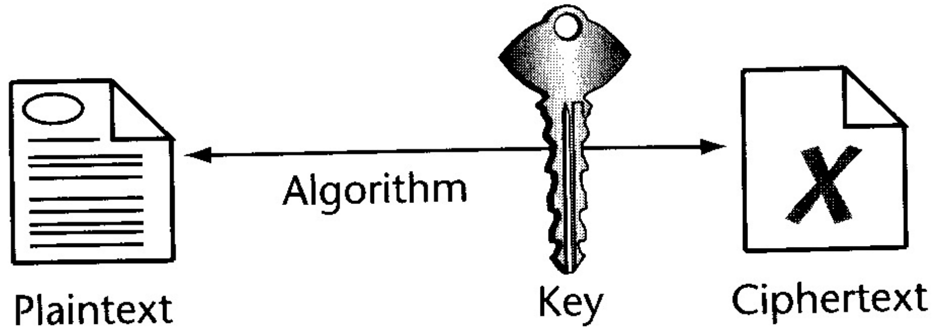
Cryptography

Four elements of a Crypto-system



- Only need to keep the "Key" secret, can afford to have the "algorithm" public
  - Again, can facilitate implementation by the mass
- It is easy to change the "Key", but difficult to design and describe /communicate a new secure algorithm
- Kerckhoff's Principle: The security of a cipher MUST NOT depend on anything that cannot be easily changed

More Terminology

- Encryption: converting plaintext to ciphertext
- Decryption: converting ciphertext to plaintext
- Cryptanalysis: to break the code by analyzing the the algorithm/system
- Brute-force attack: Enumerate over all the possible keys
- Types of attacks on Encrypted Messages by Cryptanalyst:
    - Ciphertext only: Given Ciphertext only to derive Plaintext/key
    - Known Plaintext: Given <Plaintext,Ciphertext> pair(s) to derive the key, e.g. “From:” at the beginning of each email
    - Chosen Plaintext: The attacker has the ability to inject chosen plaintext and observe the ciphertext outcome, e.g. send an email of chosen words to the victim while sniffing at the cipher

Increasing knowledge/level of control
by the Cryptanalyst

The three laws of security:

- Absolutely secure systems do not exist
- To halve your vulnerability, you have to double your expenditure
- Cryptography is typically bypassed, not penetrated

Fundamental Tenet of Cryptography:

- If lots of smart people have failed to solve a problem (break the code), then it probably won't be solved (broken) (soon).

Cryptographic Misconceptions (from the S. of RSA)

- By Policy Makers: crypto is dangerous, but:
  - weak crypto is not a solution
  - controls can’t stop the inevitable
- By Researchers: A provably secure system is secure but:
  - proven false by indirect attack
  - can be based on false assumption
  - requires careful choice of parameters
- By Implementers: Cryptography solves everything, but:
  - only basic ideas are successfully deployed
  - only simple attacks are avoided
  - bad crypto can provide a false sense of security

To Publish or Not to Publish (your cryptography algorithm)

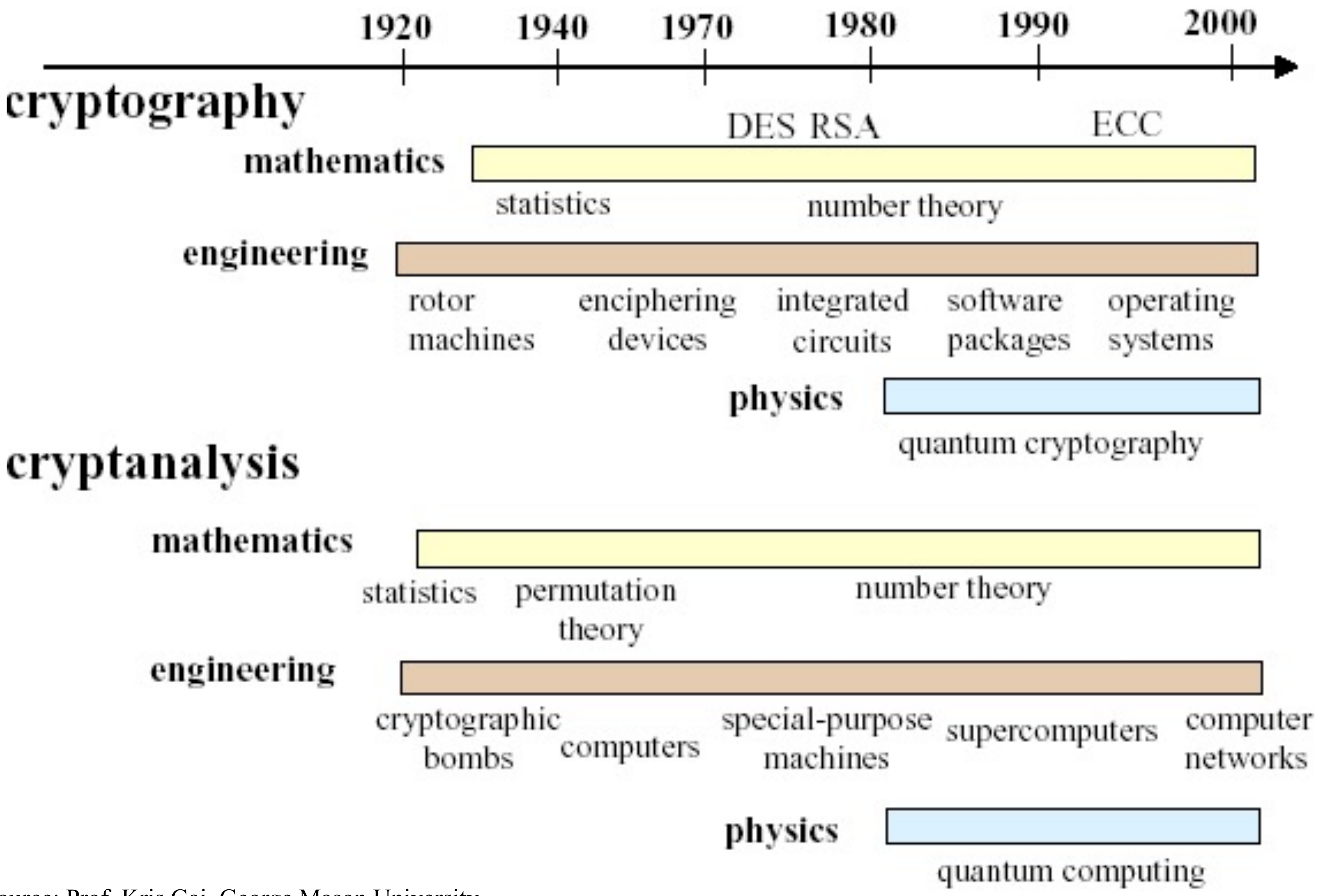
e.g. PKZip, those used by MS Office, Word-Perfect etc, RCx

- Security via Obscurity
- Disclose of design rationale may let others (your enemy) learn better cryptography design/ cryptanalysis techniques (Alternative is to publish the specification without explaining the rationale, e.g. DES design was published but rationale was never disclosed
- Smaller motivation to break ?
- Cryptanalysis must include recovering/reverse-engineering of the algorithm => the cryptosystem should be built to prevent reverse-engineering
- Treated as Trade Secrets for commercial benefits (Vs. Patents), e.g. RCx

e.g. DES, AES,

- Get free consulting/ scrutiny/ research from the large community of academic researchers
- Better security confidence if the scheme has been under more vigorous scrutiny. (However, "publicly available" does not guarantee public scrutiny)
- Difficult to really keep the algorithm secret for long anyway, GSM, RC4, GemStar,Electronic Voting Systems in US.
- Allow international standardization
- Facilitate large-scale lower cost production of the corresponding hardware/software
- No need to include anti-reverse-engineering protection

Evolution of cryptography and cryptanalysis



Source: Prof. Kris Gaj, George Mason University

Design Principles of Good Ciphers

- **Confusion: to make the relation between the key and the ciphertext as complex as possible.**
- **Diffusion: spreads the influence of a single plaintext bit over many ciphertext bits.**
- **Avalanche Effect: a minor change to the plaintext or the key must result in significant and random-looking changes to the ciphertext.**

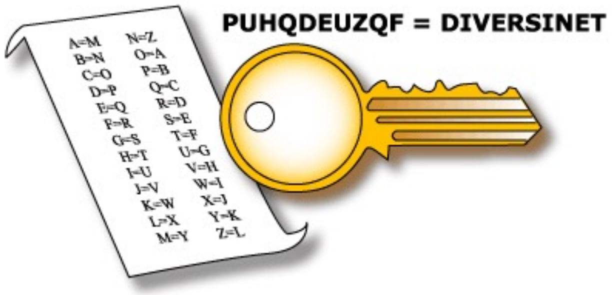
Basic Techniques in Conventional Ciphers

- **Permutations (aka Transposition): rearrange bits, or bytes or characters in the plaintext**
- **Substitution: replace bits, or bytes, or characters, or block of characters with substitute value**

Some Classical Ciphers

Caesar Cipher

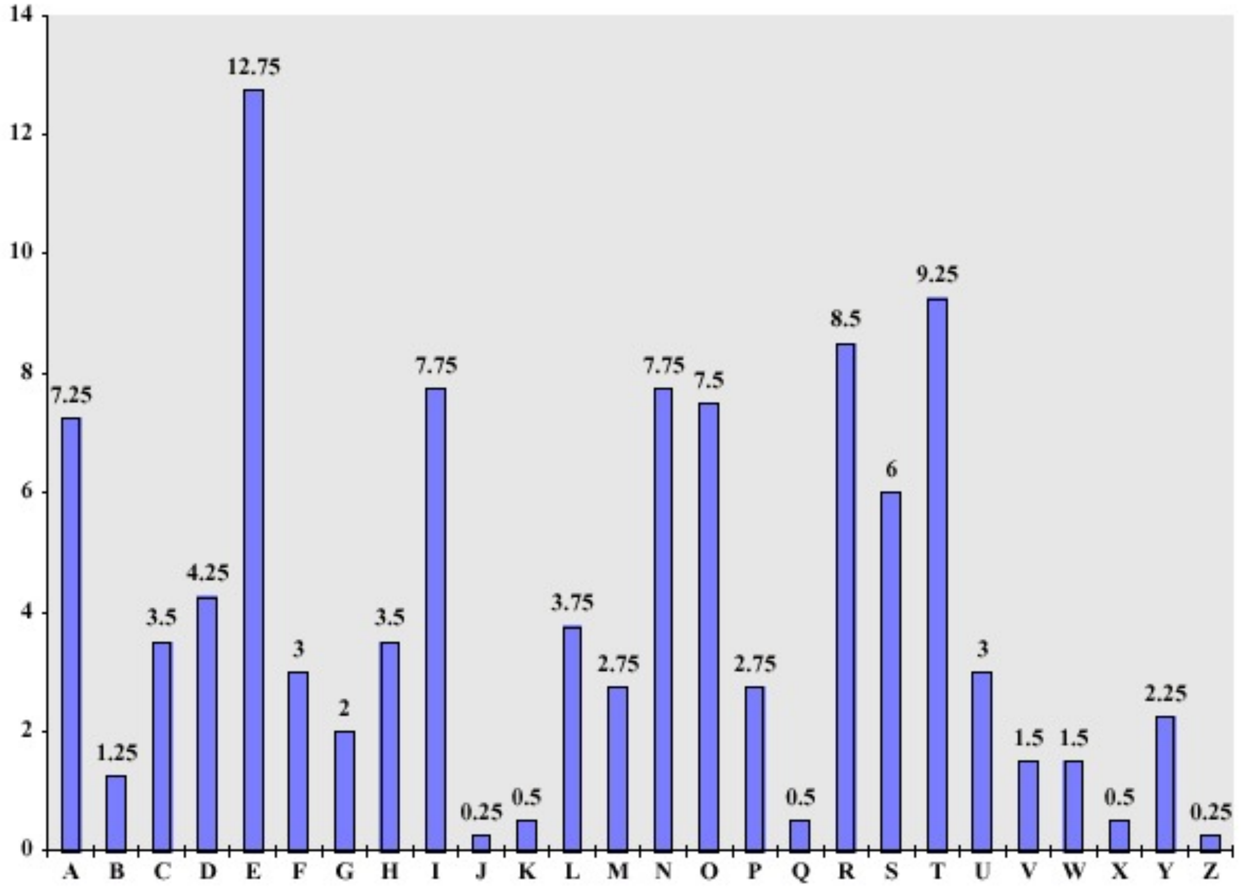
```
plain:   abcdefghijklmnopqrstuvwxyz
key:     defghijklmnopqrstuvwxyzabc
```



```
plain:  SEE ME AFTER THIS CLASS
cipher: VHH PH DIWHU WKLV FODVV
```

