COM3020J: Cryptography

Symmetric Crypto:

Dr. Anca Jurcut
E-mail: anca.jurcut@ucd.ie

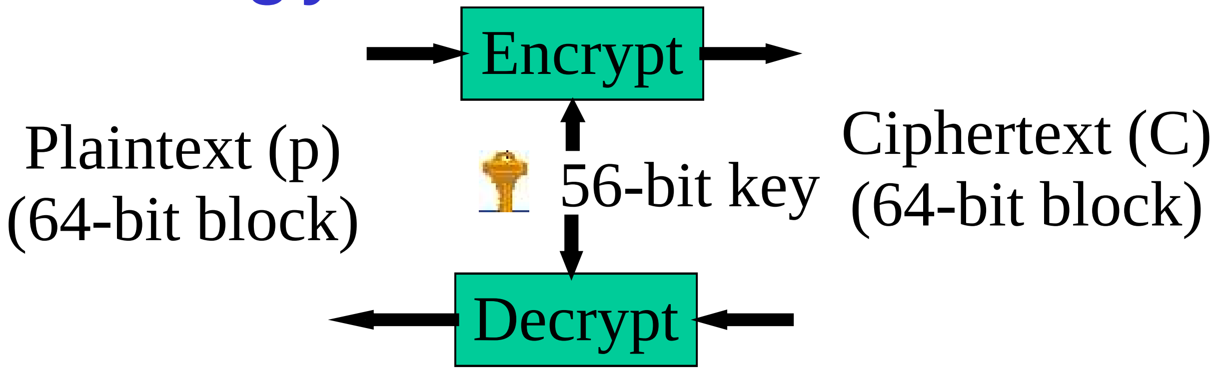
School of Computer Science and Informatics
University College Dublin

