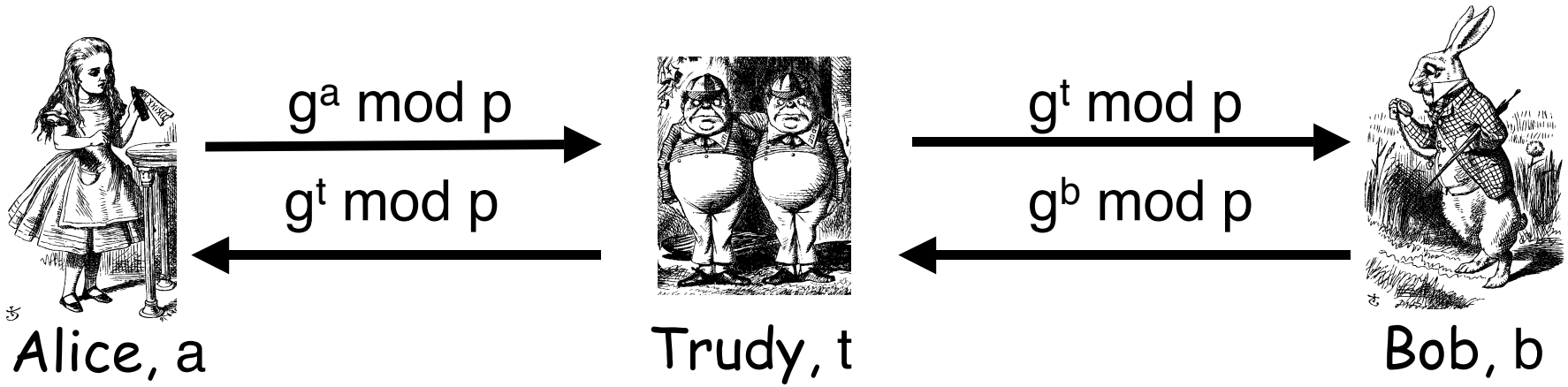
- **Public:** g and p
- **Secret:** Alice's exponent a, Bob's exponent b



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

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 exists!

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?
- 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 are a different way to do the math in public key system
- Elliptic curve versions of DH, RSA, etc.
- Elliptic curves may be more efficient
  - Fewer bits needed for same security
  - But the operations are more complex

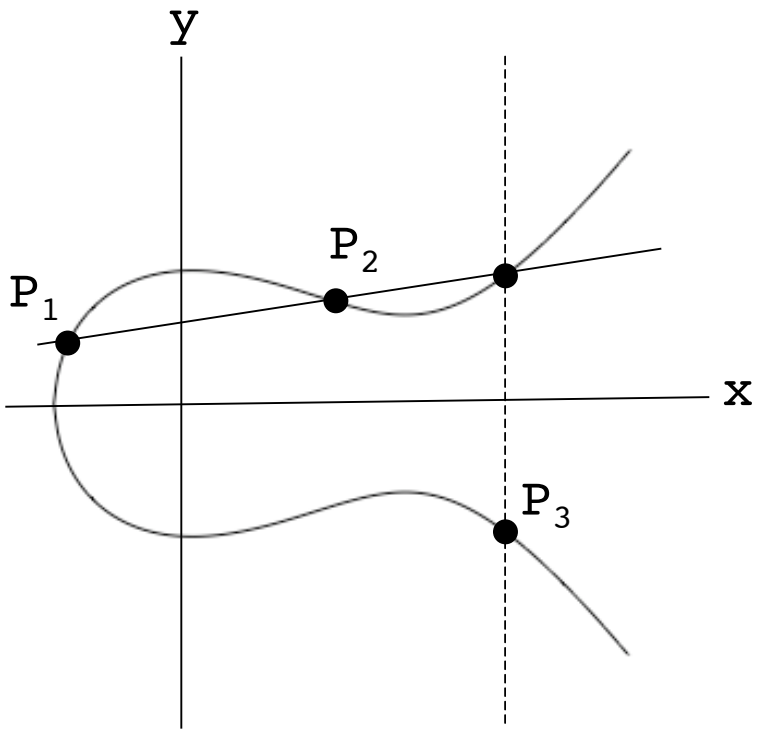
What is an Elliptic Curve?

- 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

$$P_3 = P_1 + P_2$$

  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
- **Secret:** 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 secret:** $A = 4$
- **Bob's secret:** $B = 7$
- 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: $7(7,32) = (22,1)$

Uses for Public Key Crypto

Uses for Public Key Crypto

- Confidentiality
  - Transmitting data over insecure channel
  - Secure storage on insecure media
- Authentication (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!** Since Bob also knows symmetric key, he could have forged message
- **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!** Only someone with Alice's private key could have signed the order
- This assumes Alice's private key is not stolen (revocation problem)

Sign and Encrypt
vs
Encrypt and Sign

Public Key Notation

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

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

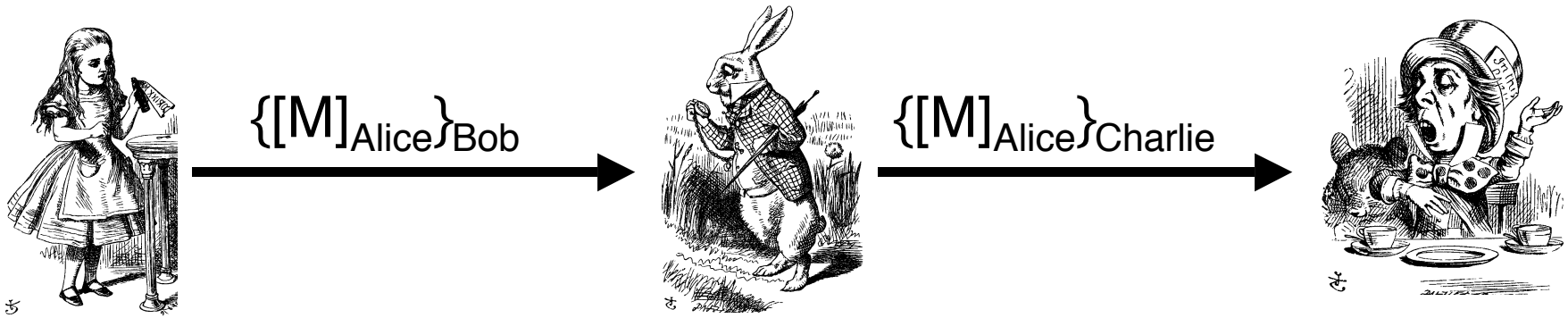
$$[\{M\}_{Alice}]_{Alice} = M$$

Confidentiality and Non-repudiation

- Suppose that we want confidentiality and 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

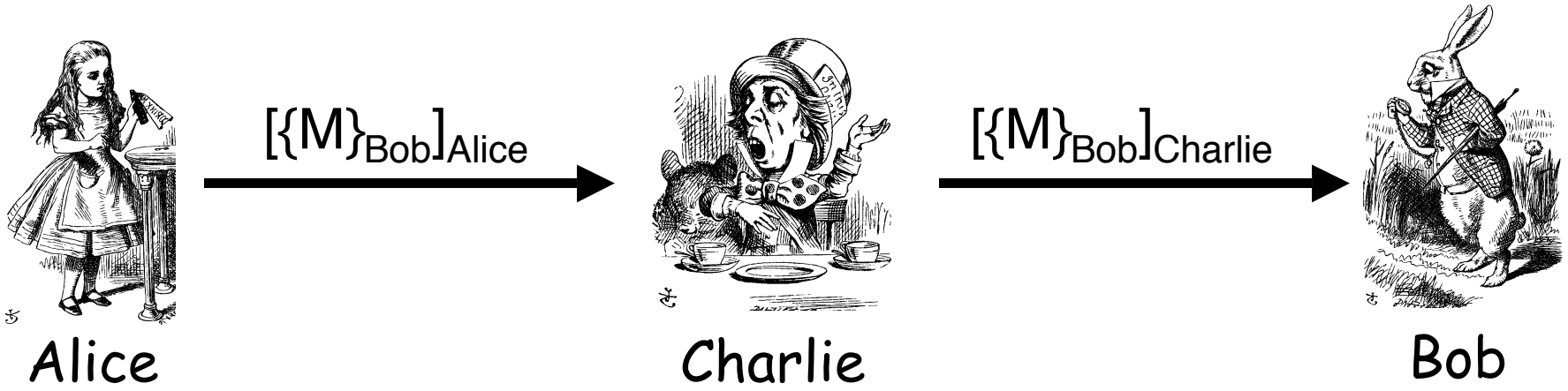
- M = "I love you"



- **Q:** What is the problem?
- **A:** Charlie misunderstands crypto!

Encrypt and Sign

- M = "My theory, which is mine…."



- **Note** that Charlie cannot decrypt M
- **Q:** What is the problem?
- **A:** Bob misunderstands crypto!

Public Key Infrastructure

Public Key Certificate

- Contains name of user and user's public key (and possibly other info)
- Certificate is **signed** by the issuer (such as VeriSign) who vouches for it
- Signature on certificate is verified using signer's public key

Certificate Authority

- Certificate authority (CA) is a trusted 3rd party (TTP) that issues and signs cert's
  - Verifying signature verifies the identity of the owner of corresponding private key
  - Verifying signature does **not** verify the identity of the source of certificate!
  - Certificates are public!
  - Big problem if CA makes a mistake (a CA once issued Microsoft certificate to someone else!)
  - Common format for certificates is X.509

PKI

- Public Key Infrastructure (PKI) consists of all pieces needed to securely use public key cryptography
  - Key generation and management
  - Certificate authorities
  - Certificate revocation (CRLs), etc.
- No general standard for PKI
- We consider a few "trust models"

PKI Trust Models

- Monopoly model
  - One universally trusted organization is the *CA* for the known universe
  - Favored by VeriSign (for obvious reasons)
  - Big problems if *CA* is ever compromised
  - Big problem if you don't trust the *CA*!

PKI Trust Models

- Oligarchy
  - Multiple trusted CAs
  - This approach used in browsers today
  - Browser may have 80 or more certificates, just to verify signatures!
  - User can decide which CAs to trust

PKI Trust Models

- Anarchy model
  - Everyone is a CA!
  - Users must decide which "CAs" to trust
  - This approach used in PGP (Web of trust)
  - Why do they call it "anarchy"? Suppose cert. is signed by Frank and I don't know Frank, but I do trust Bob and Bob says Alice is trustworthy and Alice vouches for Frank. Should I trust Frank?
- Many other PKI trust models

Confidentiality in the Real World

Symmetric Key vs Public Key

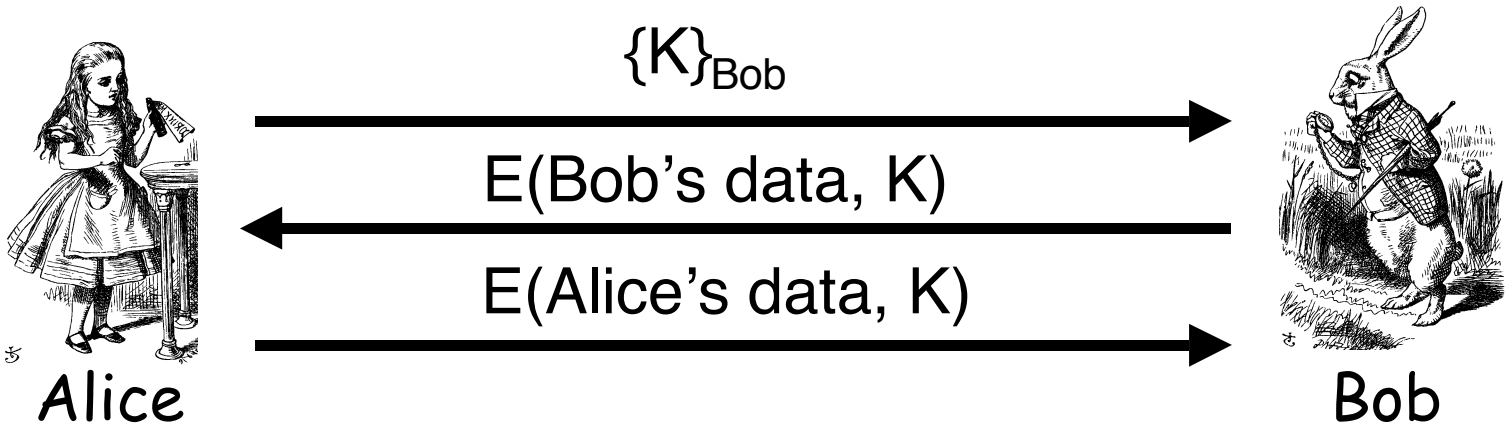
- Symmetric key +'s
  - **Speed**
  - No public key infrastructure (PKI) needed
- Public Key +'s
  - **Signatures** (non-repudiation)
  - No shared secret