Caesar Cipher can be easily broken via Frequency Analysis of alphabets

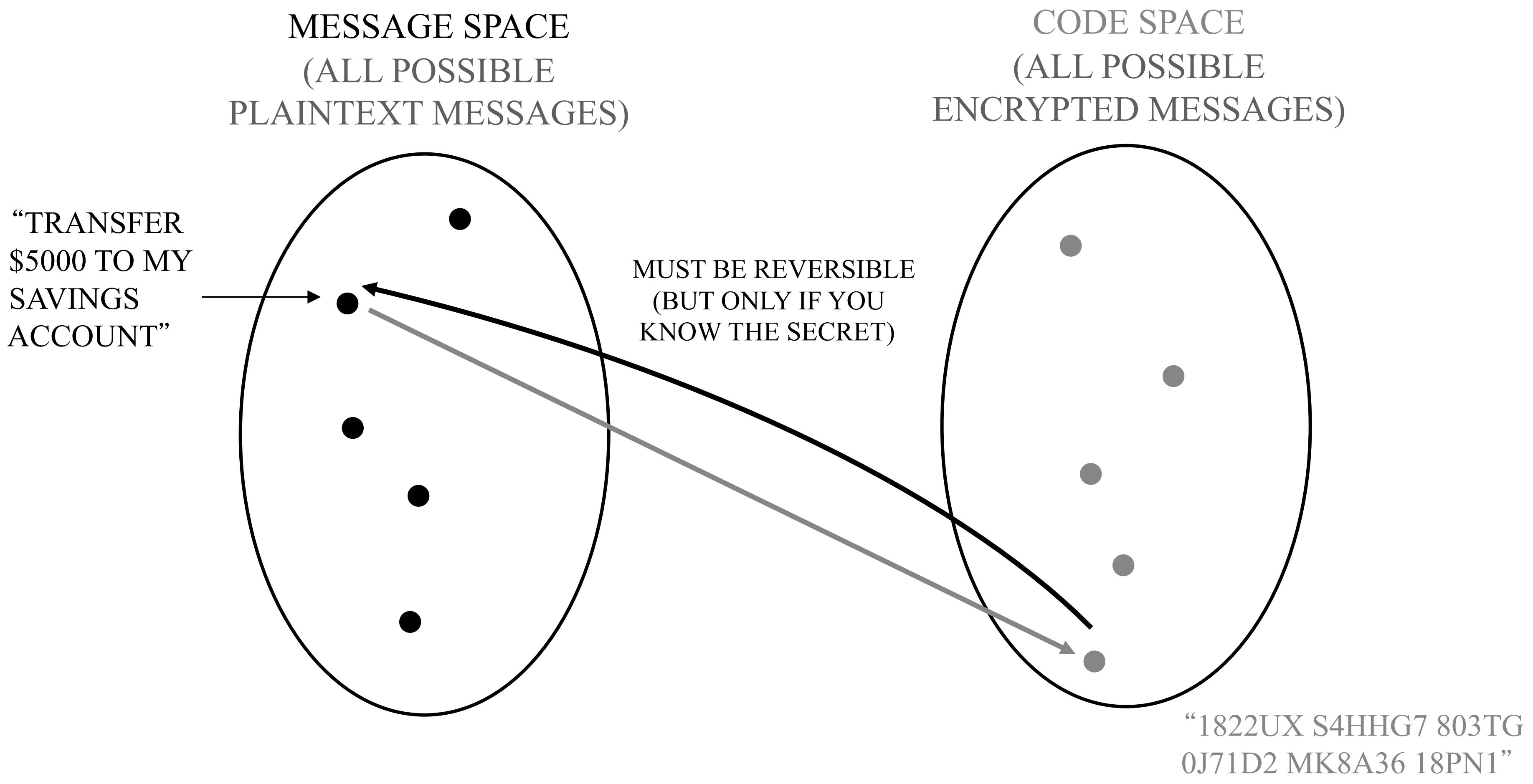
Letter Frequency in English Language



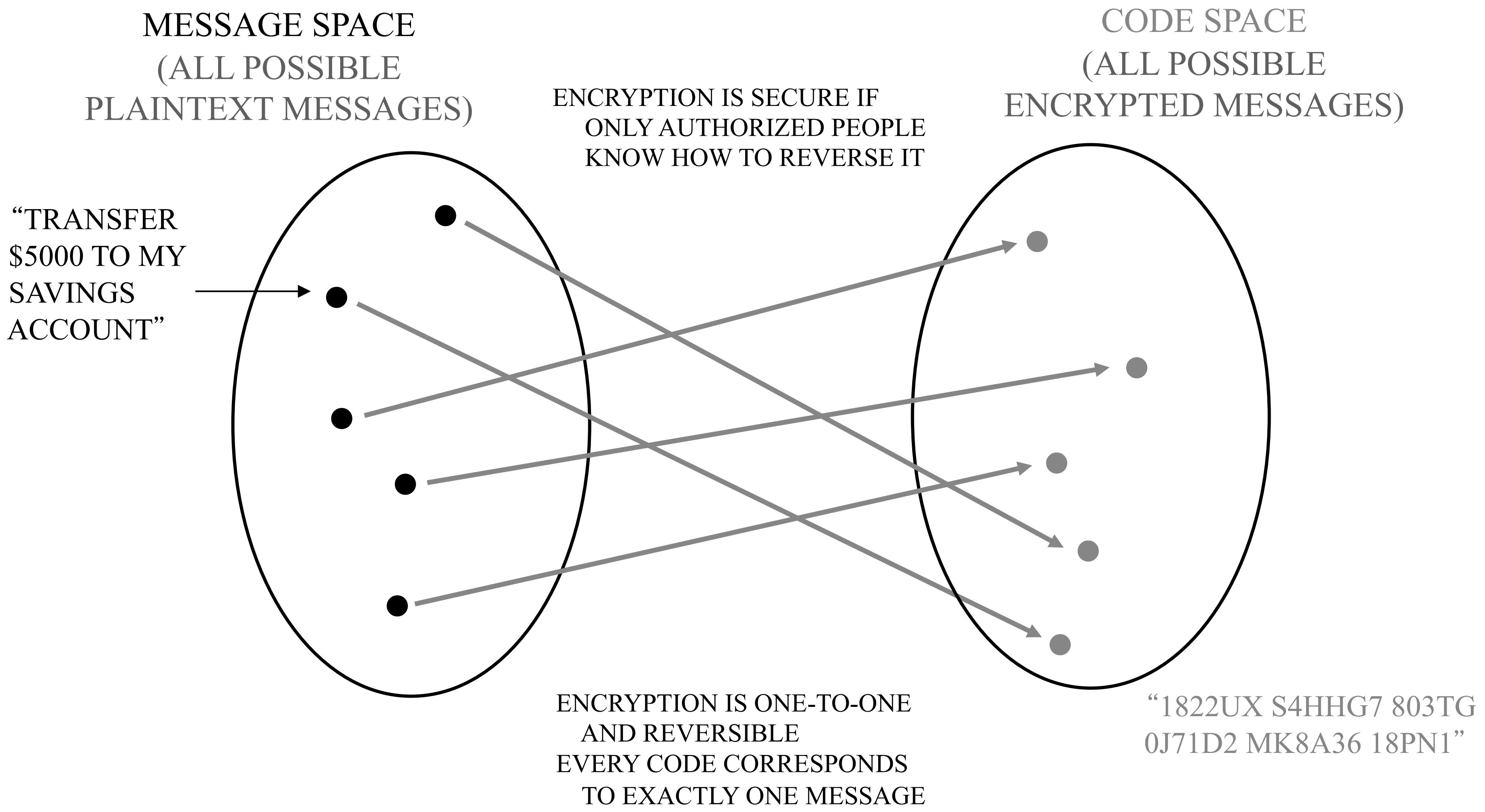
“Rail-Fence” Cipher

```
DISGRUNTLED EMPLOYEE
         ↓
D   R   L   E   O
 I G U T E   M L Y E
  S   N   D   P   E
         ↓
DRLEOIGUTE MLYESNDPE
```

Cryptography



Cryptography



One-time Pad

```
XMCKLA EJKAWO FECIFE WSNZIP PXPKIY URMZHI JZTLBC YLGDYJ HTSVTV
RRYYEG EXNCGA GGQVRF FHZCIB EWLGGR BZXQDQ DGGIAK YHJYEQ TDLCQT
HZBSIZ IRZDYS RBYJFZ AIRCWI UCVXTW YKPQMK CKHVEX VXYVCS WOGAAZ
OUVVON GCNEVR LMBLYB SBDCDC PCGVJX QXAUIP PXZQIJ JIUWYH COVWMJ
UZOJHL DWHPER UBSRUJ HGAAPR CRWVHI FRNTQW AJVWRT ACAKRD OZKIIB
VIQGBK IJCWHF GTTSSE EXFIPJ KICASQ IOUQTP ZSGXGH YTYCTI BAZSTN
```

Encryption:

```
      H          E          L          L          O   plain
    7 (H)      4 (E)     11 (L)     11 (L)     14 (O) plain
+  23 (X)     12 (M)      2 (C)     10 (K)     11 (L) key
=  30         16         13         21         25     plain + key
=   4 (E)     16 (Q)     13 (N)     21 (V)     25 (Z)(plain + key) mod 26
      E          Q          N          V          Z   → cipher
```

Decryption: **`Plain = (cipher – key) mod 26`**

- **Achieve "Perfect Secrecy": i.e. the Ciphertext alone does NOT tell you Any Information about the Plaintext**
  - **Can't be cracked even with Infinite Computational Power:**