Beijing-Dublin International College

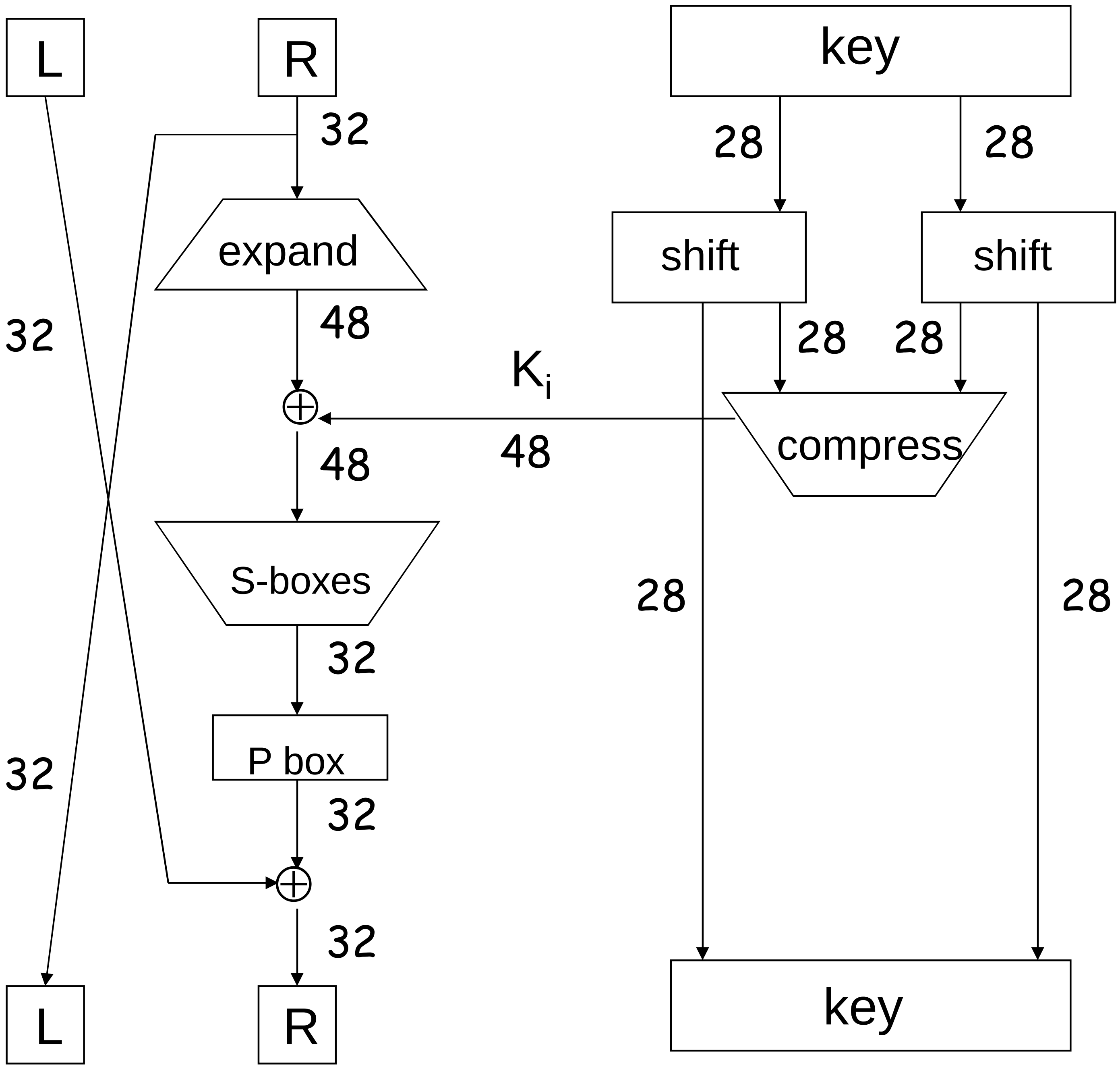
Data Encryption Standard

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

DES Numerology



- DES is a Feistel cipher with…
  - 64 bit block length: 64 bit input (plaintext), 64 bit output (ciphertext).
  - 56 bit key length
  - 16 rounds for Encryption & Decryption
  - 48 bits of key used each round (subkey)
- Round function is simple (for block cipher)
- Security depends heavily on "S-boxes"
  - Each S-box maps 6 bits to 4 bits



One Round of DES

DES Expansion Permutation

- Input 32 bits

  0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

  16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

- Output 48 bits

  31 0 1 2 3 4 3 4 5 6 7 8

  7 8 9 10 11 12 11 12 13 14 15 16

  15 16 17 18 19 20 19 20 21 22 23 24

  23 24 25 26 27 28 27 28 29 30 31 0

DES S-box

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

input bits (0,5)
↓
input bits (1,2,3,4)

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

DES P-box

- Input 32 bits

  0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

  16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

- Output 32 bits

  15 6 19 20 28 11 27 16 0 14 22 25 4 17 30 9

  1 7 23 13 31 26 2 8 18 12 29 5 21 10 3 24

DES Subkey

- 56 bit DES key, numbered 0,1,2,…,55
- Left half key bits, LK

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

- Right half key bits, RK

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

DES Subkey

- For rounds i=1,2,...,16
  - Let LK = (LK circular shift left by r_i)
  - Let RK = (RK circular shift left by r_i)
  - Left half of subkey K_i is of LK bits

    13 16 10 23 0 4 2 27 14 5 20 9
    22 18 11 3 25 7 15 6 26 19 12 1
  - Right half of subkey K_i is RK bits

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

DES Subkey

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

DES Last Word (Almost)

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

Security of DES

- Security depends heavily on S-boxes
  - Everything else in DES is linear
- 35+ years of intense analysis has revealed no back door
- Attacks, essentially exhaustive key search
- **Inescapable conclusions**
  - Designers of DES knew what they were doing
  - Designers of DES were way ahead of their time (wrt certain cryptanalytic techniques)

Block Cipher Notation

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

Symmetric Encryption Algorithms

	DES	Triple DES	AES
Plaintext block size (bits)	64	64	128
Ciphertext block size (bits)	64	64	128
Key size (bits)	56	112 or 168	128, 192, or 256

DES = Data Encryption Standard
AES = Advanced Encryption Standard

Triple DES

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

3DES

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

Advanced Encryption Standard

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

AES: Executive Summary

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

AES ByteSub

- Treat 128 bit block as 4x4 byte array

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

- 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

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix} \longrightarrow \texttt{ShiftRow} \longrightarrow \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{11} & a_{12} & a_{13} & a_{10} \\ a_{22} & a_{23} & a_{20} & a_{21} \\ a_{33} & a_{30} & a_{31} & a_{32} \end{bmatrix}$$

AES MixColumn

- Invertible, linear operation applied to each column

$$\begin{bmatrix} a_{0i} \\ a_{1i} \\ a_{2i} \\ a_{3i} \end{bmatrix} \longrightarrow \texttt{MixColumn} \longrightarrow \begin{bmatrix} b_{0i} \\ b_{1i} \\ b_{2i} \\ b_{3i} \end{bmatrix} \quad \text{for } i = 0, 1, 2, 3$$