Notation Reminder

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



- 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 $M = \{S\}_{Alice}$
  - Is it OK to just send S?
- If M is big, $[M]_{Alice}$ is costly to compute
- Suppose instead, Alice signs h(M), where h(M) is much smaller than M
  - Alice sends M and $S = [h(M)]_{Alice}$ to Bob
  - Bob verifies that $h(M) = \{S\}_{Alice}$

Crypto Hash Function

- Crypto hash function h(x) must provide
  - **Compression** — output length is small
  - **Efficiency** — h(x) easy to computer 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 ≠ x such that h(y) = h(x)
  - **Strong collision resistance** — infeasible to find any x and y, with x ≠ y such that h(x) = h(y)
  - Lots of collisions exist, but hard to find one

Pre-Birthday Problem

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

Birthday Problem

- How many people must be in a room before probability is ≥ 1/2 that two or more have same birthday?
  - 1 – 365/365 · 364/365 · · ·(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

Of Hashes and Birthdays

- If h(x) is N bits, then 2^N different hash values are possible
- sqrt(2^N) = $2^{N/2}$
- Therefore, hash about $2^{N/2}$ random values and you expect to find a collision
- **Implication:** secure N bit symmetric key requires 2^{N-1} work to "break" while secure N bit hash requires $2^{N/2}$ work to "break"

Non-crypto Hash (1)

- Data X = $(X_0,X_1,X_2,\ldots,X_{n-1})$, each X_i is a byte
- Spse hash(X) = $X_0+X_1+X_2+\ldots+X_{n-1}$
- Is this secure?
- Example: X = (10101010,00001111)
- Hash is 10111001
- But so is hash of Y = (00001111,10101010)
- 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
  - $h(X) = nX_0 + (n-1)X_1 + (n-2)X_2 + \ldots + 1 \cdot X_{n-1}$
- Is this hash secure?
- At least
  - $h(10101010,00001111) \neq h(00001111,10101010)$
- But hash of (00000001,00001111) is same as hash of (00000000,00010001)
- Not one-way, 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 problem
- Good for detecting burst **errors**
- But easy to construct collisions
- CRC sometimes mistakenly used in crypto applications (WEP)

Popular Crypto Hashes

- **MD5** — invented by Rivest
  - 128 bit output
  - Note: MD5 collision recently found
- **SHA-1** — A US government standard (similar to MD5)
  - 160 bit output
- Many others hashes, but MD5 and SHA-1 most widely used
- Hashes work by hashing message in blocks

Crypto Hash Design

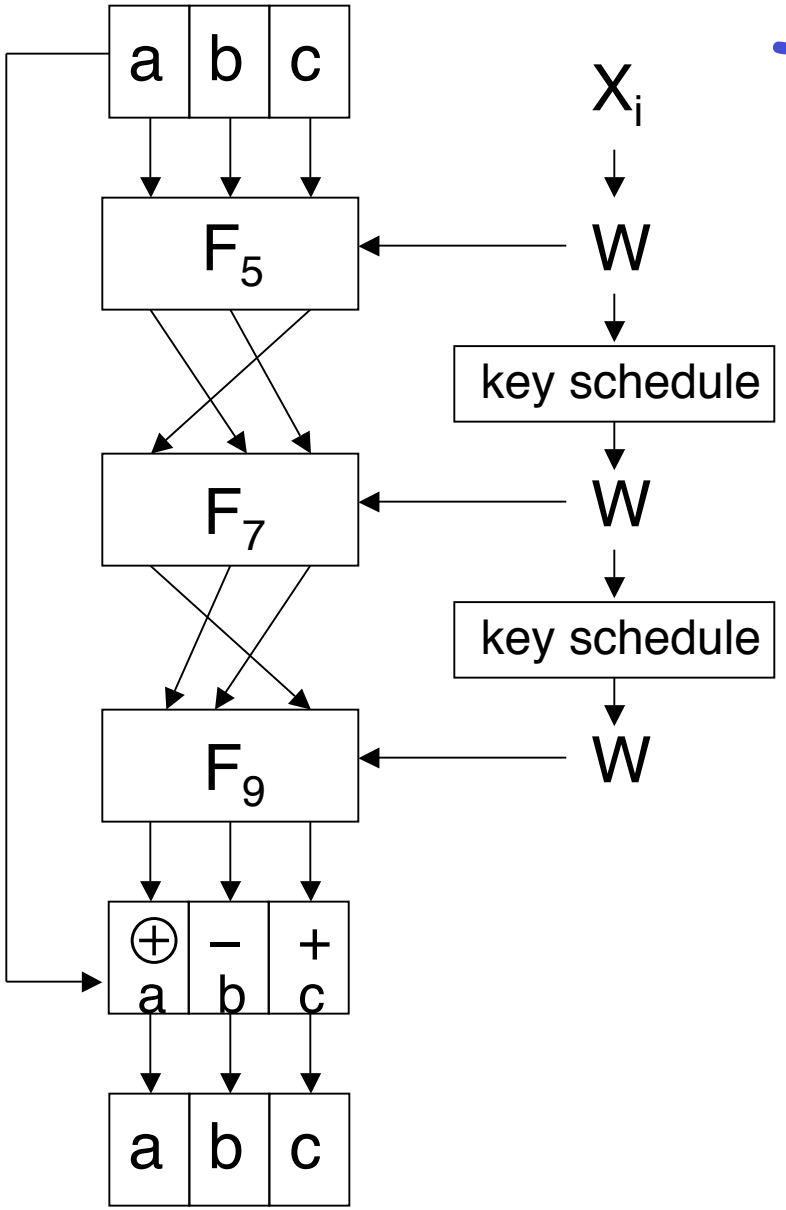
- Desired property: **avalanche effect**
  - Change to 1 bit of input should affect about half of output bits
- Crypto hash functions consist of some number of rounds
- Want security and speed
  - Avalanche effect after few rounds
  - But simple rounds
- Analogous to design of block ciphers

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

Tiger Hash

- Like MD5/SHA-1, input divided into 512 bit blocks (padded)
- Unlike MD5/SHA-1, output is **192 bits** (three 64-bit words)
  - Truncate output if replacing MD5 or SHA-1
- Intermediate rounds are all 192 bits
- 4 S-boxes, each maps 8 bits to 64 bits
- A "key schedule" is used

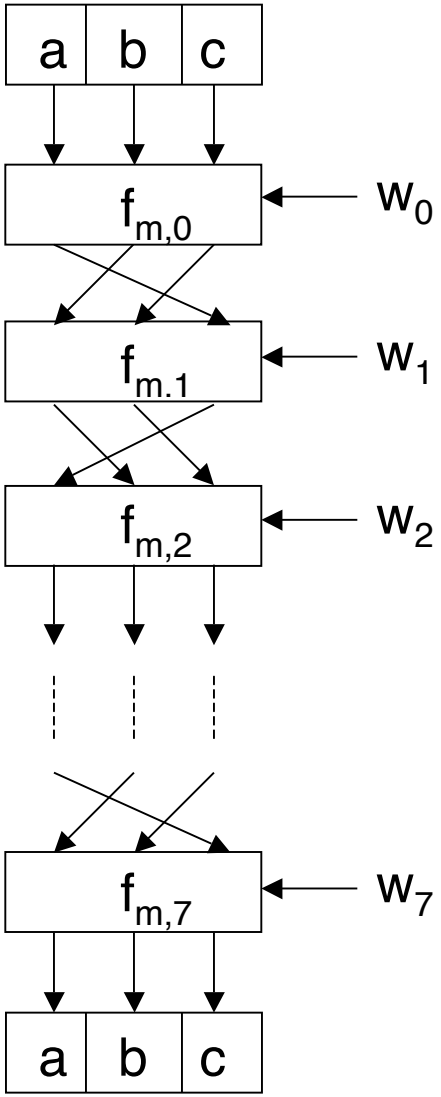


Tiger Outer Round

- Input is X
  - $X = (X_0, X_1, \ldots, X_{n-1})$
  - X is padded
  - Each X_i is 512 bits
- There are n iterations of diagram at left
  - One for each input block
- Initial (a,b,c) constants
- Final (a,b,c) is hash
- Looks like block cipher!

Tiger Inner Rounds

- Each F_m consists of precisely **8 rounds**
- 512 bit input W to F_m
  - $W=(w_0,w_1,\ldots,w_7)$
  - W is one of the input blocks X_i
- All lines are 64 bits
- The $f_{m,i}$ depend on the S-boxes (next slide)



Tiger Hash: One Round

- Each $f_{m,i}$ is a function of a,b,c,w_i and m
  - Input values of a,b,c from previous round
  - And w_i is 64-bit block of 512 bit W
  - Subscript m is multiplier
  - And $c = (c_0,c_1,\ldots,c_7)$
- Output of $f_{m,i}$ is
  - $c = c \oplus w_i$
  - $a = a - (S_0[c_0] \oplus S_1[c_2] \oplus S_2[c_4] \oplus S_3[c_6])$
  - $b = b + (S_3[c_1] \oplus S_2[c_3] \oplus S_1[c_5] \oplus S_0[c_7])$
  - $b = b * m$
- Each S_i is **S-box**: 8 bits mapped to 64 bits

Tiger Hash Key Schedule

- Input is X
  - $X=(x_0,x_1,\ldots,x_7)$
- Small change in X will produce large change in key schedule output

$x_0 = x_0 - (x_7 \oplus 0xA5A5A5A5A5A5A5A5)$
$x_1 = x_1 \oplus x_0$
$x_2 = x_2 + x_1$
$x_3 = x_3 - (x_2 \oplus ((\sim x_1) << 19))$
$x_4 = x_4 \oplus x_3$
$x_5 = x_5 + x_4$
$x_6 = x_6 - (x_5 \oplus ((\sim x_4) >> 23))$
$x_7 = x_7 \oplus x_6$
$x_0 = x_0 + x_7$
$x_1 = x_1 - (x_0 \oplus ((\sim x_7) << 19))$
$x_2 = x_2 \oplus x_1$
$x_3 = x_3 + x_2$
$x_4 = x_4 - (x_3 \oplus ((\sim x_2) >> 23))$
$x_5 = x_5 \oplus x_4$
$x_6 = x_6 + x_5$
$x_7 = x_7 - (x_6 \oplus 0x0123456789ABCDEF)$

Tiger Hash Summary (1)

- Hash and intermediate values are 192 bits
- 24 rounds
  - **S-boxes:** Claimed that each input bit affects a, b and c after 3 rounds
  - **Key schedule:** Small change in message affects many bits of intermediate hash values
  - **Multiply:** Designed to insure that input to S-box in one round mixed into many S-boxes in next
- S-boxes, key schedule and multiply together designed to insure strong **avalanche** effect

Tiger Hash Summary (2)

- Uses lots of ideas from block ciphers
  - S-boxes
  - Multiple rounds
  - Mixed mode arithmetic
- At a higher level, Tiger employs
  - Confusion
  - Diffusion

HMAC

- Can compute a MAC of M with key K using a "hashed MAC" or **HMAC**
- HMAC is an example of a keyed hash
  - Why do we need a key?
- How to compute HMAC?
- Two obvious choices
  - h(K,M)
  - h(M,K)

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)

The Right 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
- ipad = 0x36 repeated B times
- opad = 0x5C repeated B times
- Then

  HMAC(M,K) = H(K ⊕ opad, H(K ⊕ ipad, M))

Hash Uses

- Authentication (HMAC)
- Message integrity (HMAC)
- Message fingerprint
- Data corruption detection
- Digital signature efficiency
- Anything you can do with symmetric crypto

Online Auction

- 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
- Solution?
  - Alice, Bob, Charlie submit **hashes** h(A), h(B), h(C)
  - All hashes received and posted online
  - Then bids A, B and C revealed
- Hashes don't reveal bids (one way)
- Can't change bid after hash sent (collision)

Spam Reduction

- Spam reduction
- Before I accept an email from you, I want proof that you spent "effort" (e.g., CPU cycles) to create the email
- Limit amount of email that can be sent
- Make spam much more costly to send

Spam Reduction

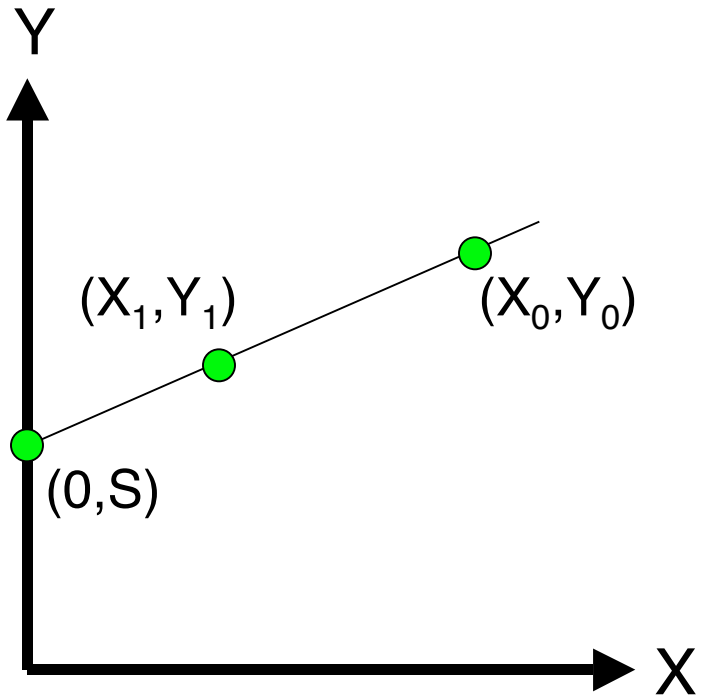
- Let M = email message
- Let **R** = value to be determined
- Let T = current time
- Sender must find **R** such that
  - hash(M,**R**,T) = (00…0,X), where
  - N initial bits of hash are **all zero**
- Sender then sends (M,**R**,T)
- Recipient accepts email, provided
  - hash(M,**R**,T) begins with N zeros

Spam Reduction

- Sender: hash(M,R,T) begins with N zeros
- Recipient: verify that hash(M,R,T) begins with N zeros
- **Work for sender:** about **2^N hashes**
- **Work for recipient: 1 hash**
- Sender's work increases exponentially in N
- Same work for recipient regardless of N
- Choose N so that
  - Work acceptable for normal email users
  - Work unacceptably high for spammers!