    `EQVNZ -> LATER` **with another key =** `TQURI`

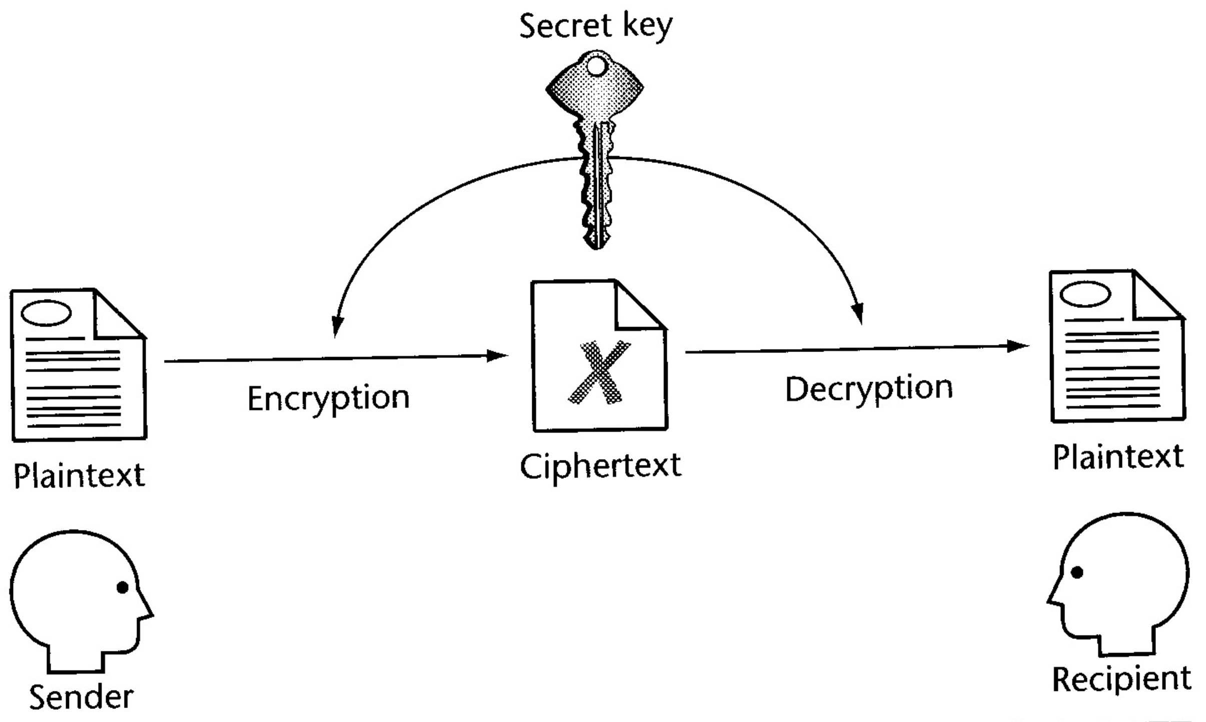
Classification of Cryptosystems and Terminology

- Secret-Key
- Conventional
- Classical
- Symmetric
- Symmetric-key

- Public-Key
- Asymmetric

Symmetric (aka Secret-Key) Cryptosystems

A Secret-key Cryptosystem



- The Same key is used for encryption as well as decryption ; That's why it is also also "symmetric key" system
- The encryption/decryption algorithm is sometimes referred as the "cipher"

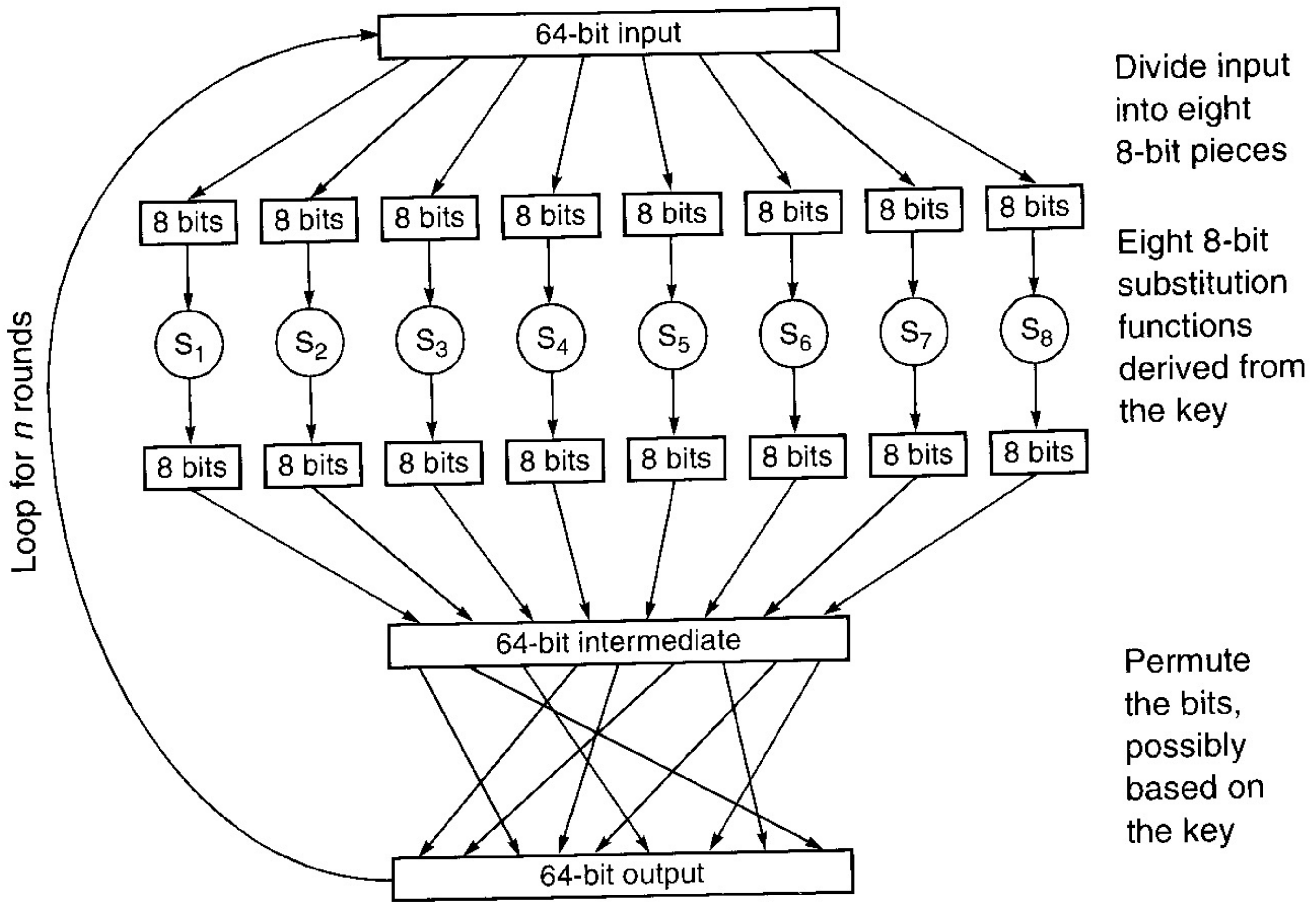
Block Cipher Vs. Stream Cipher

- Process the message block by block of constant size, e.g. 64 bits
- Each block goes through multiple rounds of permutation and substitution
- Mixed operators, data or key dependent rotation/shifting =>permutation
- Key dependent substitution (S-boxes) =>substitution
- More complex key scheduling: part of the key is used to generate the "per-round key" for each round

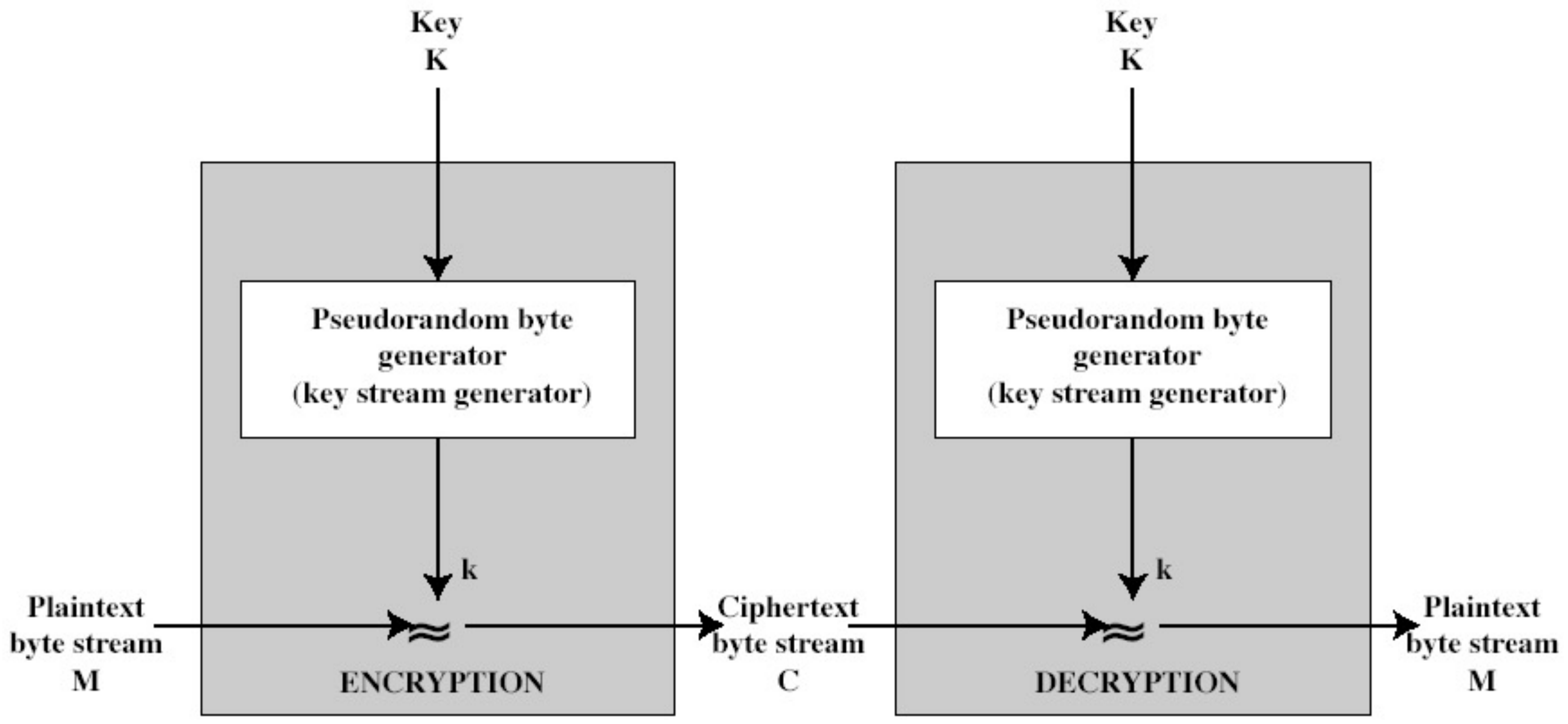
- Process the message bit by bit (as a stream) or byte-by-byte
- Typically have a (pseudo) random **stream key**
- combined ("⊕" XOR) with plaintext bit by bit
- randomness of **stream key** completely destroys any statistically properties in the message
  - $C_i = M_i \oplus S_i$

  ```
  Ci = i-th bit of ciphertext
  Mi = i-th bit of plaintext
  Si = i-th bit of stream key
  ```
- what could be simpler, faster !!!!
- **but must never reuse stream key**
  - **otherwise can remove effect and recover messages**