- Implemented as a (big) lookup table

AES AddRoundKey

- XOR subkey with block

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

Block Subkey

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

AES Decryption

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

A Few Other Block Ciphers

- Briefly…
  - IDEA
  - Blowfish
  - RC6
- 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 for algorithm to depend on plaintext

Time for TEA…

- Tiny Encryption Algorithm (TEA)
- 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 of 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 $\oplus$ (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

Symmetric Algorithms - Summary

Symmetric Encryption Algorithm	Key length (in bits)	Description
DES	56	Designed at IBM during the 1970s and was the NIST standard until 1997. Although considered outdated, DES remains widely in use. Designed to be implemented only in hardware, and is therefore extremely slow in software.
3DES	112 and 168	Based on using DES three times which means that the input data is encrypted three times and therefore considered much stronger than DES. However, it is rather slow compared to some new block ciphers such as AES.
AES	128, 192, and 256	Fast in both software and hardware, is relatively easy to implement, and requires little memory. As a new encryption standard, it is currently being deployed on a large scale.
Software Encryption Algorithm (SEAL)	160	SEAL is an alternative algorithm to DES, 3DES, and AES. It uses a 160-bit encryption key and has a lower impact to the CPU when compared to other software-based algorithms.
The RC series	RC2 (40 and 64) RC4 (1 to 256) RC5 (0 to 2040) RC6 (128, 192, and 256)	A set of symmetric-key encryption algorithms invented by Ron Rivest. RC1 was never published and RC3 was broken before ever being used. RC4 is the world's most widely used stream cipher. RC6, a 128-bit block cipher based heavily on RC5, was an AES finalist developed in 1997.

Block Cipher Modes

Multiple Blocks

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

Modes of Operation

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

ECB Mode

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

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

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

ECB Cut and Paste

- Suspose 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

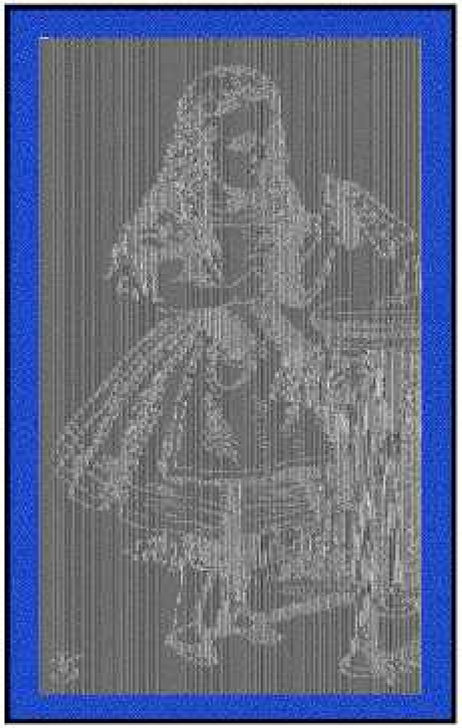
  Alice digs Tom. Trudy digs Bob.

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, and ECB encrypted (TEA)



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

- $C_i = E(C_{i-1} \oplus P_i, K)$
- $P_i = C_{i-1} \oplus D(C_i, K)$,

CBC Mode

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

Encryption

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

Decryption

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

- Analogous to classic codebook *with additive*

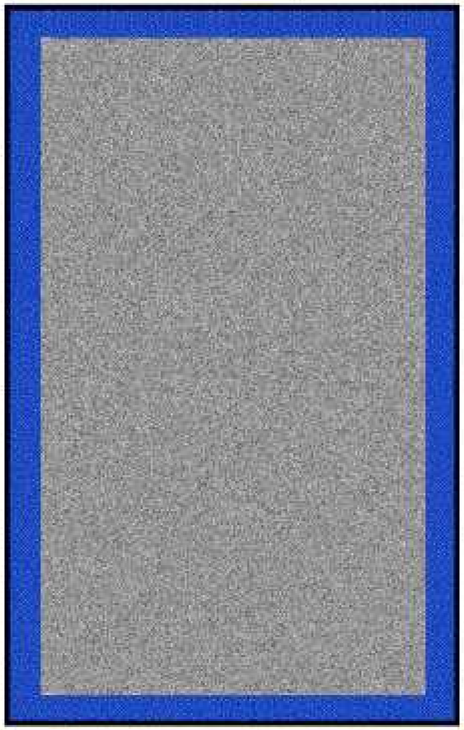
CBC Mode

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

    $P_1 \neq C_0 \oplus D(G, K)$, $P_2 \neq G \oplus D(C_2, K)$
  - But $P_3 = C_2 \oplus D(C_3, K)$, $P_4 = C_3 \oplus D(C_4, K)$, …
  - Automatically recovers from errors!
- Cut and paste is still possible, but more complex (and will cause garbles)