Secret Sharing

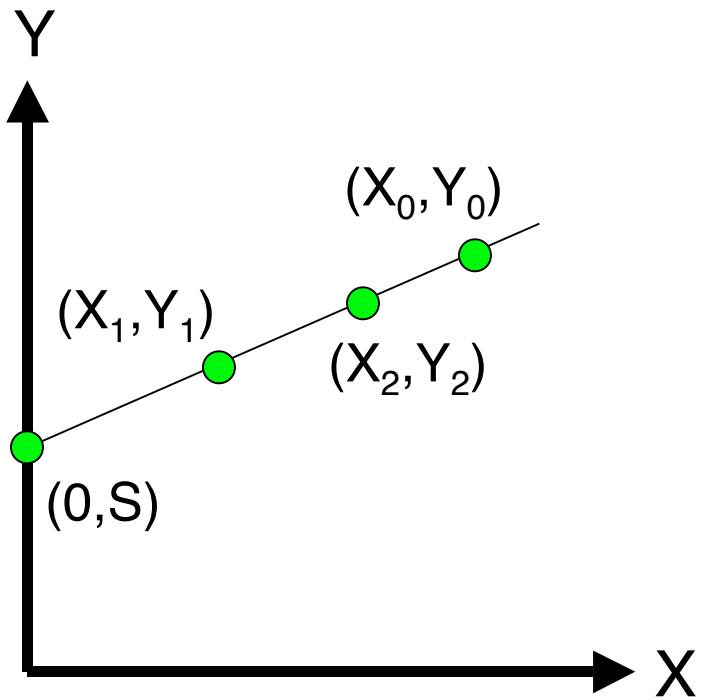
Shamir's Secret Sharing



2 out of 2

- 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 $m \leq n$

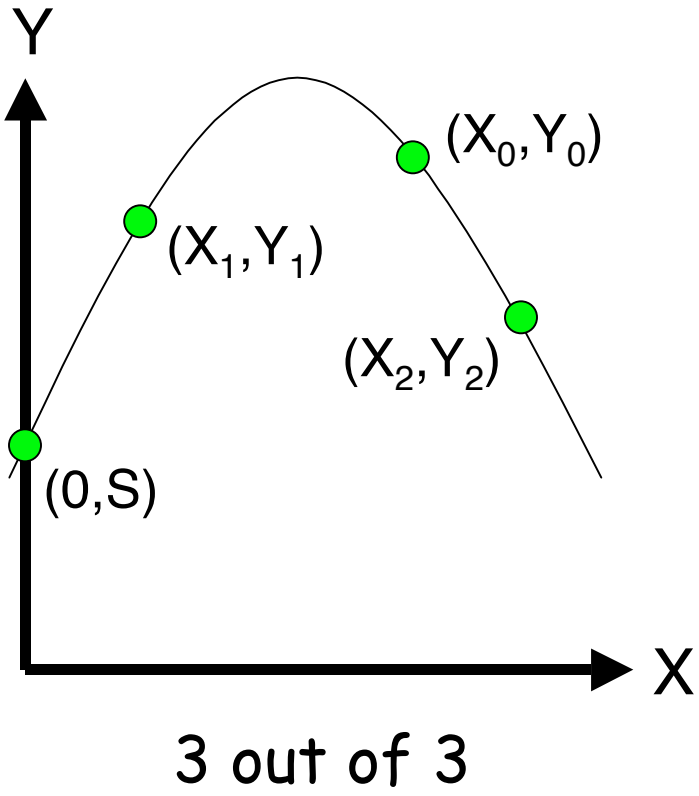
Shamir's Secret Sharing



2 out of 3

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

Shamir's Secret Sharing



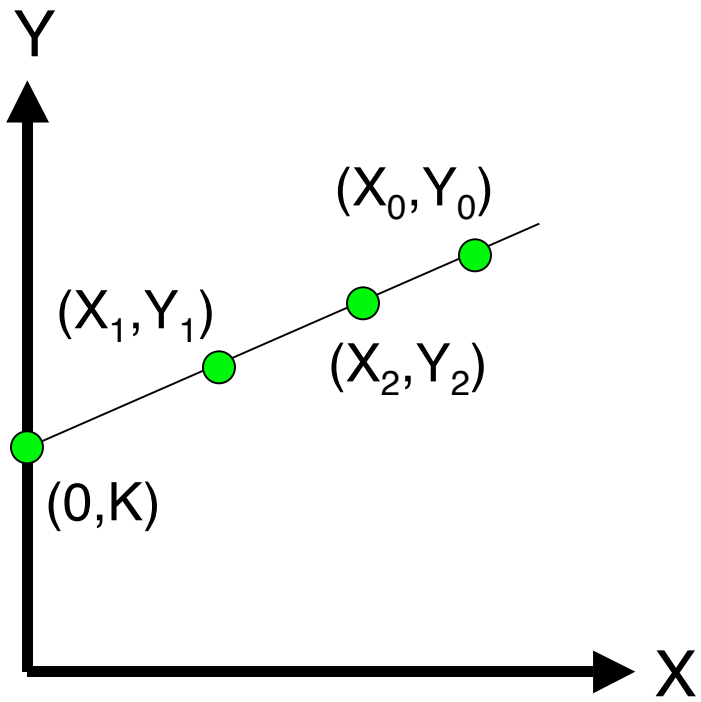
3 out of 3

- Give (X_0, Y_0) to Alice
- Give (X_1, Y_1) to Bob
- Give (X_2, Y_2) to Charlie
- 3 points determine a parabola
- Alice, Bob **and** Charlie must cooperate to find secret S
- A "3 out of 3" scheme
- Can you make a "3 out of 4"?

Secret Sharing Example

- **Key escrow** — required that your key be stored somewhere
- Key can be used with court order
- But you don't trust FBI to store 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

Random Numbers in Cryptography

Random Numbers

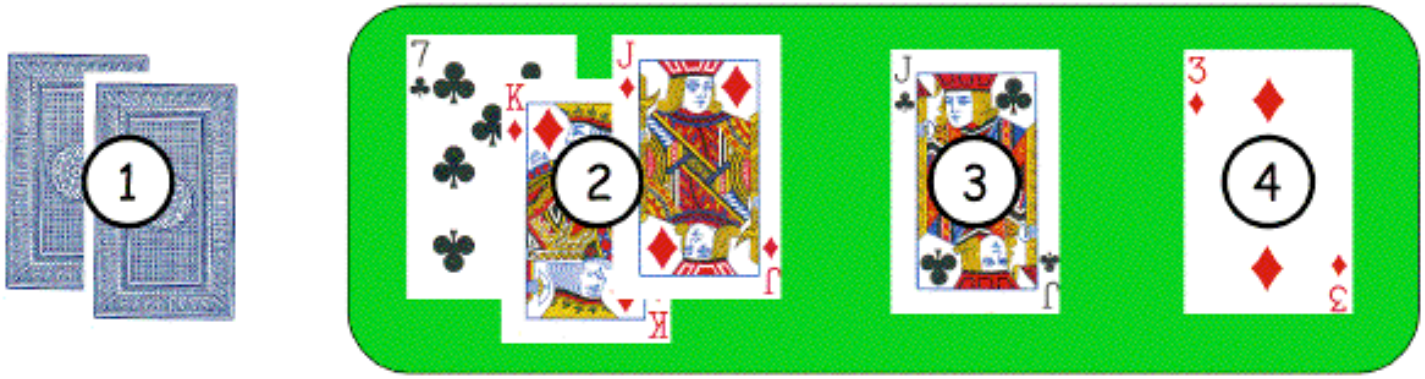
- Random numbers used to generate **keys**
  - Symmetric keys
  - RSA: Prime numbers
  - Diffie Hellman: secret values
- Random numbers used for nonces
  - Sometimes a sequence is OK
  - But sometimes nonces must be random
- Random numbers also used in simulations, statistics, etc., where numbers only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

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
- Spse Alice, Bob and Charlie don't like Dave
- Alice, Bob and Charlie working together must **not** be able to determine K_D

Bad Random Number Example

- 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
- Could determine the shuffle in real time!

Card Shuffle

- There are $52! > 2^{225}$ possible shuffles
- The poker program used "random" 32-bit integer to determine the shuffle
  - Only 2^{32} distinct shuffles could occur
- 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
  - Therefore, 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 try 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 sequence is not predictable
  - For example, keystream from RC4 cipher
- But "seed" (or key) selection is still an issue!
- How to generate initial **random** values?
  - Applies to both keys and seeds

Randomness

- True randomness is hard to define
- **Entropy** is a measure of randomness
- Good sources of "true" randomness
  - Radioactive decay — though 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 (hopefully) deterministic
  - So must rely on external "random" events
  - Mouse movements, keyboard dynamics, network activity, etc., etc.
- Can get **quality** random bits via software
- But **quantity** of such 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" identifier to data
  - Defense against music or software piracy
- Steganography
  - Secret communication channel
  - A kind of **covert channel**
  - Example: Hide data in image or music file

Watermark

- Add a "mark" to data
- Several types of watermarks
  - Invisible — Not obvious the mark exists
  - Visible — Such as **TOP SECRET**
  - Robust — Readable even if attacked
  - Fragile — Mark destroyed if attacked

Watermark

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

Watermark Example (1)

- US currency includes watermark

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

Watermark Example (2)

- Add **invisible** watermark to photo print
- It is claimed that 1 square inch can contain enough info to reconstruct entire photo
- If photo is damaged, watermark can be read from an undamaged section and entire photo can be reconstructed!

Steganography

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

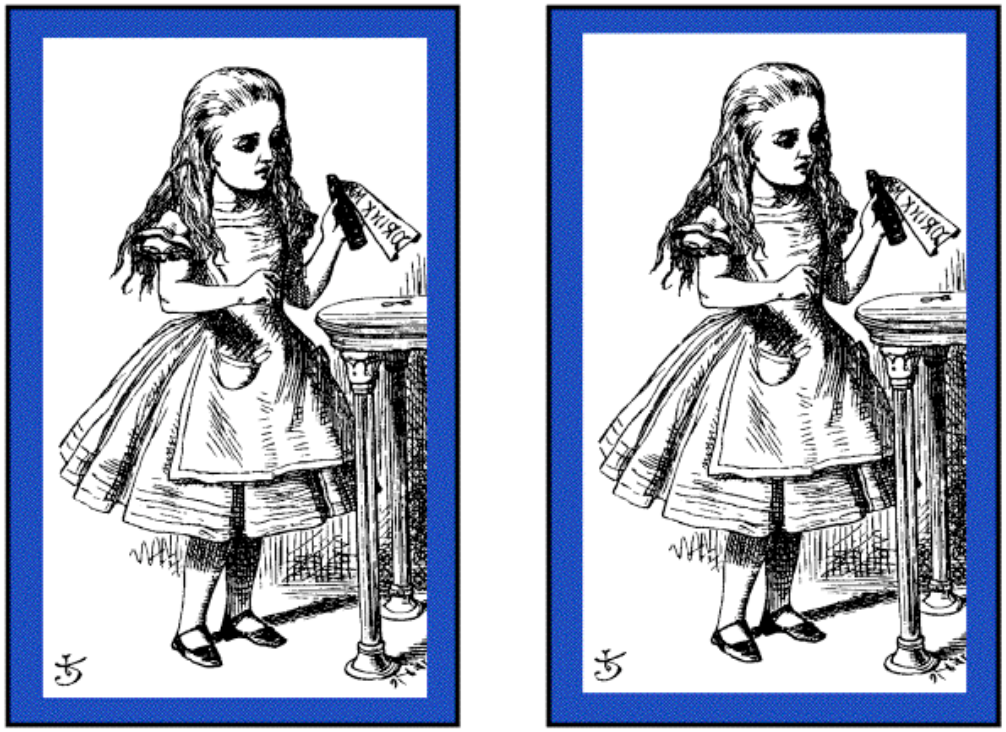
Images and Steganography

- 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 are unimportant!