General Example of a Block Cipher


64-bit input
8 bits
S_1
S_2
S_3
S_4
S_5
S_6
S_7
S_8
64-bit intermediate
64-bit output
Loop for *n* rounds
Divide input into eight 8-bit pieces
Eight 8-bit substitution functions derived from the key
Permute the bits, possibly based on the key

Stream Cipher


Key
K
Pseudorandom byte generator (key stream generator)
k
Plaintext byte stream M
ENCRYPTION
Ciphertext byte stream C
Key
K
Pseudorandom byte generator (key stream generator)
k
DECRYPTION
Plaintext byte stream M

The ⊕"XOR" function is its own Inverse

A	B	A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

A	A	A⊕A
0	0	0
1	1	0

A	0	A⊕0
0	0	0
1	0	1

Given: $Ci = Mi \oplus Si$;

$Ci \oplus Si = (Mi \oplus Si) \oplus Si = Mi \oplus (Si \oplus Si) = Mi \oplus \mathbf{0} = Mi$

because

$A \oplus \mathbf{A} = \mathbf{0}$ for all A and

$A \oplus \mathbf{0} = \mathbf{A}$ for all $\mathbf{A}$

Some design Considerations for Stream Cipher

- The "Secret Key" is used to generate the "seed" of a pseudo random number generator (PRNG) which output the random stream key
- The seed should be "large enough" to avoid exhaustive enumeration
- The PRNG should have
  - long period with no repetitions
  - statistically random
  - correlation immunity

"The generation of random numbers is too important to be left to chance" -- --- Robert Coveyou

"Random numbers should not be generated with a method chosen at random" ---- Donald Knuth

RC4 (Ron’s Code #4)

- a proprietary cipher (trade-secret) owned by RSA DSI
- Leaked to the public in 1994 via an Internet posting
- Another Ron Rivest design, simple but effective
- Variable key size (1 to 256-byte long), byte-oriented stream cipher
- Widely used in practice (Web SSL/TLS, Wireless LAN WEP)
- Key forms random permutation of all 8-bit values
- Uses that permutation to scramble input info processed a byte at a time

RC4 Key Schedule

- starts with an array S of numbers: 0..255
- use key to well and truly shuffle
- S forms *internal state* of the cipher
- given a key k of length L bytes

```
for i = 0 to 255 do
    S[i] = i
j = 0
for i = 0 to 255 do
    j = (j + S[i] + k[i mod L]) (mod 256)
    swap (S[i], S[j])
```

RC4 Encryption

- encryption continues shuffling array values
- sum of shuffled pair selects "stream key" value
- XOR with next byte of message to en/decrypt

```
i = j = 0
for each message byte $M_i$
    i = (i + 1) (mod 256)
    j = (j + S[i]) (mod 256)
    swap(S[i], S[j])
    t = (S[i] + S[j]) (mod 256)
    $C_i$ = $M_i$ XOR S[t]
```

RC4 Security

- claimed secure against known attacks
  - have some analytical attacks, none practical
- result is very non-linear
- since RC4 is a stream cipher, must **never reuse a key**
- MUST Throw away the 1st few hundred bytes of the key-stream to avoid the "weak-key" problem of RC4

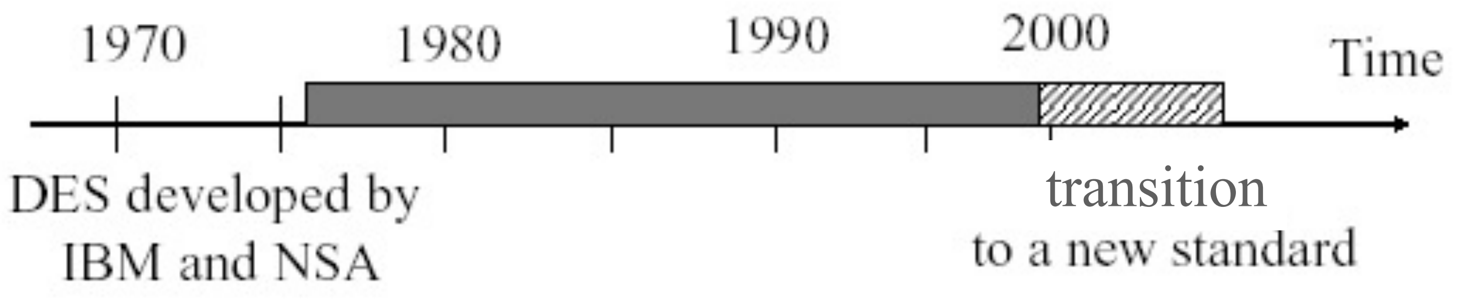
- Have a concern with WEP in 802.11 Wireless LANs, but mostly due to key handling rather than RC4 itself

- Latest: Based on research presented in Usenix Security 2015 and more recent discoveries (http://www.rc4nomore.com), one can decrypt web-cookies that are protected by TLS using RC4 within 52-75 hours. As a result, by Oct 2017, major browsers have disabled RC4: < 1% of all HTTPS and TLS connections still use RC4.

DES: The Data Encryption Standard

- Designed in the 1970's by IBM with inputs from the NSA (National Security Agency) of US
- The most widely used encryption standard in the world
- Encrypt plaintext block-by-block with 64 bits per block ; this is so-called a "block cipher".
- Use 56-bit key (7-bit * 8) ; for each 7-bit key-segment, a parity bit is added as checksum to form an octet (which contains redundant info). This results in a 64-bit word to be fed into the algorithm
- The same hardware can be used for both encryption and decryption based on the elegant "Feistel" network structure
- Designed to facilitate hardware implementation WHILE making software implementation much slower
- In 1994, DES was reaffirmed by NIST for US Federal Government use for another 5 years, i.e., due 1999.
- In 1999, DES NIST issued a new standard requiring "Triple DES" to be used instead.