Alice Likes CBC Mode

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

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

Counter Mode (CTR)

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

Encryption

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

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

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

Decryption

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

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

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

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

Integrity using Symmetric Key Crypto

Data Integrity

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

MAC

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

MAC Computation

- MAC computation (assuming N blocks)

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

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

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

  $C_{N-1} = E(C_{N-2} \oplus P_{N-1}, K) = MAC$
- Send IV, P_0, P_1, …, P_{N-1} and MAC
- Receiver does same computation and verifies that result agrees with MAC
- Both sender and receiver must know K

Does a MAC work?

- Suppose Alice has 4 plaintext blocks
- Alice computes

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

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

  $\mathbf{C_0} = E(IV \oplus P_0, K)$, $\boldsymbol{C_1} = E(\mathbf{C_0} \oplus X, K)$,

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

Confidentiality and Integrity

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

Uses for Symmetric Crypto

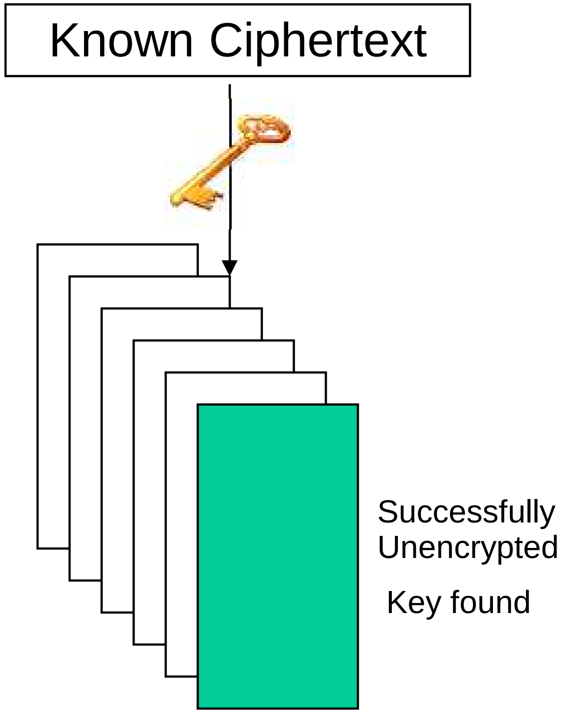
- Confidentiality
  - Transmitting data over insecure channel
  - Secure storage on insecure media
- Integrity (MAC)
- Authentication protocols (later…)
- Anything you can do with a hash function (upcoming lectures…)

Attacking Symmetric Encryption

- cryptanalysis
  - rely on nature of the algorithm
  - plus some knowledge of plaintext characteristics
  - even some sample plaintext-ciphertext pairs
  - exploits characteristics of algorithm to deduce specific plaintext or key

  attacks:
    - *ciphertext only - least info, hardest*
    - *known plaintext - some plain/cipher pairs*
    - *chosen plaintext - get own plain/cipher pairs*
    - *chosen ciphertext - rarer*
    - *chosen text - rarer*
- only weak algs fail a ciphertext-only attack
- usually design algs to withstand a known-plaintext attack
- brute-force attack
  - try all possible keys on some ciphertext until get an intelligible translation into plaintext