Images and Stego

- Given an uncompressed image file
  - For example, BMP format
- Then we can insert any information into low-order RGB bits
- Since low-order RGB bits don't matter, result will be "invisible" to human eye
- But a 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 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

```
<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>
<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

```
<font color="#010100">"The time has come," the Walrus said,</font><br>
<font color="#000100">"To talk of many things:</font><br>
<font color="#010100">Of shoes and ships and sealing wax</font><br>
<font color="#000101">Of cabbages and kings</font><br>
<font color="#000000">And why the sea is boiling hot</font><br>
<font color="#010001">And whether pigs have wings."</font><br>
```

- "Hidden" message: **110 010 110 011 000 101**

Steganography

- Some formats (jpg, gif, wav, etc.) are more difficult (than html) for humans to read
- Easy to hide information in **unimportant bits**
- Easy to **destroy** or remove info stored in unimportant bits!
- To be robust, information must be stored in **important bits**
- But stored information must not damage data!
- Collusion attacks also a major concern
- Robust steganography is trickier than it seems

Information Hiding
The Bottom Line

- Surprisingly difficult to hide digital information: "obvious" approach **not** robust
  - **Stirmark** makes most watermarks in jpg images unreadable — **without** damaging the image
  - Watermarking is very active research area
- If information hiding is suspected
  - Attacker can probably make information/watermark unreadable
  - Attacker may be able to read the information, given the original document (image, audio, etc.)

Advanced Cryptanalysis

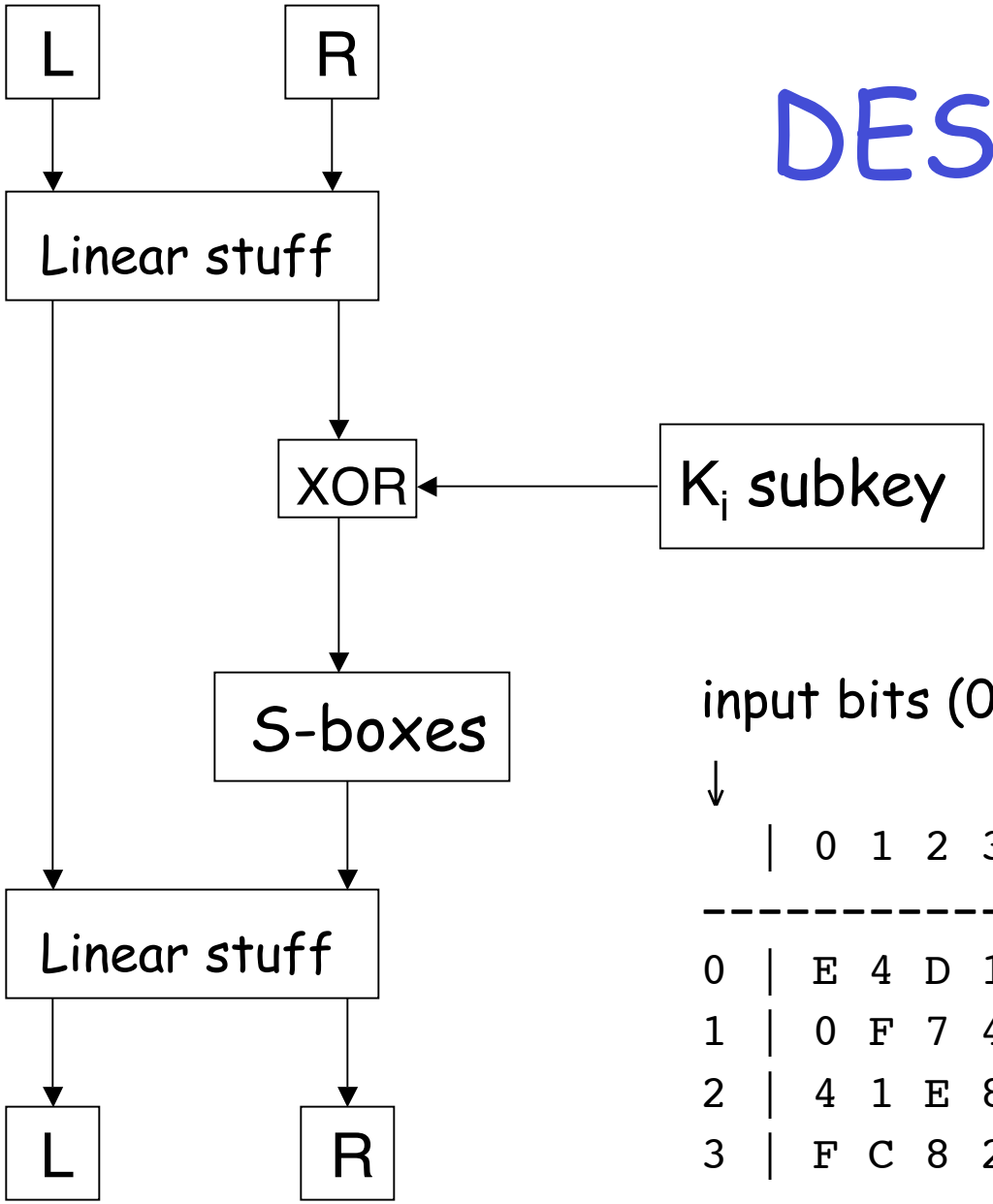
Advanced Cryptanalysis

- Modern cryptanalysis
  - Differential cryptanalysis
  - Linear cryptanalysis
- Side channel attack on RSA
- Lattice reduction attack on knapsack
- Hellman's TMTO attack on DES

Linear and Differential Cryptanalysis

Introduction

- Both linear and differential cryptanalysis developed to attack DES
- Applicable to other block ciphers
- Differential — Biham and Shamir, 1990
  - Apparently known to NSA in 1970's
  - For analyzing ciphers, not a practical attack
  - A chosen plaintext attack
- Linear cryptanalysis — Matsui, 1993
  - Perhaps not know to NSA in 1970's
  - Slightly more feasible than differential cryptanalysis
  - A known plaintext attack



DES Overview

- 8 S-boxes
- Each S-box maps 6 bits to 4 bits
- Example: S-box 1

input bits (0,5)

↓ input bits (1,2,3,4)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	E	4	D	1	2	F	B	8	3	A	6	C	5	9	0	7
1	0	F	7	4	E	2	D	1	A	6	C	B	9	5	3	4
2	4	1	E	8	D	6	2	B	F	C	9	7	3	A	5	0
3	F	C	8	2	4	9	1	7	5	B	3	E	A	0	6	D

Overview of Differential Cryptanalysis

Differential Cryptanalysis

- Consider DES
- All of DES is linear except S-boxes
- Differential attack focuses on nonlinearity
- Idea is to compare input and output **differences**
- For simplicity, first consider one round and one S-box

Differential Cryptanalysis

- Spse DES-like cipher has 3 to 2 bit S-box

	column			
row	00	01	10	11
0	10	01	11	00
1	00	10	01	11

- Sbox(abc) is element in row a column bc
- Example: Sbox(010) = 11

Differential Cryptanalysis

	column			
row	00	01	10	11
0	10	01	11	00
1	00	10	01	11

- Suppose $X_1 = 110$, $X_2 = 010$, $K = 011$
- Then $X_1 \oplus K = 101$ and $X_2 \oplus K = 001$
- $Sbox(X_1 \oplus K) = 10$ and $Sbox(X_2 \oplus K) = 01$

	column			
row	00	01	10	11
0	10	01	11	00
1	00	10	01	11

Differential Cryptanalysis

- Suppose
  - **Unknown:** K
  - **Known:** X = 110, X = 010
  - **Known:** Sbox(X ⊕ K) = 10, Sbox(X ⊕ K) = 01
- Know X ⊕ K ∈ {000,101}, X ⊕ K ∈ {001,110}
- Then K ∈ {110,011} ∩ {011,100} ⇒ K = 011
- Like a known plaintext attack on S-box

Differential Cryptanalysis

- Attacking one S-box not very useful!
  - And Trudy can't always see input and output
- To make this work we must do 2 things

1. Extend the attack to **one round**
   - Must account for all S-boxes
   - Choose input so only one S-box "active"
2. Then extend attack to (almost) **all rounds**
   - Note that output is input to next round
   - Choose input so output is "good" for next round

Differential Cryptanalysis

- We deal with input and output differences
- Suppose we know inputs X and X
  - For X the input to S-box is X ⊕ K
  - For X the input to S-box is X ⊕ K
  - Key K is unknown
  - **Input difference**: (X ⊕ K) ⊕ (X ⊕ K) = X ⊕ X
- Input difference is independent of key K
- **Output difference**: Y ⊕ Y is (almost) input difference to next round
- Goal is to "chain" differences thru rounds

Differential Cryptanalysis

- If we obtain known output difference from known input difference…
  - May be able to chain differences thru rounds
  - It's OK if this only occurs with some probability
- If input difference is 0…
  - …output difference is 0
  - Allows us to make some S-boxes "inactive" with respect to differences

S-box Differential Analysis

- Input diff 000 not interesting
- Input diff 010 always gives output diff 01
- More biased, the better (for Trudy)

row	column 00	01	10	11
0	10	01	11	00
1	00	10	01	11

X ⊕ X

	Sbox(X)⊕Sbox(X) 00	01	10	11
000	8	0	0	0
001	0	0	4	4
010	0	8	0	0
011	0	0	4	4
100	0	0	4	4
101	4	4	0	0
110	0	0	4	4
111	4	4	0	0

Overview of Linear Cryptanalysis

Linear Cryptanalysis

- Like differential cryptanalysis, we target the nonlinear part of the cipher
- But instead of differences, we approximate the nonlinearity with **linear equations**
- For DES-like cipher we need to approximate S-boxes by linear functions
- How well can we do this?

S-box Linear Analysis

- Input $x_0x_1x_2$ where x_0 is row and x_1x_2 is column
- Output y_0y_1
- Count of 4 is unbiased
- Count of 0 or 8 is best for Trudy

row	column 00	01	10	11
0	10	01	11	00
1	00	10	01	11

		output y_0	y_1	$y_0 \oplus y_1$
	0	4	4	4
i	x_0	4	4	4
n	x_1	4	6	2
p	x_2	4	4	4
u	$x_0 \oplus x_1$	4	2	2
t	$x_0 \oplus x_2$	0	4	4
	$x_1 \oplus x_2$	4	6	6
	$x_0 \oplus x_1 \oplus x_2$	4	6	2

Linear Analysis

- For example,
  $y_1 = x_1$
  with prob. 3/4
- And
  $y_0 = x_0 \oplus x_2 \oplus 1$
  with prob. 1
- And
  $y_0 \oplus y_1 = x_1 \oplus x_2$
  with prob. 3/4

row	column 00	01	10	11
0	10	01	11	00
1	00	10	01	11

input	output y_0	y_1	$y_0 \oplus y_1$
0	4	4	4
x_0	4	4	4
x_1	4	6	2
x_2	4	4	4
$x_0 \oplus x_1$	4	2	2
$x_0 \oplus x_2$	0	4	4
$x_1 \oplus x_2$	4	6	6
$x_0 \oplus x_1 \oplus x_2$	4	6	2

(input: i n p u t)

Linear Cryptanalysis

- Consider a single DES S-box
- Let Y = Sbox(X)
- Suppose $y_3 = x_2 \oplus x_5$ with high probability
  - This is a linear approximation to output y_3
- Can we extend this so that we can solve linear equations for the key?
- As in differential cryptanalysis, we need to "chain" thru multiple rounds

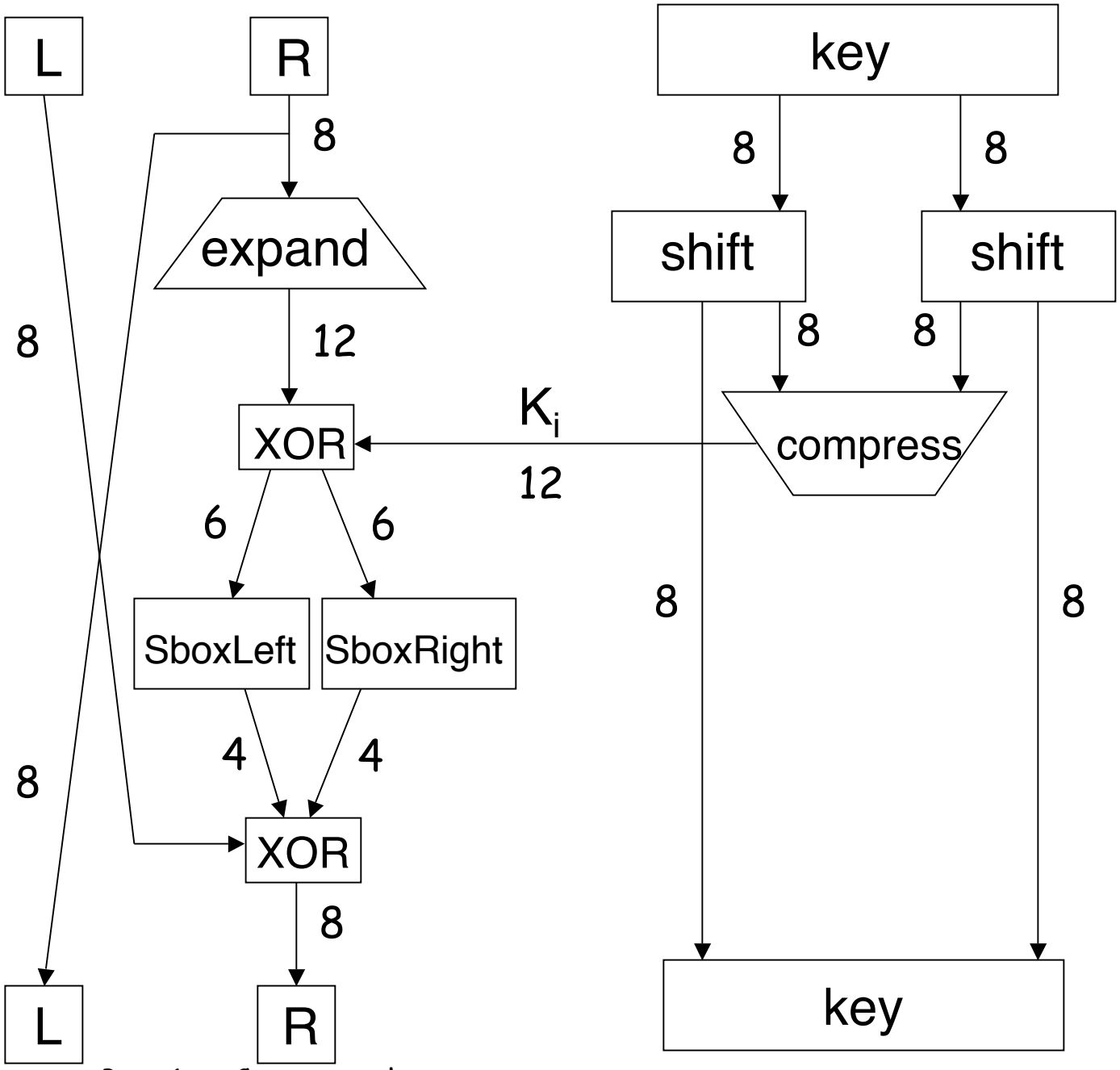
Linear Cryptanalysis of DES

- DES is linear except for S-boxes
- How well can we approximate S-boxes with linear functions?
- DES S-boxes designed so there are no good linear approximations to any one output bit
- But there **are** linear combinations of output bits that can be approximated by linear combinations of input bits

Tiny DES

Tiny DES (TDES)

- A much simplified version of DES
  - 16 bit block
  - 16 bit key
  - 4 rounds
  - 2 S-boxes, each maps 6 bits to 4 bits
  - 12 bit subkey each round
- Plaintext = (L_0,R_0)
- Ciphertext = (L_4,R_4)
- No useless junk



One Round of TDES

TDES Fun Facts

- TDES is a Feistel Cipher
- (L_0,R_0) = plaintext
- For i = 1 to 4

  $L_i = R_{i-1}$

  $R_i = L_{i-1} \oplus F(R_{i-1},K_i)$
- Ciphertext = (L_4,R_4)
- $F(R_{i-1}, K_i)$ = Sboxes(expand(R_{i-1}) $\oplus$ K_i)

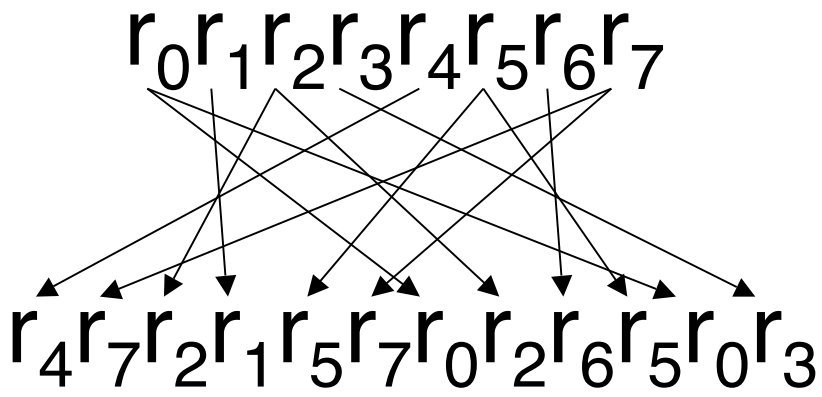
  where Sboxes($x_0x_1x_2\ldots x_{11}$) = (SboxLeft($x_0x_1\ldots x_5$),SboxRight($x_6x_7\ldots x_{11}$))

TDES Key Schedule

- Key: K = $k_0k_1k_2k_3k_4k_5k_6k_7k_8k_9k_{10}k_{11}k_{12}k_{13}k_{14}k_{15}$
- Subkey
  - Left: $k_0k_1\ldots k_7$ rotate left 2, select 0,2,3,4,5,7
  - Right: $k_8k_9\ldots k_{15}$ rotate left 1, select 9,10,11,13,14,15
- Subkey K_1 = $k_2k_4k_5k_6k_7k_1k_{10}k_{11}k_{12}k_{14}k_{15}k_8$
- Subkey K_2 = $k_4k_6k_7k_0k_1k_3k_{11}k_{12}k_{13}k_{15}k_8k_9$
- Subkey K_3 = $k_6k_0k_1k_2k_3k_5k_{12}k_{13}k_{14}k_8k_9k_{10}$
- Subkey K_4 = $k_0k_2k_3k_4k_5k_7k_{13}k_{14}k_{15}k_9k_{10}k_{11}$

TDES expansion perm

- Expansion permutation: 8 bits to 12 bits



- We can write this as

  $\text{expand}(r_0r_1r_2r_3r_4r_5r_6r_7) = r_4r_7r_2r_1r_5r_7r_0r_2r_6r_5r_0r_3$

TDES S-boxes

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	C	5	0	A	E	7	2	8	D	4	3	9	6	F	1	B
1	1	C	9	6	3	E	B	2	F	8	4	5	D	A	0	7
2	F	A	E	6	D	8	2	4	1	7	9	0	3	5	B	C
3	0	A	3	C	8	2	1	E	9	7	F	6	B	5	D	4

- Right S-box
- SboxRight

- Left S-box
- SboxLeft

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	6	9	A	3	4	D	7	8	E	1	2	B	5	C	F	0
1	9	E	B	A	4	5	0	7	8	6	3	2	C	D	1	F
2	8	1	C	2	D	3	E	F	0	9	5	A	4	B	6	7
3	9	0	2	5	A	D	6	E	1	8	B	C	3	4	7	F

Differential Cryptanalysis of TDES

TDES

- TDES SboxRight

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	C	5	0	A	E	7	2	8	D	4	3	9	6	F	1	B
1	1	C	9	6	3	E	B	2	F	8	4	5	D	A	0	7
2	F	A	E	6	D	8	2	4	1	7	9	0	3	5	B	C
3	0	A	3	C	8	2	1	E	9	7	F	6	B	5	D	4

- For X and X suppose $X \oplus X = 001000$
- Then SboxRight(X) $\oplus$ SboxRight(X) = 0010 with probability 3/4

Differential Crypt. of TDES

- The game plan…
- Select P and P so that

  $P \oplus P = 0000\ 0000\ 0000\ 0010 = 0x0002$
- Note that P and P differ in exactly 1 bit
- Let's carefully analyze what happens as these plaintexts are encrypted with TDES

TDES

- If Y ⊕ Y = 001000 then with probability 3/4 SboxRight(Y) ⊕ SboxRight(Y) = 0010
- Y⊕Y = 001000 ⇒ (Y⊕K)⊕(Y⊕K) = 001000
- If Y ⊕ Y = 000000 then for any S-box, Sbox(Y) ⊕ Sbox(Y) = 0000
- Difference of (0000 0010) is expanded by TDES expand perm to diff. (000000 001000)
- **The bottom line:** If X ⊕ X = 00000010 then F(X,K) ⊕ F(X,K) = 00000010 with prob. 3/4

TDES

- From the previous slide
  - Suppose R ⊕ R = 0000 0010
  - Suppose K is unknown key
  - Then with probability 3/4
    F(R,K) ⊕ F(R,K) = 0000 0010
- The bottom line
  - Input to next round is like input to current round
  - Maybe we can chain this thru multiple rounds!

TDES Differential Attack

- Select P and P' with $P \oplus P' = 0x0002$

$(L_0,R_0) = P$	$(L_0,R_0) = P'$	$P \oplus P' = 0x0002$
$L_1 = R_0$ $R_1 = L_0 \oplus F(R_0,K_1)$	$L_1 = R_0$ $R_1 = L_0 \oplus F(R_0,K_1)$	With probability 3/4 $(L_1,R_1) \oplus (L_1,R_1) = 0x0202$
$L_2 = R_1$ $R_2 = L_1 \oplus F(R_1,K_2)$	$L_2 = R_1$ $R_2 = L_1 \oplus F(R_1,K_2)$	With probability $(3/4)^2$ $(L_2,R_2) \oplus (L_2,R_2) = 0x0200$
$L_3 = R_2$ $R_3 = L_2 \oplus F(R_2,K_3)$	$L_3 = R_2$ $R_3 = L_2 \oplus F(R_2,K_3)$	With probability $(3/4)^2$ $(L_3,R_3) \oplus (L_3,R_3) = 0x0002$
$L_4 = R_3$ $R_4 = L_3 \oplus F(R_3,K_4)$	$L_4 = R_3$ $R_4 = L_3 \oplus F(R_3,K_4)$	With probability $(3/4)^3$ $(L_4,R_4) \oplus (L_4,R_4) = 0x0202$
$C = (L_4,R_4)$	$C = (L_4,R_4)$	$C \oplus C = 0x0202$

TDES Differential Attack

- Choose P and P with $P \oplus P = 0x0002$
- If $C \oplus C = 0x0202$ then

  $R_4 = L_3 \oplus F(R_3,K_4)$ $\quad$ $R_4 = L_3 \oplus F(R_3,K_4)$
  $R_4 = L_3 \oplus F(L_4,K_4)$ $\quad$ $R_4 = L_3 \oplus F(L_4,K_4)$

  and $(L_3,R_3) \oplus (L_3,R_3) = 0x0002$
- Then $L_3 = L_3$ and $C=(L_4,R_4)$ and $C=(L_4,R_4)$ are both known
- Since $L_3 = R_4 \oplus F(L_4,K_4)$ and $L_3 = R_4 \oplus F(L_4,K_4)$, for correct subkey K_4 we have

  $R_4 \oplus F(L_4,K_4) = R_4 \oplus F(L_4,K_4)$

TDES Differential Attack

- Choose P and P with $P \oplus P = 0x0002$
- If $C \oplus C = (L_4, R_4) \oplus (L_4, R_4) = 0x0202$
- Then for the correct subkey K_4

  $R_4 \oplus F(L_4,K_4) = R_4 \oplus F(L_4,K_4)$

  which we rewrite as

  $R_4 \oplus R_4 = F(L_4,K_4) \oplus F(L_4,K_4)$

  where the only unknown is K_4
- Let $L_4 = I_0I_1I_2I_3I_4I_5I_6I_7$. Then we have

  $0010 = \text{SBoxRight}(I_0I_2I_6I_5I_0I_3 \oplus k_{13}k_{14}k_{15}k_9k_{10}k_{11})$

  $\oplus \text{SBoxRight}(I_0I_2I_6I_5I_0I_3 \oplus k_{13}k_{14}k_{15}k_9k_{10}k_{11})$

TDES Differential Attack

Algorithm to find right 6 bits of subkey K_4