Chronology of DES



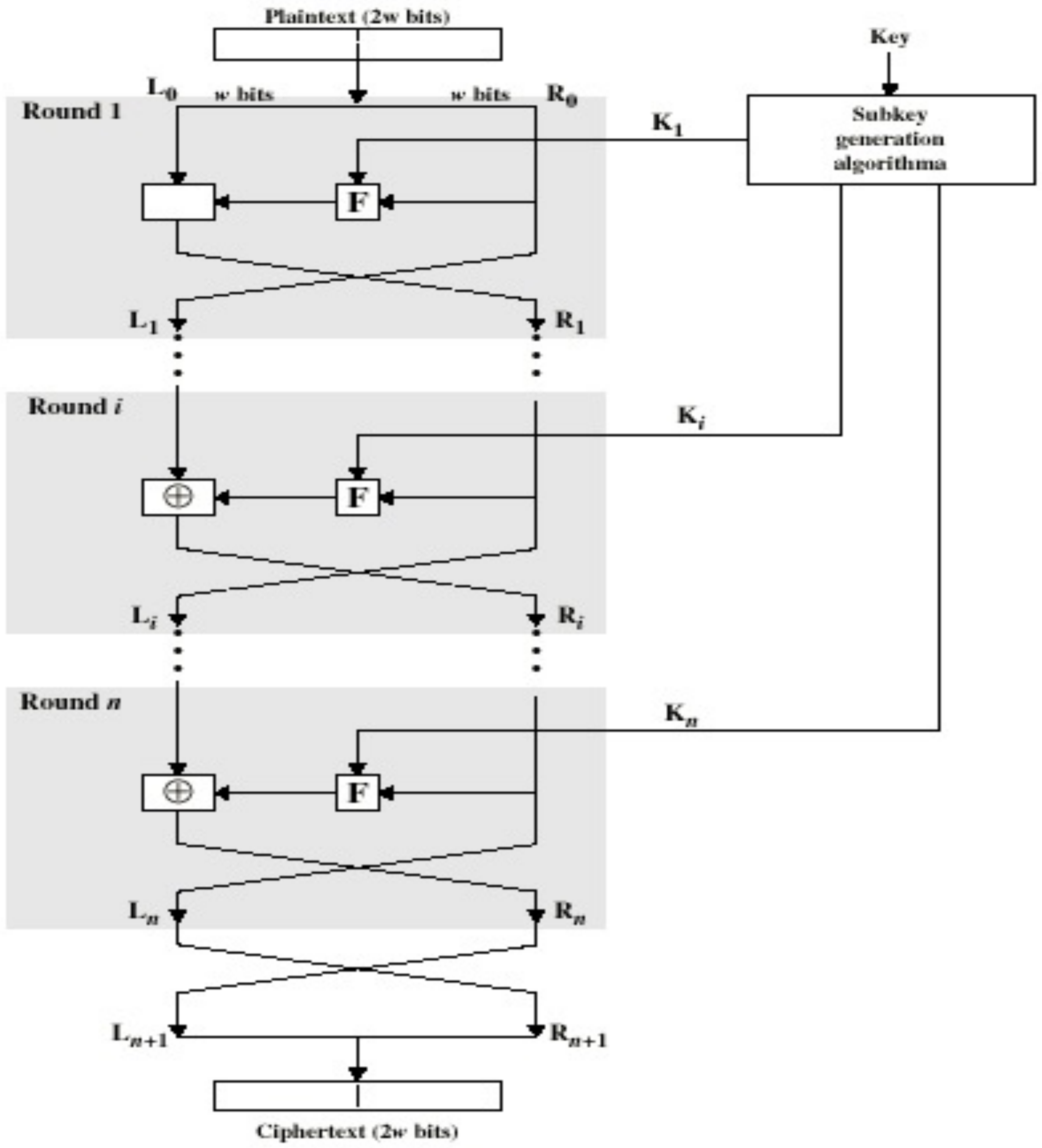
In common use for over 20 years

Federal and banking standard

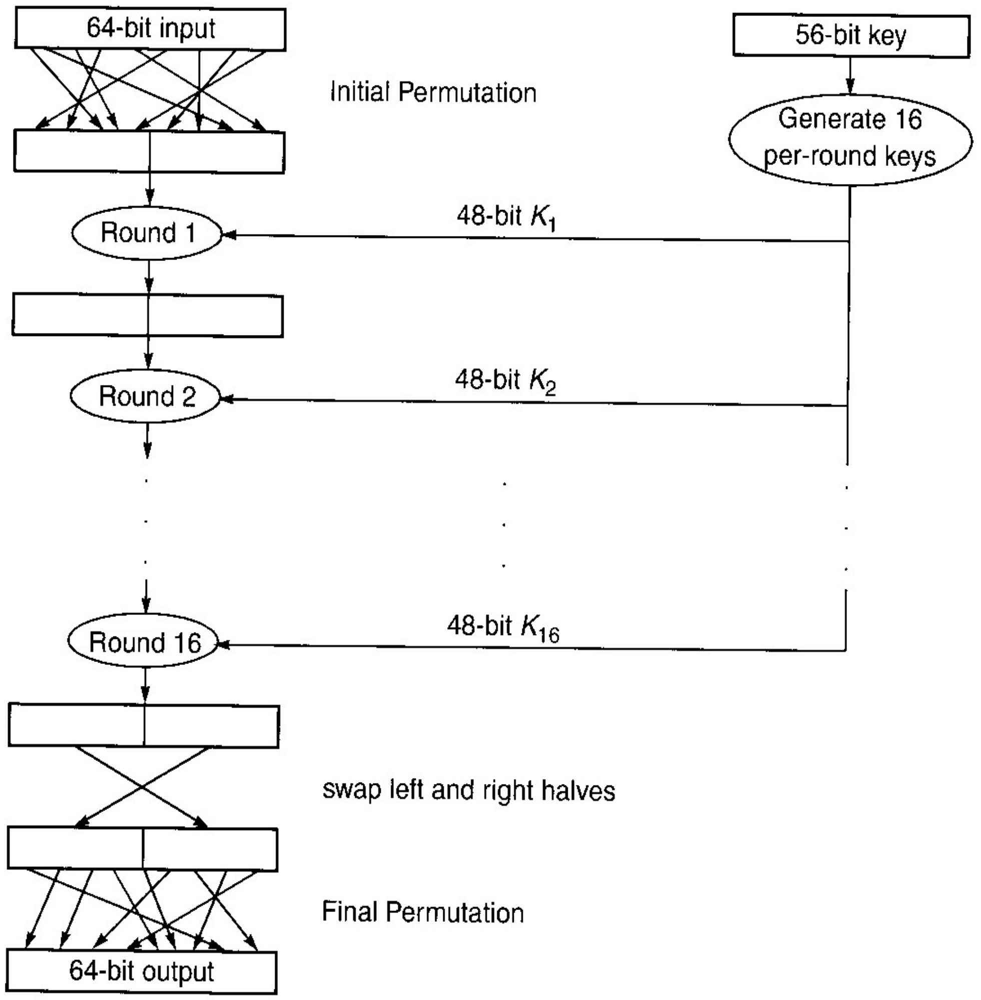
Over 130 validated implementations

De facto world-wide standard

The Feistel Network

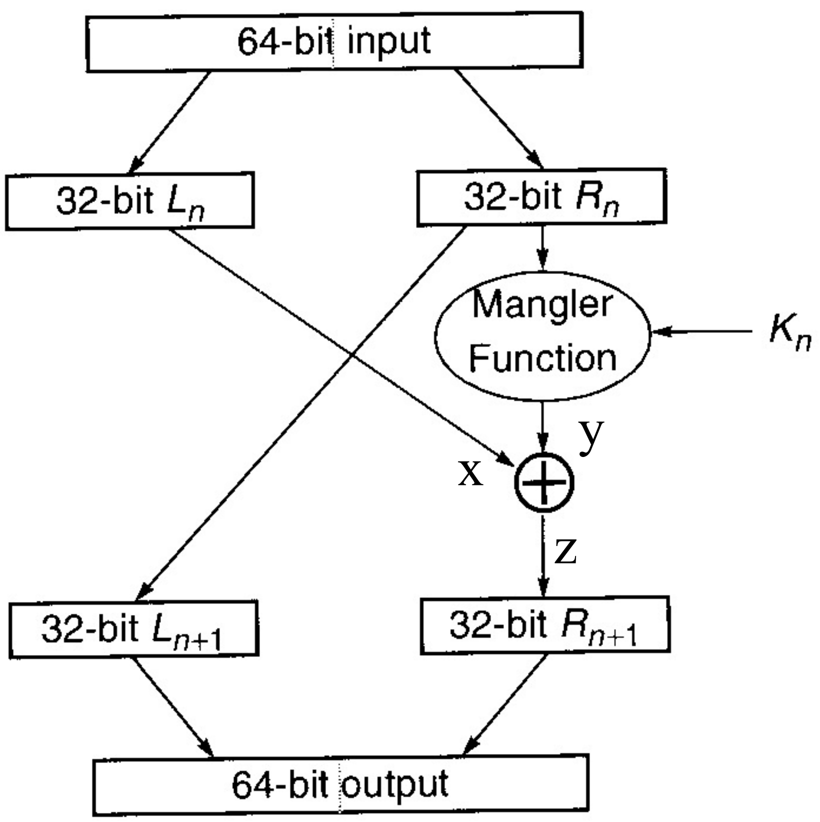


Overall Structure of the DES algorithm

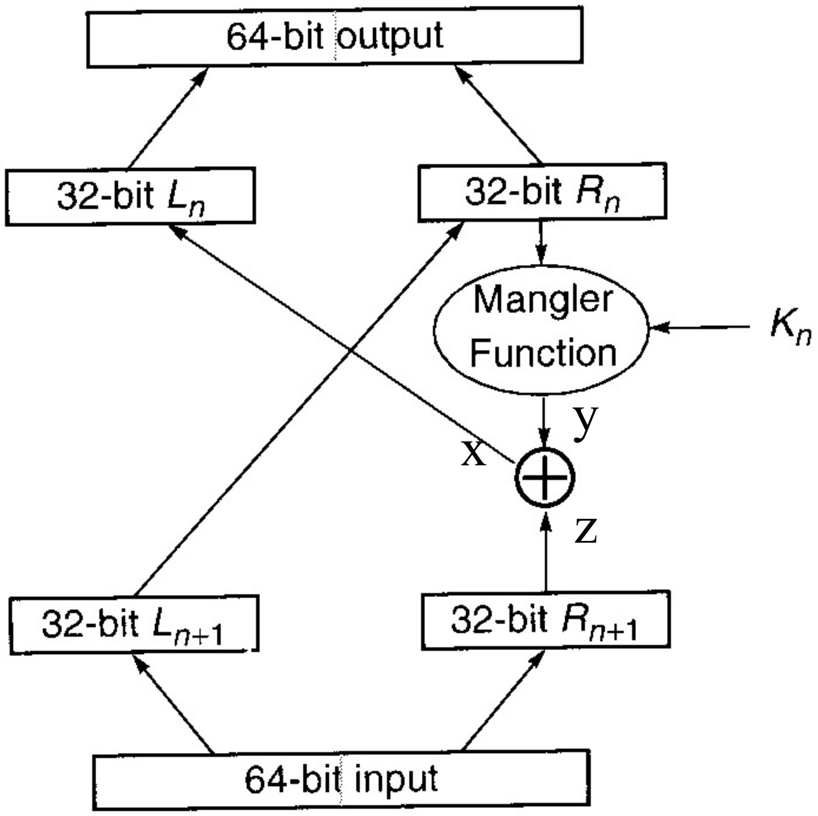


Why 16 rounds ?

A DES Round



Encryption



Decryption

The Mangler Function takes a 32-bit input Rn, and a 48-bit per-round key Kn to yield an 32-bit output based on S-Box look-up followed by permutation

The ⊕“XOR” function is its own Inverse

A	B	A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

A	A	A⊕A
0	0	0
1	1	0

A	0	A⊕0
0	0	0
1	0	1

Given: $z = x \oplus y$;

$z \oplus y = (x \oplus y) \oplus y = x \oplus (y \oplus y) = x \oplus \mathbf{0} = x$

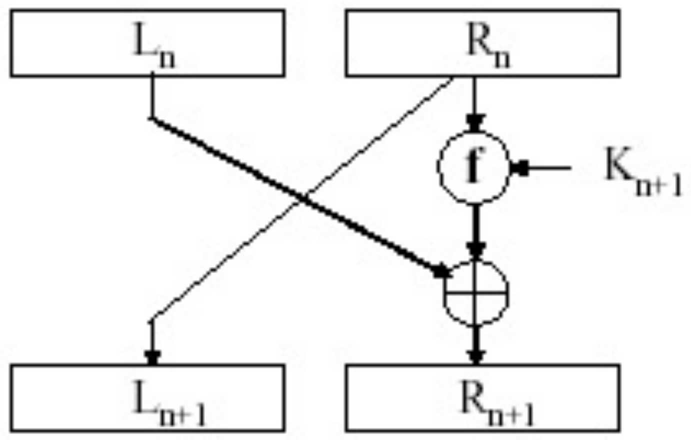
because

$A \oplus \mathbf{A} = \mathbf{0}$ for all A and

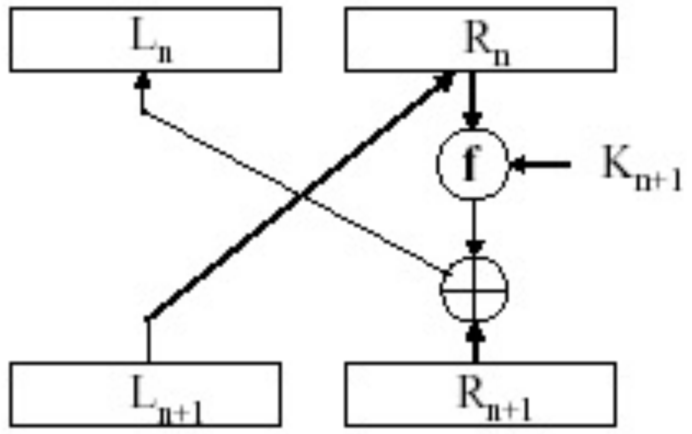
$A \oplus 0 = \mathbf{A}$ for all $\mathbf{A}$

The “Magic” of the Feistel Structure

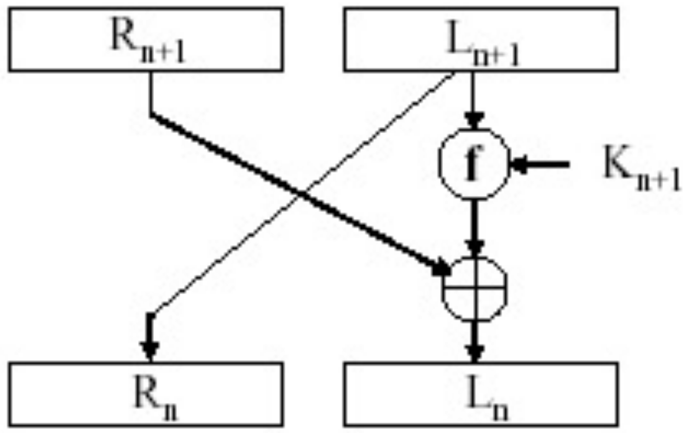
Encryption



Decryption



(1) The same hardware can be used for both encryption and decryption by simply swapping the left and right halves of the input ; (no need to run The algorithm backward)



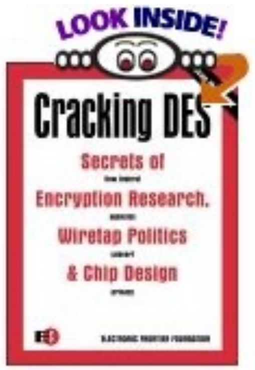
(2) No need to construct the inverse of function f = the Mangler function in DES

Controversy behind DES

- 56-bit key is too small to start with (it seems like the original design was for 64-bit key and was forced to shorten the key to 56-bit, so that … NSA can break it ??)
- Because of the involvement of NSA and because it is commonly believed that NSA will not recommend an algorithm which it cannot break, people were worried about the existent of "backdoor" within the algorithm, e.g. the S-boxes were all chosen by NSA and has shown to be not random and have special structure
- To make things worse, the rationale behind the design was never disclosed by the designers
  - The designers knew a lot of advanced ways to attack encryption algorithms (cryptanalysis) and DES was designed to protect against all these known ways of attack ; explaining the design rationale would teach "the enemy" about those advanced cryptanalysis techniques.
- Software Implementation of DES has poor performance
  - The initial and final permutations do not add secure value, merely to slow down software implementation of the algorithm
  - tried to limit the spread of the technology