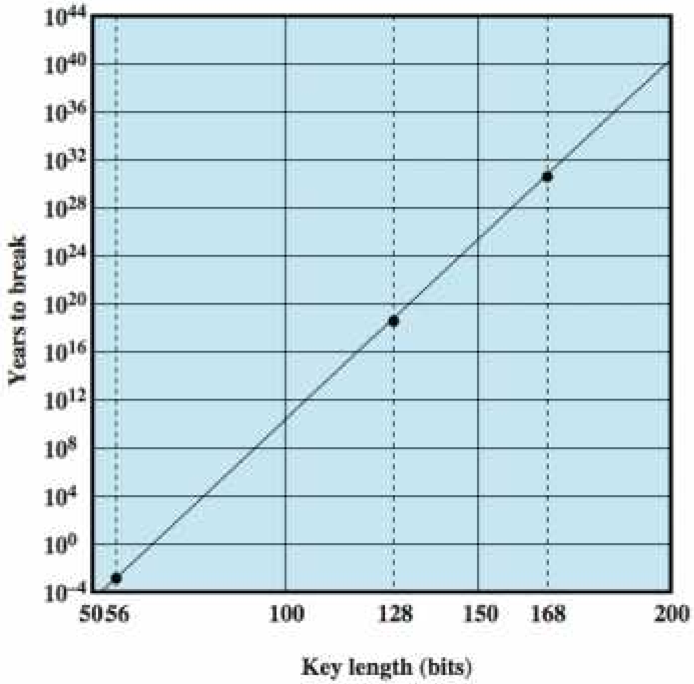
Brute Force Attack



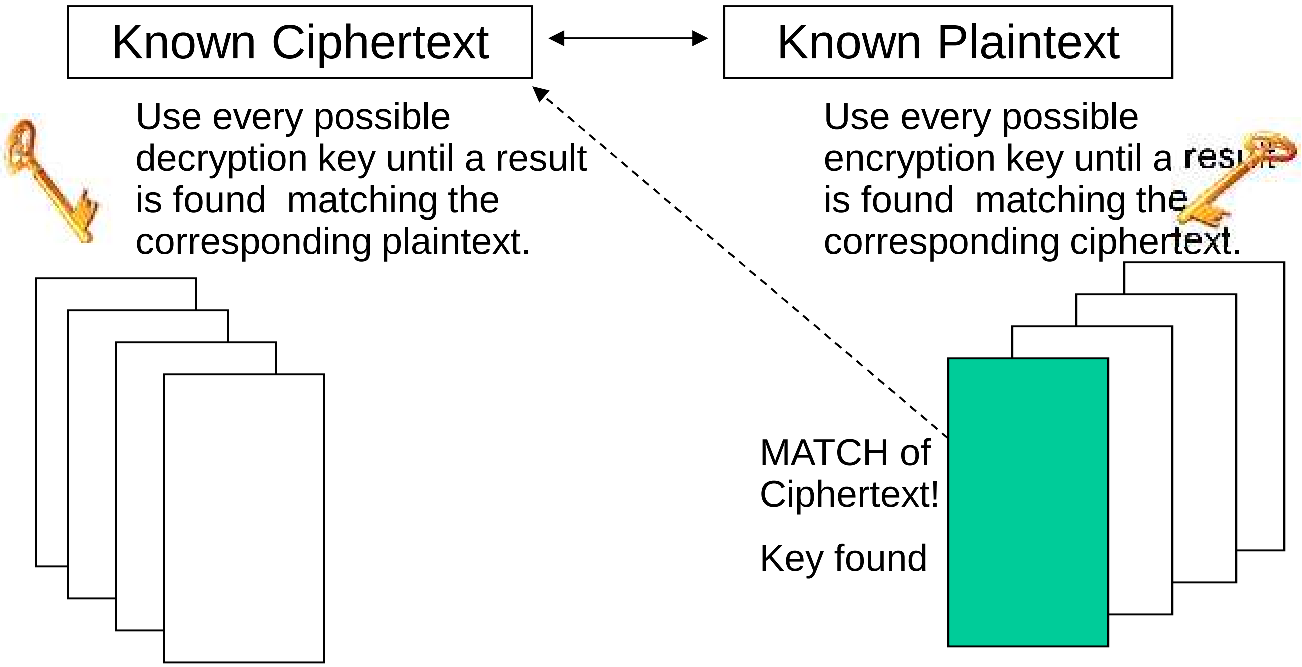
With a Brute Force attack, the attacker has some portion of ciphertext. The attacker attempts to unencrypt the ciphertext with all possible keys.

Exhaustive Key Search

Key Size (bits)	Number of Alternative Keys	Time Required at 1 Decryption/μs	Time Required at 10^6 Decryptions/μs
32	$2^{32} = 4.3 \times 10^9$	$2^{31}\ \mu s = 35.8$ minutes	2.15 milliseconds
56	$2^{56} = 7.2 \times 10^{16}$	$2^{55}\ \mu s = 1142$ years	10.01 hours
128	$2^{128} = 3.4 \times 10^{38}$	$2^{127}\ \mu s = 5.4 \times 10^{24}$ years	5.4×10^{18} years
168	$2^{168} = 3.7 \times 10^{50}$	$2^{167}\ \mu s = 5.9 \times 10^{36}$ years	5.9×10^{30} years
26 characters (permutation)	$26! = 4 \times 10^{26}$	$2 \times 10^{26}\ \mu s = 6.4 \times 10^{12}$ years	6.4×10^6 years



Meet-in-the-Middle Attack



With a Meet-in-the-Middle attack, the attacker has some portion of text in both plaintext and ciphertext. The attacker attempts to unencrypt the ciphertext with all possible keys while at the same time encrypt the plaintext with another set of possible keys until one match is found.