```
count[i] = 0, for i = 0,1,. . .,63
for i = 1 to iterations
    Choose P and P with P ⊕ P = 0x0002
    Obtain corresponding C and C
    if C ⊕ C = 0x0202
        for K = 0 to 63
            if 0010 == (SBoxRight( I0I2I6I5I0I3 ⊕K)⊕SBoxRight( I0I2I6I5I0I3 ⊕K))
                ++count[K]
            end if
        next K
    end if
next i
```

All K with max count[K] are possible (partial) K_4

TDES Differential Attack

- Computer program results
- Choose 100 pairs P and P with P⊕ P= 0x0002
- Found 47 of these give C ⊕ C = 0x0202
- Tabulated counts for these 47
  - Max count of 47 for each
    K ∈ {000001,001001,110000,111000}
  - No other count exceeded 39
- Implies that K_4 is one of 4 values, that is,

  $k_{13}k_{14}k_{15}k_9k_{10}k_{11} \in \{000001,001001,110000,111000\}$
- Actual key is K=1010 1001 1000 0111

Linear Cryptanalysis of TDES

Linear Approx. of Left S-Box

- TDES left S-box or SboxLeft

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	6	9	A	3	4	D	7	8	E	1	2	B	5	C	F	0
1	9	E	B	A	4	5	0	7	8	6	3	2	C	D	1	F
2	8	1	C	2	D	3	E	F	0	9	5	A	4	B	6	7
3	9	0	2	5	A	D	6	E	1	8	B	C	3	4	7	F

- Notation: $y_0y_1y_2y_3 = \text{SboxLeft}(x_0x_1x_2x_3x_4x_5)$
- For this S-box, $y_1=x_2$ and $y_2=x_3$ both with probability 3/4
- Can we "chain" this thru multiple rounds?

TDES Linear Relations

- Recall that the expansion perm is expand($r_0r_1r_2r_3r_4r_5r_6r_7$) = $r_4r_7\mathbf{r_2r_1}r_5r_7r_0r_2r_6r_5r_0r_3$
- And $y_0y_1y_2y_3$ = SboxLeft($x_0x_1x_2x_3x_4x_5$) with $y_1=x_2$ and $y_2=x_3$ each with probability 3/4
- Also, expand(R_{i-1}) $\oplus$ K_i is input to Sboxes at round i
- Then $y_1=r_2\oplus k_m$ and $y_2=r_1\oplus k_n$ both with prob 3/4
- New right half is $y_0y_1y_2y_3$… plus old left half
- **Bottom line:** New right half bits: $r_1 \leftarrow r_2 \oplus k_m \oplus l_1$ and $r_2 \leftarrow r_1 \oplus k_n \oplus l_2$ both with probability 3/4

Recall TDES Subkeys

- Key: $K = k_0k_1k_2k_3k_4k_5k_6k_7k_8k_9k_{10}k_{11}k_{12}k_{13}k_{14}k_{15}$
- Subkey $K_1 = k_2k_4\mathbf{k_5k_6}k_7k_1k_{10}k_{11}k_{12}k_{14}k_{15}k_8$
- Subkey $K_2 = k_4k_6\mathbf{k_7k_0}k_1k_3k_{11}k_{12}k_{13}k_{15}k_8k_9$
- Subkey $K_3 = k_6k_0\mathbf{k_1k_2}k_3k_5k_{12}k_{13}k_{14}k_8k_9k_{10}$
- Subkey $K_4 = k_0k_2k_3k_4k_5k_7k_{13}k_{14}k_{15}k_9k_{10}k_{11}$

TDES Linear Cryptanalysis

- Known $P=p_0p_1p_2\ldots p_{15}$ and $C=c_0c_1c_2\ldots c_{15}$

	Bit 1, Bit 2 (numbering from 0)	probability
$(L_0,R_0) = (p_0\ldots p_7,p_8\ldots p_{15})$		
$L_1 = R_0$	p_9, p_{10}	1
$R_1 = L_0 \oplus F(R_0,K_1)$	$p_1\oplus p_{10}\oplus k_5$, $p_2\oplus p_9\oplus k_6$	3/4
$L_2 = R_1$	$p_1\oplus p_{10}\oplus k_5$, $p_2\oplus p_9\oplus k_6$	3/4
$R_2 = L_1 \oplus F(R_1,K_2)$	$p_2\oplus k_6\oplus k_7$, $p_1\oplus k_5\oplus k_0$	$(3/4)^2$
$L_3 = R_2$	$p_2\oplus k_6\oplus k_7$, $p_1\oplus k_5\oplus k_0$	$(3/4)^2$
$R_3 = L_2 \oplus F(R_2,K_3)$	$p_{10}\oplus k_0\oplus k_1$, $p_9\oplus k_7\oplus k_2$	$(3/4)^3$
$L_4 = R_3$	$p_{10}\oplus k_0\oplus k_1$, $p_9\oplus k_7\oplus k_2$	$(3/4)^3$
$R_4 = L_3 \oplus F(R_3,K_4)$		
	$k_0 \oplus k_1 = c_1 \oplus p_{10}$	$(3/4)^3$
$C = (L_4,R_4)$	$k_7 \oplus k_2 = c_2 \oplus p_9$	$(3/4)^3$

TDES Linear Cryptanalysis

- Computer program results
- Use 100 known plaintexts, get ciphertexts.
  - Let $P=p_0p_1p_2\ldots p_{15}$ and let $C=c_0c_1c_2\ldots c_{15}$
- Resulting counts
  - $c_1 \oplus p_{10} = 0$ occurs 38 times
  - $c_1 \oplus p_{10} = 1$ occurs 62 times
  - $c_2 \oplus p_9 = 0$ occurs 62 times
  - $c_2 \oplus p_9 = 1$ occurs 38 times
- Conclusions
  - Since $k_0 \oplus k_1 = c_1 \oplus p_{10}$ we have $k_0 \oplus k_1 = 1$
  - Since $k_7 \oplus k_2 = c_2 \oplus p_9$ we have $k_7 \oplus k_2 = 0$
- Actual key is K = 1010 0011 0101 0110

To Build a Better Block Cipher…

- How can cryptographers make linear and differential attacks more difficult?
  1. **More rounds** — success probabilities diminish with each round
  2. **Better confusion** (S-boxes) — reduce success probability on each round
  3. **Better diffusion** (permutations) — more difficult to chain thru multiple rounds
- Limited mixing and limited nonlinearity, with more rounds required: TEA
- Strong mixing and nonlinearity, with fewer but more complex rounds: AES

Side Channel Attack on RSA

Side Channel Attacks

- Sometimes possible to recover key without directly attacking the crypto algorithm
- A **side channel** consists of "incidental information"
- Side channels can arise due to
  - The way that a computation is performed
  - Media used, power consumed, unintended emanations, etc.
- Induced faults can also reveal information
- Side channel may reveal a crypto key
- Paul Kocher is the leader in this field

Side Channels

- Emanations security (EMSEC)
  - Electromagnetic field (EMF) from computer screen can allow screen image to be reconstructed at a distance
  - Smartcards have been attacked via EMF emanations
- Differential power analysis (DPA)
  - Smartcard power usage depends on the computation
- Differential fault analysis (DFA)
  - Key stored on smartcard in GSM system could be read using a flashbulb to induce faults
- Timing analysis
  - Different computations take different time
  - RSA keys recovered over a network (openSSL)!

The Scenario

- Alice's public key: (N,e)
- Alice's private key: d
- Trudy wants to find d
- Trudy can send any message M to Alice and Alice will respond with M^d mod N
- Trudy can precisely time Alice's computation of M^d mod N

Timing Attack on RSA

- Consider $M^d \bmod N$
- We want to find **private key** d, where $d = d_0 d_1 \ldots d_n$
- Spse repeated squaring used for $M^d \bmod N$
- Suppose, for efficiency