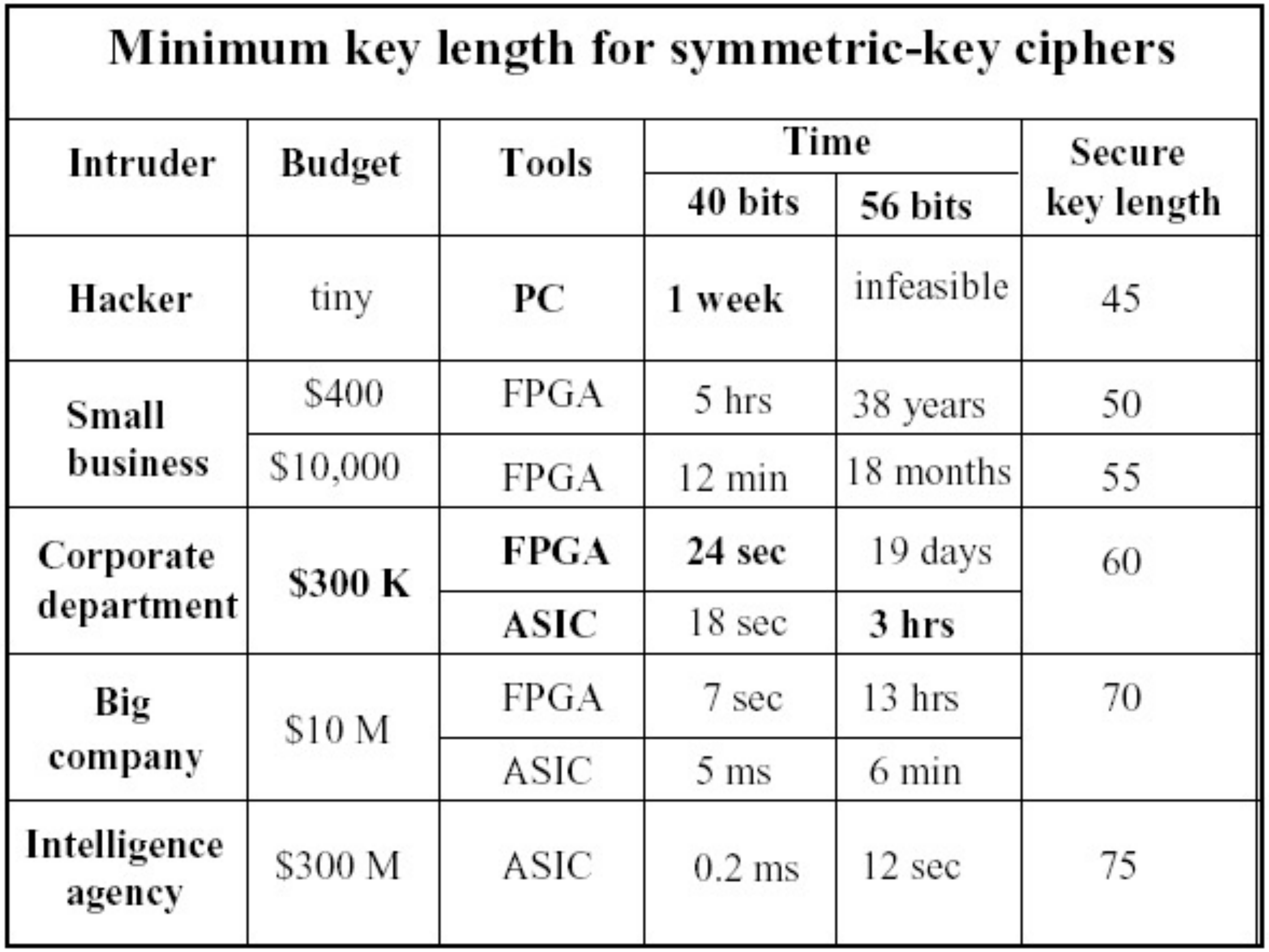
How Secure is DES ?

- In practice, most commercial DES hardware solution is deliberately designed to have long key-loading time: it can encrypt/decrypt a large amount of data with a given key at top-speed but not suitable for key-search attack
- To show the inadequacy of a 56-bit key length, the Electronic Frontier Foundation (EFF) had designed "Deep Crack", an ASIC-based specialized machine (1800 ASIC chips, 40MHz clock) in 1998
  - Total Cost: US$220,000
  - EFF also made the entire design documents publicly available:
    - **"Cracking DES: Secrets of Encryption Research, Wiretap Politics & Chip Design" by Electronic Frontier Foundation, John Gilmore (Ed), Publisher:** O'Reilly & Associates; May 98
  - With the design done, only need US$150,000 to replicate a machine (in 1998)
  - Average time of search: 4.5 days/key then
  - Moorse's Law: hardware price/performance improving 40% per year, keys must grow by about 1-bit every 2 years
    - DES was designed in 1979, assuming 56-bit key was just sufficient, 64-bit is about right in 1995 and 128 bits would suffice until 2123
- Conclusion: DES is now too easy to crack to be a useful encryption method


LOOK INSIDE!
Cracking DES
Secrets of Encryption Research, Wiretap Politics & Chip Design

Minimum Key-length requirement

- Recommended by an NAS/NRC expert panel in 1996:



Minimum key length for symmetric-key ciphers

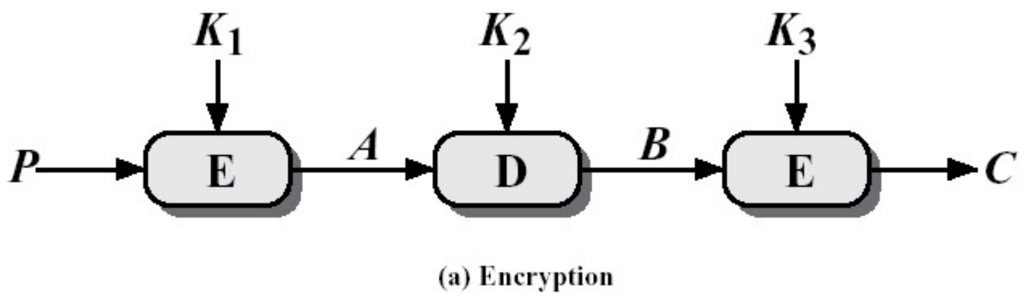
Intruder	Budget	Tools	Time		Secure key length
			40 bits	56 bits	
Hacker	tiny	**PC**	**1 week**	infeasible	45
Small business	$400	FPGA	5 hrs	38 years	50
	$10,000	FPGA	12 min	18 months	55
Corporate department	**$300 K**	**FPGA**	**24 sec**	19 days	60
		ASIC	18 sec	**3 hrs**	
Big company	$10 M	FPGA	7 sec	13 hrs	70
		ASIC	5 ms	6 min	
Intelligence agency	$300 M	ASIC	0.2 ms	12 sec	75

- What should be the corresponding Secure key lengths in 2018 ?

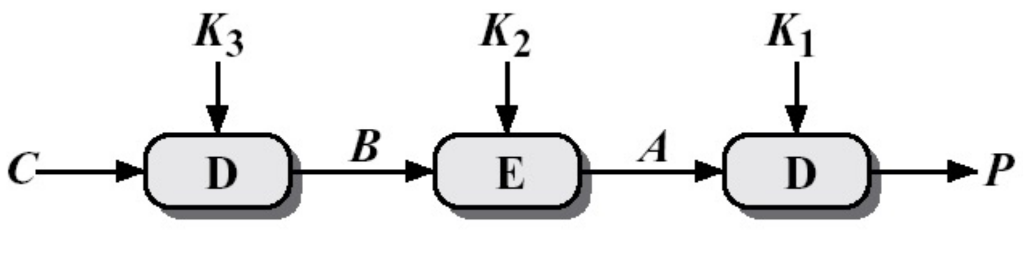
What should we do then ?

- Increase the effective key-length of DES by doing multiple DES with different keys => also mean slow down the encryption for multiple times

=> Here comes the "Triple DES" or 3DES *where* $C = E_{K_3}[D_{K_2}[E_{K_1}[P]]]$



(a) Encryption

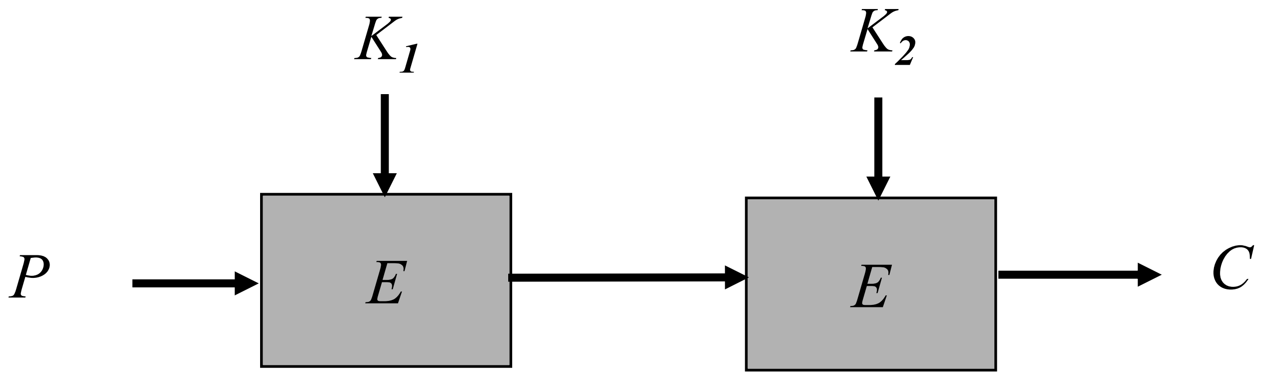


(b) Decryption

- The actual standard requires K3 to be equal to K1 => 112-bit key-length which is deemed to be sufficiently secure
- The use of E-E-E would still have worked. But using E-D-E instead of E-E-E enhance backward compatibility with DES by setting K1 equal to K2

Why not 2DES ?

- If a 112-bit key is already deemed to be secure enough, why not just doing DES encryption twice with 2 different keys, i.e. $C = E_{K_2}[E_{K_1}[P]]$?



- Because this is susceptible to a "Meet-in-the-Middle" attack which reduces the effective key-length to about 57-bit only.