```
mod(x,N)
if x >= N
    x = x % N
end if
return x
```

Repeated Squaring

```
x = M
for j = 1 to n
    x = mod(x^2,N)
    if d_j == 1 then
        x = mod(x*M,N)
    end if
next j
return x
```

Timing Attack

- If $d_j = 0$ then
  - $x = \text{mod}(x^2, N)$
- If $d_j = 1$ then
  - $x = \text{mod}(x^2, N)$
  - $x = \text{mod}(x * M, N)$
- Computation time differs in each case
- Can attack or take advantage of this?

Repeated Squaring

```
x = M
for j = 1 to n
    x = mod(x^2,N)
    if d_j == 1 then
        x = mod(x*M,N)
    end if
next j
return x
```

mod(x,N)

```
if x >= N
    x = x % N
end if
return x
```

Timing Attack

- Choose M with $M^3 < N$
- Choose M with $M^2 < N < M^3$
- Let x = M and x = M
- Consider j = 1
  - x = mod(x^2,N) does no "%"
  - x = mod(x∗M,N) does no "%"
  - x = mod(x^2,N) does no "%"
  - x = mod(x∗M,N) does "%" **only if** d_1=1
- If d_1 = 1 then j = 1 step takes longer for M than for M
- But more than one round…

Repeated Squaring

```
x = M
for j = 1 to n
    x = mod(x^2,N)
    if d_j == 1 then
        x = mod(x*M,N)
    end if
next j
return x
```

mod(x,N)

```
if x >= N
    x = x % N
end if
return x
```

Timing Attack on RSA

- "Chosen plaintext" attack
- Choose $M_0, M_1, \ldots, M_{m-1}$ with
  - $M_i^3 < N$ for $i=0,1,\ldots,m-1$
- Let t_i be time to compute $M_i^d \bmod N$
  - $t = (t_0 + t_1 + \ldots + t_{m-1}) / m$
- Choose $M_0, M_1, \ldots, M_{m-1}$ with
  - $M_i^2 < N < M_i^3$ for $i=0,1,\ldots,m-1$
- Let t_i be time to compute $M_i^d \bmod N$
  - $t = (t_0 + t_1 + \ldots + t_{m-1}) / m$
- If $t > t$ then $d_1 = 1$ otherwise $d_1 = 0$
- Once d_1 is known, similar approach to find $d_2, d_3, \ldots$

Side Channel Attacks

- If crypto is secure Trudy looks for shortcut
- What is good crypto?
  - More than mathematical analysis of algorithms
  - Many other issues (such as side channels) must be considered
  - See Schneier's article
- Lesson: **Attacker's don't play by the rules!**

Knapsack Lattice Reduction Attack

Lattice?

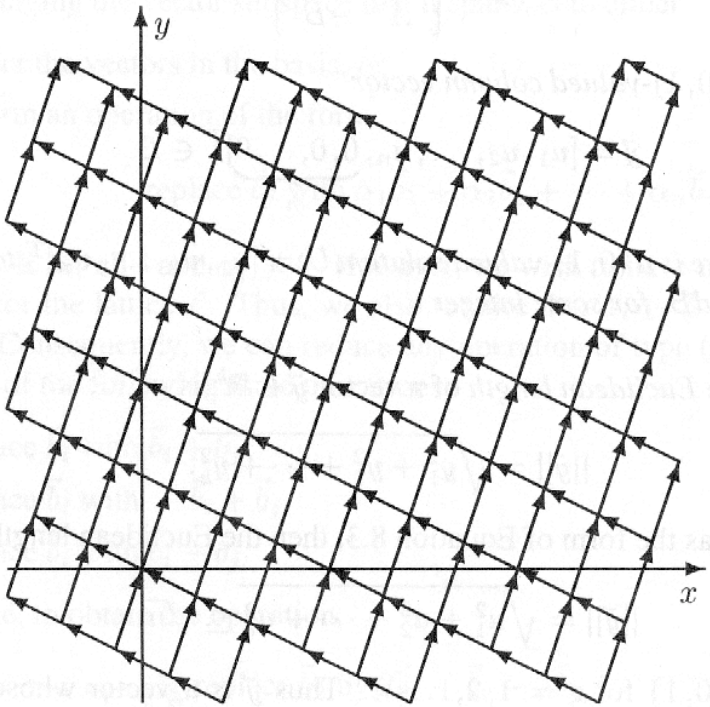
- Many problems can be solved by finding a "short" vector in a **lattice**
- Let $b_1, b_2, \ldots, b_n$ be vectors in $\Re^m$
- All $\alpha_1 b_1 + \alpha_2 b_2 + \ldots + \alpha_n b_n$, each α_i is an integer is a discrete set of points

What is a Lattice?

- Suppose $b_1=[1,3]^T$ and $b_2=[-2,1]^T$
- Then any point in the plane can be written as $\alpha_1 b_1+\alpha_2 b_2$ for some $\alpha_1,\alpha_2 \in \Re$
  - Since b_1 and b_2 are linearly independent
- We say the plane $\Re^2$ is spanned by (b_1,b_2)
- If α_1,α_2 are restricted to **integers**, the resulting span is a **lattice**
- Then a lattice is a discrete set of points

Lattice Example

- Suppose $b_1=[1,3]^T$ and $b_2=[-2,1]^T$
- The lattice spanned by (b_1,b_2) is pictured to the right



Exact Cover

- **Exact cover** — given a set S and a collection of subsets of S, find a collection of these subsets with each element of S is in exactly one subset
- Exact Cover is a combinatorial problems that can be solved by finding a "short" vector in lattice

Exact Cover Example

- Set S = {0,1,2,3,4,5,6}
- Spse m = 7 elements and n = 13 subsets

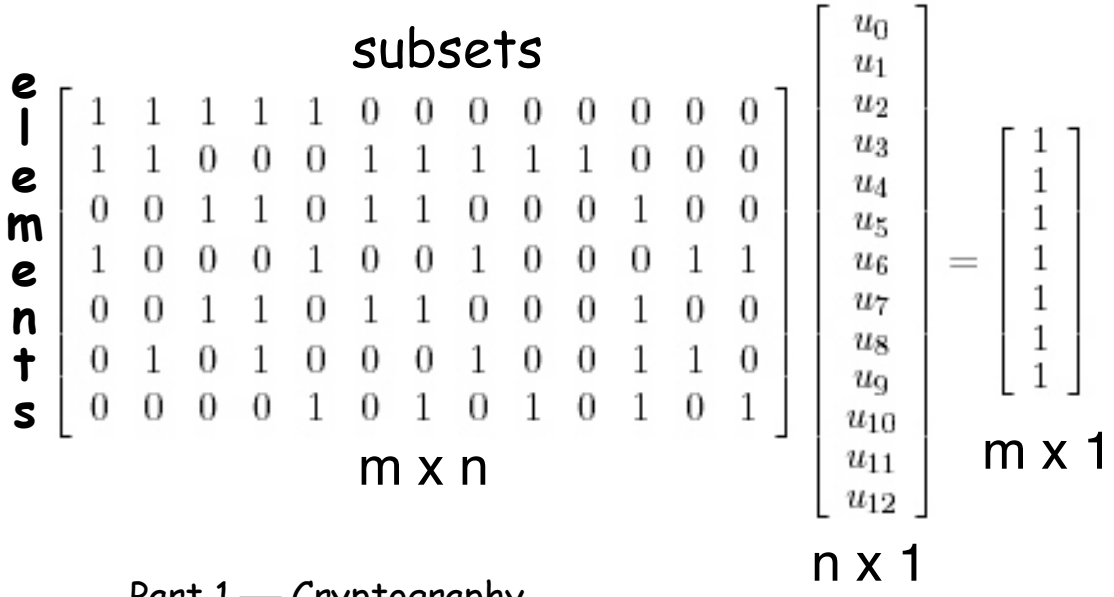
Subset:	0	1	2	3	4	5	6	7	8	9	10	11	12
Elements:	013	015	024	025	036	124	126	135	146	1	256	345	346

- Find a collection of these subsets with each element of S in exactly one subset
- Could try all 2^{13} possibilities
- If problem is too big, try **heuristic search**
- Many different heuristic search techniques

Exact Cover Solution

- Exact cover in matrix form
  - Set S = {0,1,2,3,4,5,6}
  - Spse m = 7 elements and n = 13 subsets

Subset:	0	1	2	3	4	5	6	7	8	9	10	11	12
Elements:	013	015	024	025	036	124	126	135	146	1	256	345	346



Solve: AU = B

where $u_i \in \{0,1\}$

Solution:

$U = [0001000001001]^T$

Example

- We can restate AU = B as MV = W where

$$\begin{bmatrix} I_{n\times n} & 0_{n\times 1} \\ A_{m\times n} & -B_{m\times 1} \end{bmatrix} \begin{bmatrix} U_{n\times 1} \\ 1_{1\times 1} \end{bmatrix} = \begin{bmatrix} U_{n\times 1} \\ 0_{m\times 1} \end{bmatrix} \iff AU = B$$

Matrix M Vector V Vector W

- The desired solution is U
  - Columns of M are **linearly independent**
- Let $c_0, c_1, c_2, \ldots, c_n$ be the columns of M
- Let $v_0, v_1, v_2, \ldots, v_n$ be the elements of V
- Then $W = v_0c_0 + v_1c_1 + \ldots + v_nc_n$

Example

- Let L be the lattice spanned by $c_0, c_1, c_2, \ldots, c_n$ (c_i are the columns of M)
- Recall MV = W
  - Where $W = [U,0]^T$ and we want to find U
  - But if we find W, we've also solved it!
- Note W is in lattice L since all v_i are integers and $W = v_0c_0 + v_1c_1 + \ldots + v_nc_n$

Facts

- $W = [u_0, u_1, \ldots, u_{n-1}, 0, 0, \ldots, 0] \in L$, each $u_i \in \{0,1\}$
- The length of a vector $Y \in \Re^N$ is

  $$\|Y\| = \text{sqrt}(y_0^2 + y_1^2 + \ldots + y_{N-1}^2)$$
- Then the length of W is

  $$\|W\| = \text{sqrt}(u_0^2 + u_1^2 + \ldots + u_{n-1}^2) \leq \text{sqrt}(n)$$
- So W is a very **short** vector in L where
  - First n entries of W all 0 or 1
  - Last m elements of W are all 0
- Can we use these facts to find U?

Lattice Reduction

- If we can find a short vector in L, with first n entries all 0 or 1 and last m entries all 0, then we *might* have found U
- LLL lattice reduction algorithm will efficiently find short vectors in a lattice
- Less than 30 lines of pseudo-code for LLL!
- No guarantee LLL will find a specific vector
- But probability of success is often good

Knapsack Example

- What does lattice reduction have to do with the knapsack cryptosystem?
- Suppose we have
  - Superincreasing knapsack

    S = [2,3,7,14,30,57,120,251]
  - Suppose m = 41, n = 491 $\Rightarrow$ $m^{-1} = 12 \bmod n$
  - Public knapsack: $t_i = 41 \cdot s_i \bmod 491$

    T = [82,123,287,83,248,373,10,471]
- **Public key:** T **Private key:** (S,m^{-1},n)

Knapsack Example

- **Public key:** T **Private key:** (S, m^{-1}, n)

  S = [2,3,7,14,30,57,120,251]

  T = [82,123,287,83,248,373,10,471]

  n = 491, $m^{-1} = 12$

- Example: 10010110 is encrypted as

  82+83+373+10 = 548

- Then receiver computes

  $548 \cdot 12 = 193 \bmod 491$

  and uses S to solve for 10010110

Knapsack LLL Attack

- Attacker knows public key

  T = [82,123,287,83,248,373,10,471]

- Attacker knows ciphertext: 548

- Attacker wants to find $u_i \in \{0,1\}$ s.t.

$$82u_0+123u_1+287u_2+83u_3+248u_4+373u_5+10u_6+471u_7=548$$

- This can be written as a matrix equation (dot product): $T \cdot U = 548$

Knapsack LLL Attack

- Attacker knows: T = [82,123,287,83,248,373,10,471]
- Wants to solve: $T \cdot U = 548$ where each $u_i \in \{0,1\}$
  - Same form as AU = B on previous slides!
  - We can rewrite problem as MV = W where

$$M = \begin{bmatrix} I_{8\times 8} & 0_{8\times 1} \\ T_{1\times 8} & -C_{1\times 1} \end{bmatrix} = \left[\begin{array}{cccccccc|c} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ \hline 82 & 123 & 287 & 83 & 248 & 373 & 10 & 471 & -548 \end{array}\right]$$

- LLL gives us short vectors in the lattice spanned by the columns of M

LLL Result

- LLL finds short vectors in lattice of M
- Matrix M′ is result of applying LLL to M

$$M' = \left[\begin{array}{cccc|cccc|c} & & & * & & & & & \\ -1 & -1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & -1 & 1 & 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & -1 & 1 & 2 \\ 1 & -1 & -1 & 1 & 0 & -1 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & -2 & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & -1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & -1 \\ \hline 1 & -1 & 1 & 0 & 0 & 1 & -1 & 2 & 0 \end{array}\right]$$

- Column marked with "*" has the right form
- Possible solution: U = $[1,0,0,1,0,1,1,0]^T$
- Easy to verify this is the plaintext!

Bottom Line

- Lattice reduction is a surprising method of attack on knapsack
- A cryptosystem is only secure as long as nobody has found an attack
- Lesson: **Advances in mathematics can break cryptosystems!**

Hellman's TMTO Attack

Popcnt

- Before we consider Hellman's attack, consider a simple Time-Memory TradeOff
- "Population count" or popcnt
  - Let x be a 32-bit integer
  - Define $\text{popcnt}(x)$ = number of 1's in binary expansion of x
  - How to compute $\text{popcnt}(x)$ efficiently?

Simple Popcnt

- Most obvious thing to do is