Meet-in-the-Middle Attack on 2DES

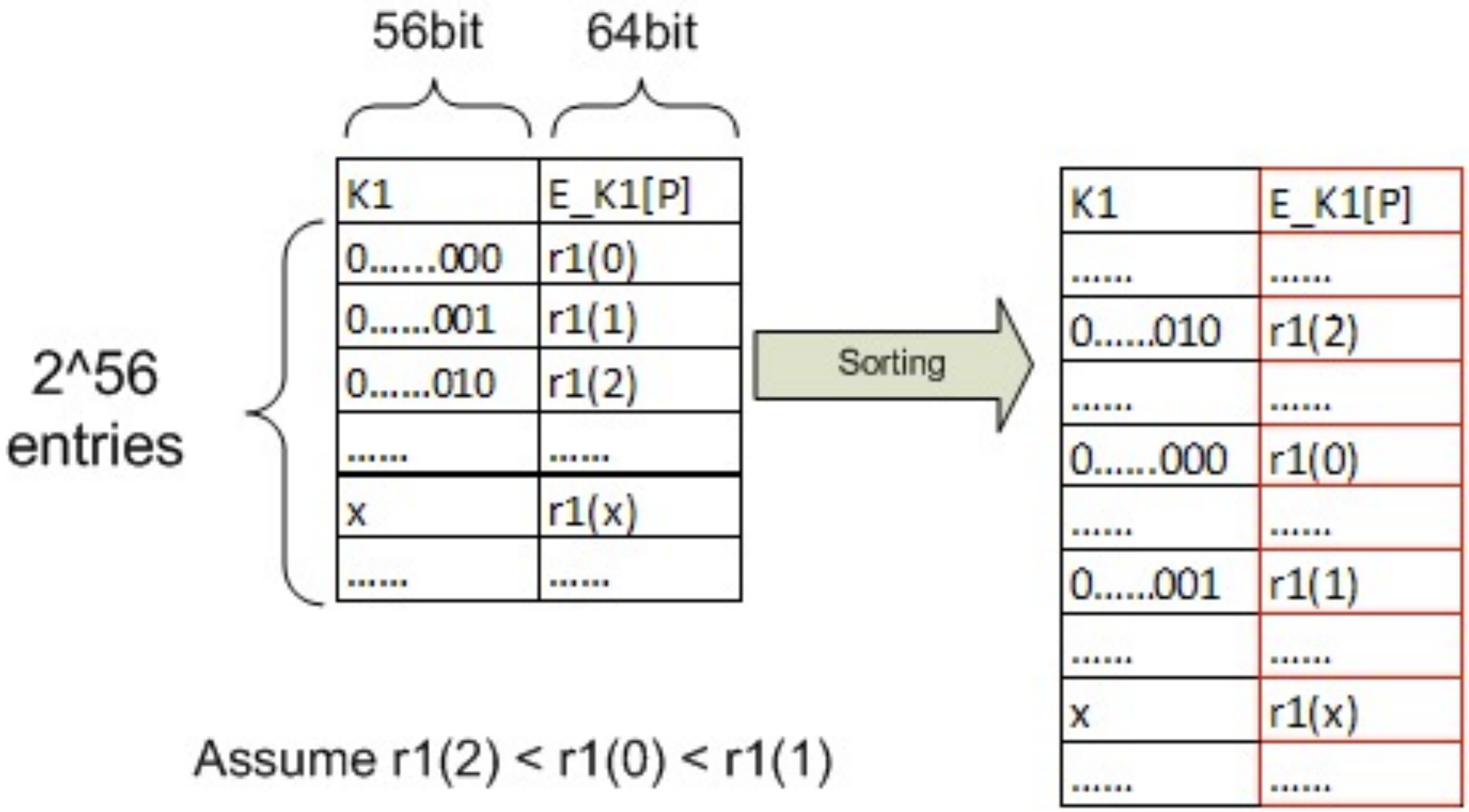
Assume the hacker has a few <plaintext,ciphertext> pairs, e.g. <*p1,c1*>, <*p2,c2*>, <*p3,c3*> where $ci = E_{K_2}[E_{K_1}[pi]]$

Key Observation: $D_{K_2}[c1] = E_{K_1}[p1]$ if K1 and K2 are the right keys.

Attack Steps:

- First make Table A with 2^{56} entries, where each entry consists of a DES key K and the result *r* of applying that key to encrypt *p1.* Then sort the table in numerical order by *r*
- Make Table B with 2^{56} entries, where each entry consists of a DES key K and the result *r* of applying that key to decrypt *c1.* Also sort Table B in numerical order by *r*
- Search through the sorted tables to find matching entries $<K_A,r>$ from Table A and $<K_B,r>$ from Table B. Each match provides K_A as candidate K1 and K_B as a candidate K2 because $D_{K_B}[c1] = E_{K_A}[p1]$
- If multiple candidate pairs of K_A and K_B are found in Step 3, use each candidate key-pair to encrypt p2, p3, …etc to see if it results in c2, c3,…; The "real" key-pair will always work ; the other "coincident" key-pairs will almost surely fail on at least one of the other <pi,ci> pairs

Meet-in-the-Middle Attack on 2DES (cont'd)

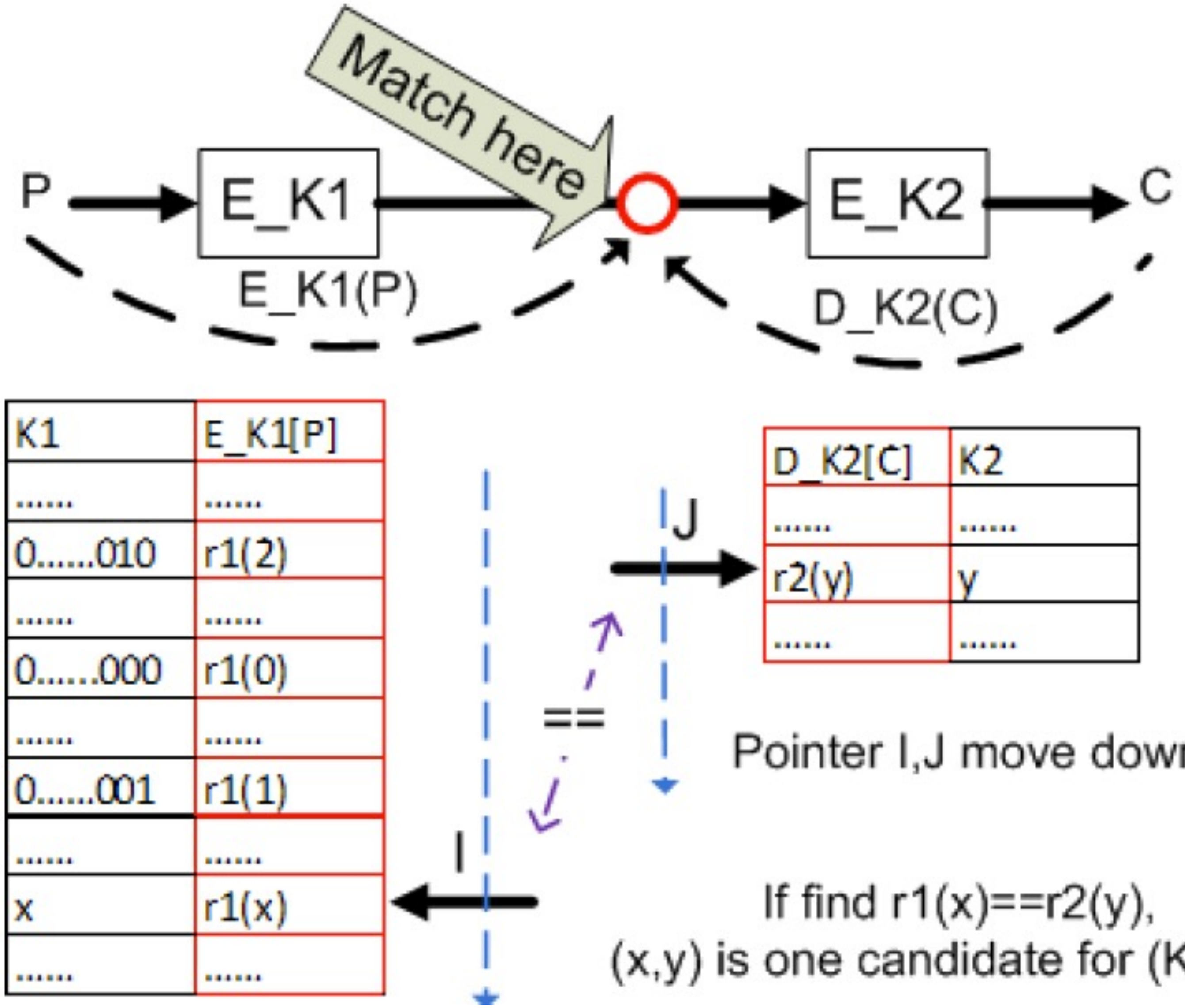


K1	E_K1[P]
0……000	r1(0)
0……001	r1(1)
0……010	r1(2)
……	……
x	r1(x)
……	……

K1	E_K1[P]
……	……
0……010	r1(2)
……	……
0……000	r1(0)
……	……
0……001	r1(1)
……	……
x	r1(x)
……	……

Assume r1(2) < r1(0) < r1(1)

Meet-in-the-Middle Attack on 2DES (cont'd)



K1	E_K1[P]
......	......
0......010	r1(2)
......	......
0......000	r1(0)
......	......
0......001	r1(1)
......	......
x	r1(x)
......	......

D_K2[C]	K2
......	......
r2(y)	y
......	......

Pointer I,J move down

If find r1(x)==r2(y),
(x,y) is one candidate for (K1,K2)

NIST Contest for Advanced Encryption Standard (AES)

- NIST had an open call for proposals, actually a contest, in 1997
  - 21 submissions from all over the world ; 15 fulfilled all the requirement (8 from North America, 4 from Europe, 2 from Asia, 1 from Australia)
  - Narrow down to five final candidates in August 1999:
    - Rijndael (Belgium), Serpent (England, Israel, Norway), MARS (IBM) , Twofish (US), RC6 (US)
- After vigorous evaluation and testing, **Rijndael** [rain´ dow] was selected as the winner in 2000 and Standardized as AES effective May 2002
- By two Belgium cryptographers: Joan Daeman and Vincent Rijmen
- AES is expected to replace DES and 3DES as “The Standard” encryption work-horse world-wide

NIST AES Evaluation Criteria

- Resistance to known attacks and randomness tests
- Complexity
- Efficient hardware and software implementation
- Flexibility, i.e. can be parameterized easily

Comments on AES Finalists

- after testing and evaluation, shortlist in Aug-99:
  - MARS (IBM) - complex, fast, high security margin
  - RC6 (USA) - v. simple, v. fast, low security margin
  - Rijndael (Belgium) - clean, fast, good security margin
  - Serpent (Euro) - slow, clean, v. high security margin
  - Twofish (USA) - complex, v. fast, high security margin
- saw contrast between algorithms with
  - few complex rounds verses many simple rounds
  - which refined existing ciphers verses new proposals

Rijndael (AES)