```
popcnt(x) // assuming x is 32-bit value
        t = 0
        for i = 0 to 31
                t = t + ((x >> i) & 1)
        next i
        return t
end popcnt
```

- But is it the most efficient?

More Efficient Popcnt

- Precompute popcnt for all 256 bytes
- Store precomputed values in a table
- Given x, lookup its bytes in this table
  - Sum these values to find $\text{popcnt}(x)$
- Note that precomputation is done once
- Each popcnt now requires 4 steps, not 32

More Efficient Popcnt

Initialize: table[*i*] = popcnt(*i*) for *i* = 0,1,…,255

popcnt(*x*) // assuming *x* is 32-bit value
    p = table[*x* & 0xff]
        + table[(*x* >> 8) & 0xff]
        + table[(*x* >> 16) & 0xff]
        + table[(*x* >> 24) & 0xff]
    return *p*
end popcnt

TMTO Basics

- A precomputation
  - One-time work
  - Results stored in a table
- Precomputation results used to make each subsequent computation faster
- Balancing "memory" and "time"
- In general, larger precomputation requires more initial work and larger "memory" but each subsequent computation is less "time"

Block Cipher Notation

- Consider a block cipher

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

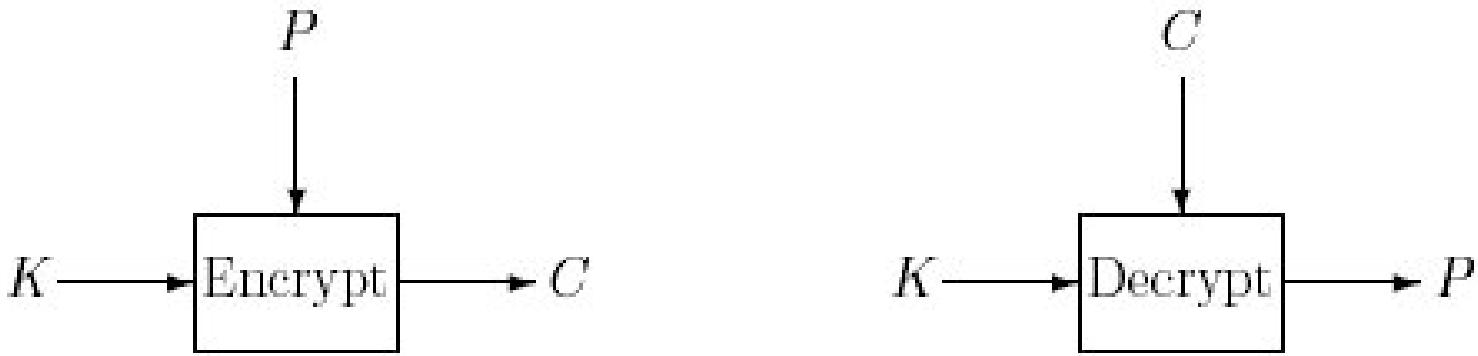
where

P is plaintext block of size n

C is ciphertext block of size n

K is key of size k

Block Cipher as Black Box



- For TMTO, treat block cipher as black box
- Details of crypto algorithm not important

Hellman's TMTO Attack

- **Chosen plaintext attack:** choose P and obtain C, where $C = E(P, K)$
- Want to find the key K
- Two "obvious" approaches
  1. Exhaustive key search
     - "Memory" is 0, but "time" of 2^{k-1} for each attack
  2. Pre-compute $C = E(P, K)$ for all possible K
     - Then given C, can simply look up key K in the table
     - "Memory" of 2^k but "time" of 0 for each attack
- TMTO lies between 1. and 2.

Chain of Encryptions

- Assume block and key lengths equal: $n = k$
- Then a **chain** of encryptions is

$$SP = K_0 = \text{Starting Point}$$

$$K_1 = E(P, SP)$$

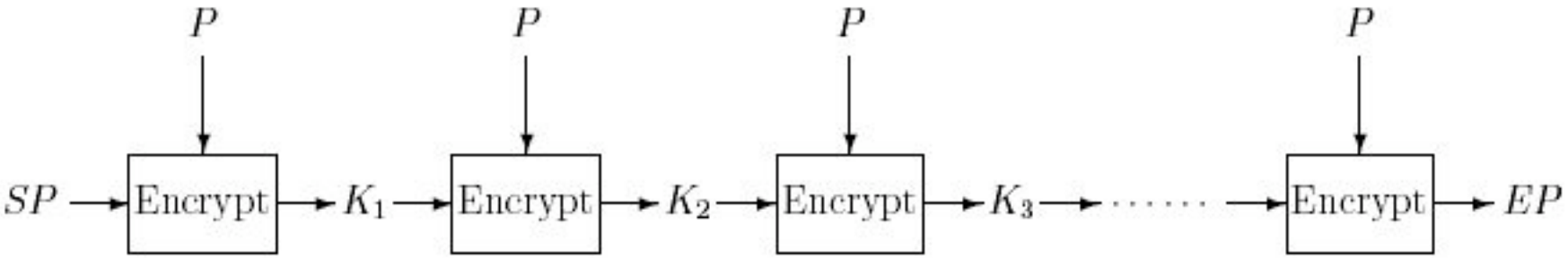
$$K_2 = E(P, K_1)$$

$$\vdots$$

$$\vdots$$

$$EP = K_t = E(P, K_{t-1}) = \text{End Point}$$

Encryption Chain



- Ciphertext used as **key** at next iteration
- Same (chosen) **plaintext** at each iteration

Pre-computation

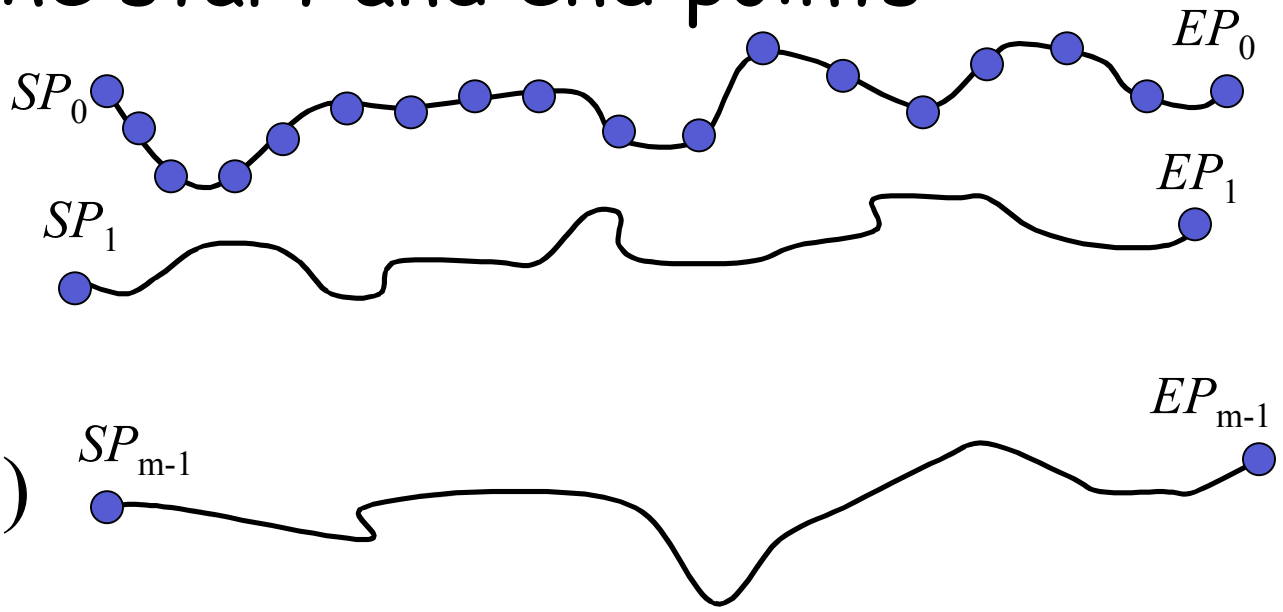
- Pre-compute m encryption chains, each of length $t+1$
- Save only the start and end points

(SP_0, EP_0)

(SP_1, EP_1)

$\vdots$

(SP_{m-1}, EP_{m-1})



TMTO Attack

- **Memory:** Pre-compute encryption chains and save (SP_i, EP_i) for $i = 0,1,\ldots,m-1$
  - This is one-time work
- Then to attack a particular unknown key K
  - For the same chosen P used to find chains, we know C where $C = E(P, K)$ and K is unknown key
  - **Time:** Compute the chain (maximum of t steps)

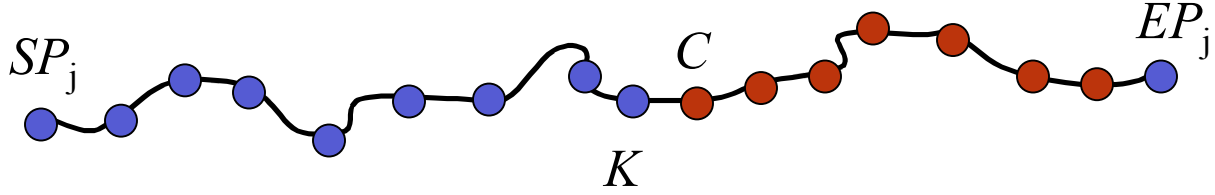
$$X_0 = C,\ X_1 = E(P, X_0),\ X_2 = E(P, X_1),\ldots$$

TMTO Attack

- Consider the computed chain

$$X_0 = C, X_1 = E(P, X_0), X_2 = E(P, X_1), \ldots$$

- Suppose for some i we find $X_i = EP_j$



- Since $C = E(P, K)$ key K before C in chain!

TMTO Attack

- To summarize, we compute chain

  $X_0 = C, X_1 = E(P, X_0), X_2 = E(P, X_1), \ldots$

- If for some i we find $X_i = EP_j$
- Then reconstruct chain from SP_j

  $Y_0 = SP_j, Y_1 = E(P, Y_0), Y_2 = E(P, Y_1), \ldots$

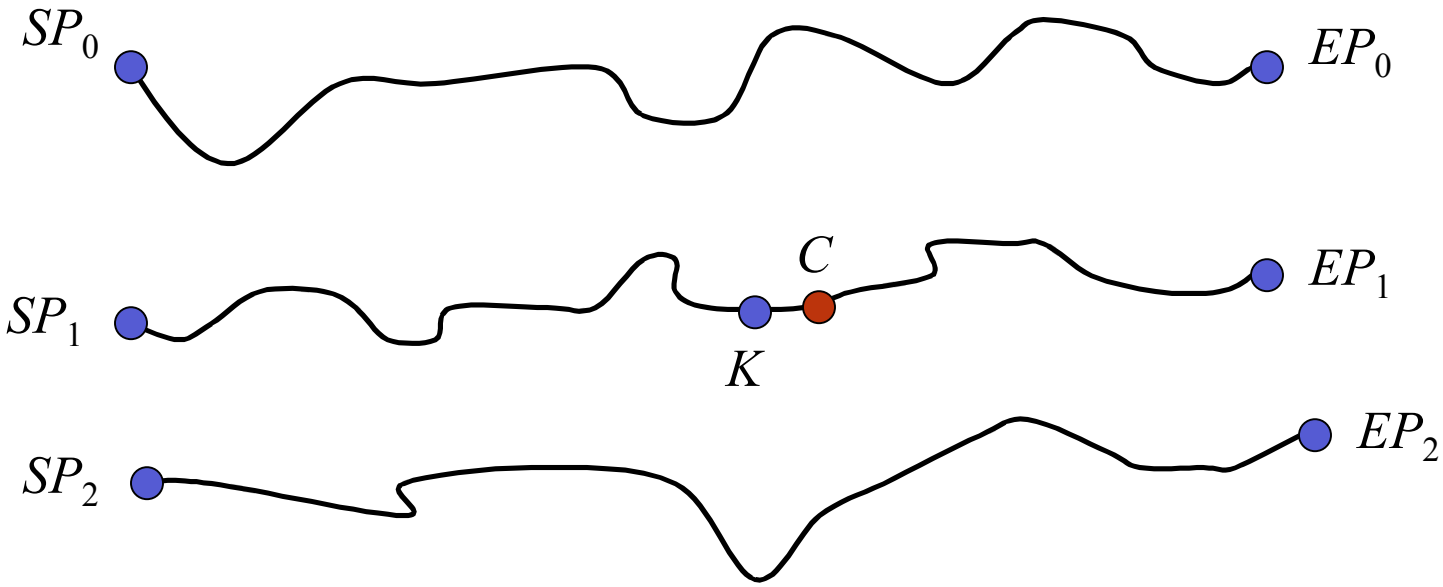
- Find $C = Y_{t-i} = E(P, Y_{t-i-1})$ (always?)
- Then $K = Y_{t-i-1}$ (always?)

Trudy's Perfect World

- Suppose block cipher has $k = 56$
  - That is, the key length is 56 bits
- Suppose we find $m = 2^{28}$ chains, each of length $t = 2^{28}$ and no chains overlap
- **Memory:** 2^{28} pairs (SP_j, EP_i)
- **Time:** about 2^{28} (per attack)
  - Start at C, find some EP_j in about 2^{27} steps
  - Find K with about 2^{27} more steps
- Attack never fails!

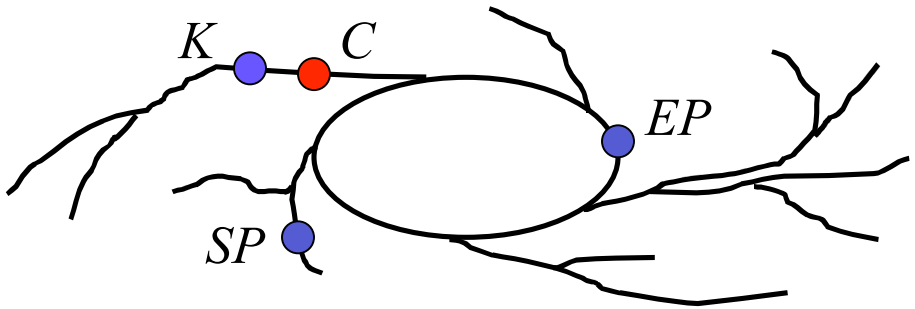
Trudy's Perfect World

- No chains overlap
- Any ciphertext C is in some chain



The Real World

- Chains are not so well-behaved!
- Chains can **cycle** and **merge**



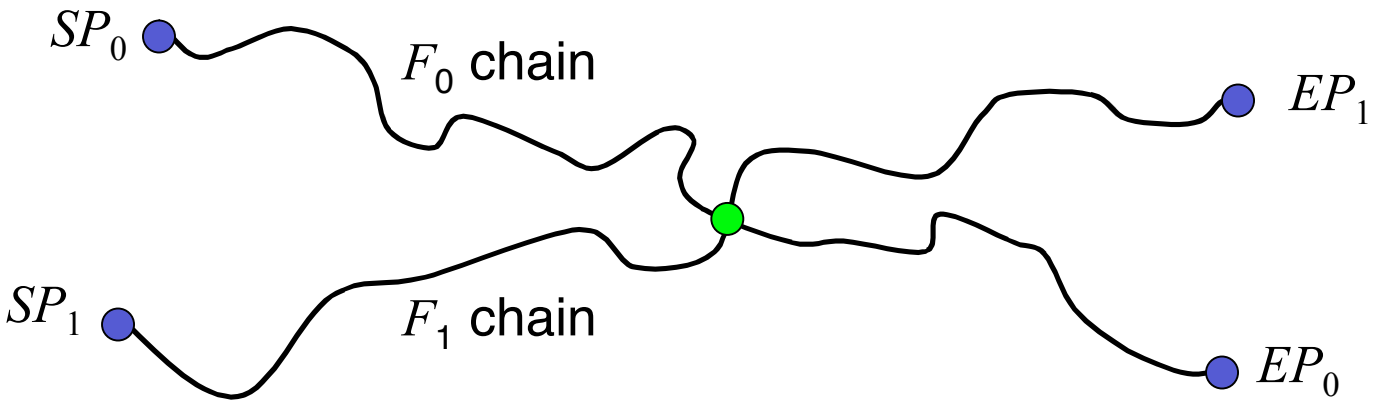
- Chain from C goes to EP
- Chain from SP to EP does not contain K
- Is this Trudy's nightmare?

Real-World TMTO Issues

- Merging, cycles, false alarms, etc.
- Pre-computation is lots of work
  - Must attack many times to make it worthwhile
- Success is not assured
  - Probability depends on initial work
- What if block size not equal key length?
  - This is easy to deal with
- What is the probability of success?
  - This is not so easy to compute

To Reduce Merging

- Compute chain as $F(E(P, K_{i-1}))$ where F permutes the bits
- Chains computed using different functions can intersect, but they will **not** merge



Hellman's TMTO in Practice

- Let
  - m = random starting points for each F
  - t = encryptions in each chain
  - r = number of "random" functions F
- Then mtr = total precomputed chain elements
- Pre-computation is $O(mtr)$ work
- Each TMTO attack requires
  - $O(mr)$ "memory" and $O(tr)$ "time"
- If we choose $m = t = r = 2^{k/3}$ then
  - Probability of success is at least 0.55

TMTO: The Bottom Line

- Attack is feasible against DES
- Pre-computation is about 2^{56} work
- Each attack requires about
  - 2^{37} "memory"
  - 2^{37} "time"
- Attack is not particular to DES
- No fancy math is required!
- Lesson: **Clever algorithms can break crypto!**

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
  - Tiger hash
  - HMAC
- Secret sharing
- Random numbers

Crypto Summary

- Information hiding
  - Steganography
  - Watermarking
- Cryptanalysis
  - Linear and differential cryptanalysis
  - RSA timing attack
  - Knapsack attack
  - Hellman's TMTO

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