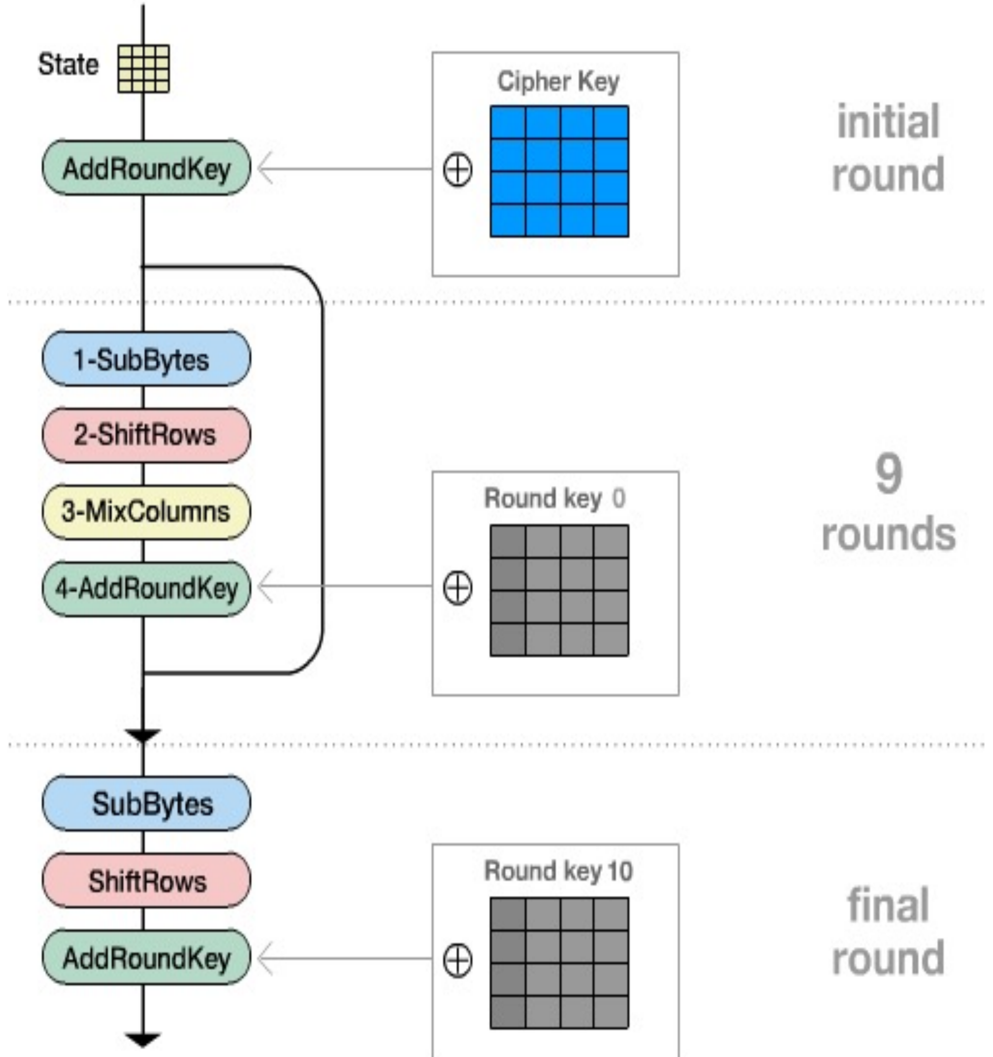
- Symmetric block cipher (AES standard uses 128 bits per block, the Rijndael support multiple block sizes) that can be reconfigured to support key lengths of 128, 192, 256
- Number of Rounds depends on key length
- Unlike several other NIST AES contest finalists, Rijndael does not use the Feistel structure ; instead, it relies on some special properties in "Generalized Field" mathematics for computing the "inverse", i.e. , decryption.
  - Unlike DES, the algorithm/hardware for Rijndael encryption and decryption process are not identical, but differ slightly.
- Software Implementation of Rijndael performs well across a wide range of platforms, from 8-bit (Smartcard like) to 64-bit CPU, e.g. support 24+ Mbps encryption/decryption on a 200MHz Pentium Pro, Borland C++
- Fastest in Hardware amongst all the finalists ; ASIC Hardware implementation by NSA demonstrated performance ranging from 443 to 606 Mbps, depending on key-length and mix of key scheduling

Overview of AES

4 transformations for Each Round:

- Substitute Bytes
- Shift Rows
- Mix Columns
- Add Round Key

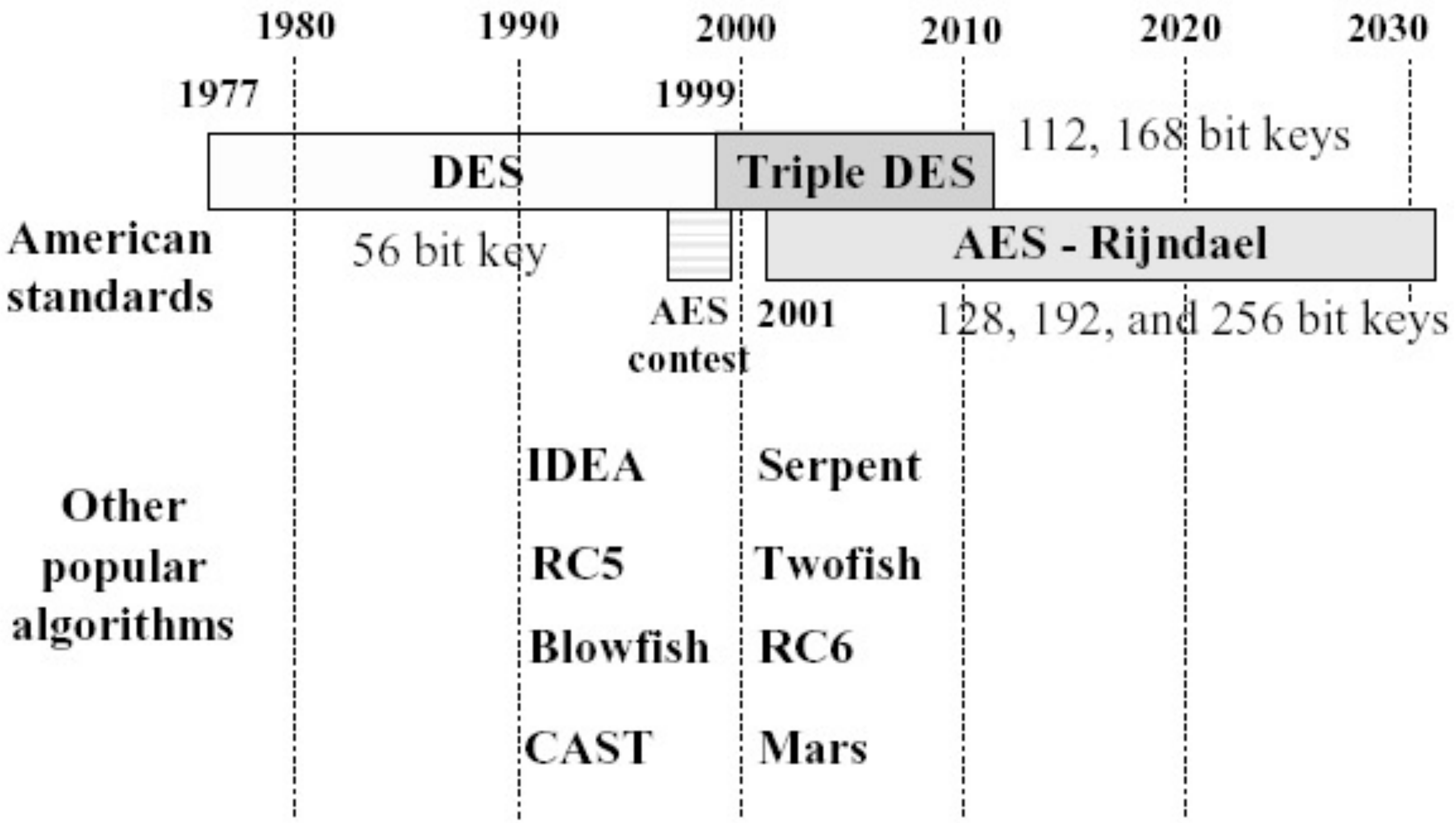
- Number of rounds depends on key length (10,12,14 rounds for 128,192,256-bit keys respectively



AES URLS

- http://csrc.nist.gov/CryptoToolkit/aes/rijndael/ - NIST AES
- http://www.esat.kuleuven.ac.be/~rijmen/rijndael/ - Rijndael Home Page
- http://www.esat.kuleuven.ac.be/~rijmen/rijndael/Rijndael_Anim.zip - Great Animation

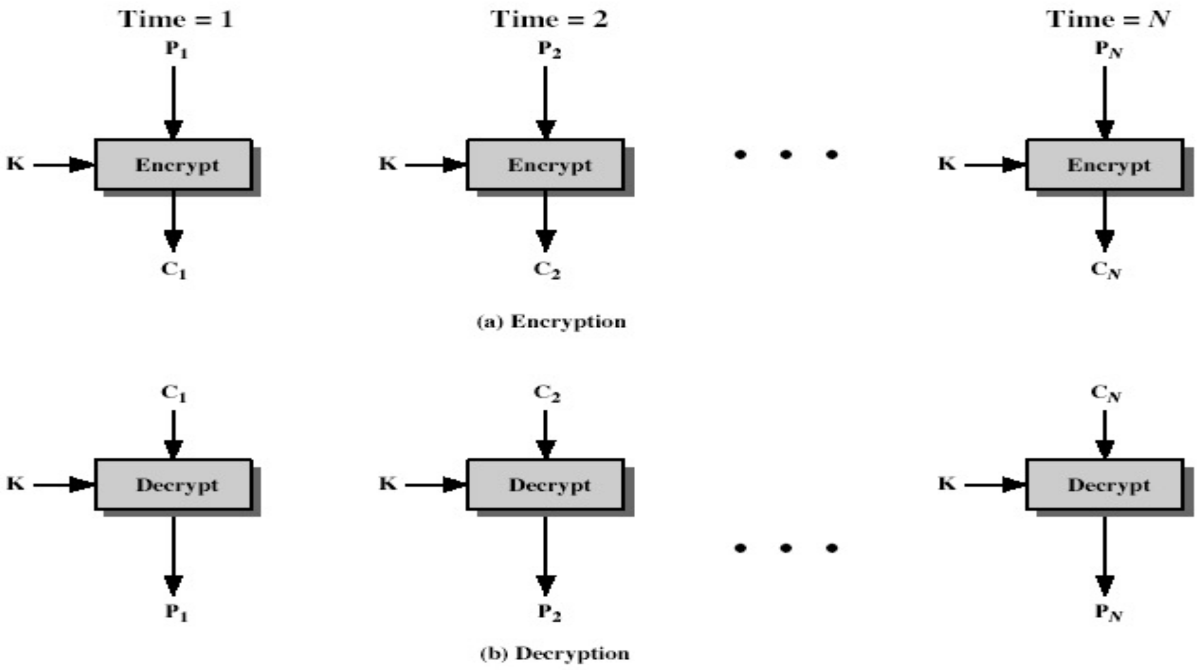
Chronology of some popular Secret-Key Encryption Algorithms



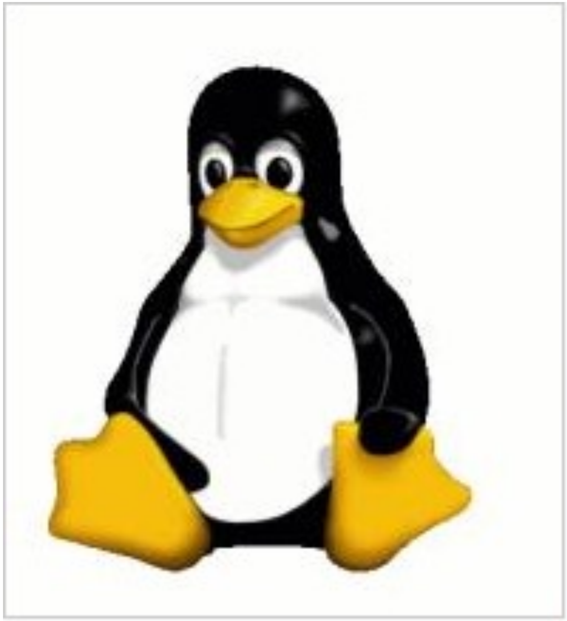
Modes of Operation for Block Cipher

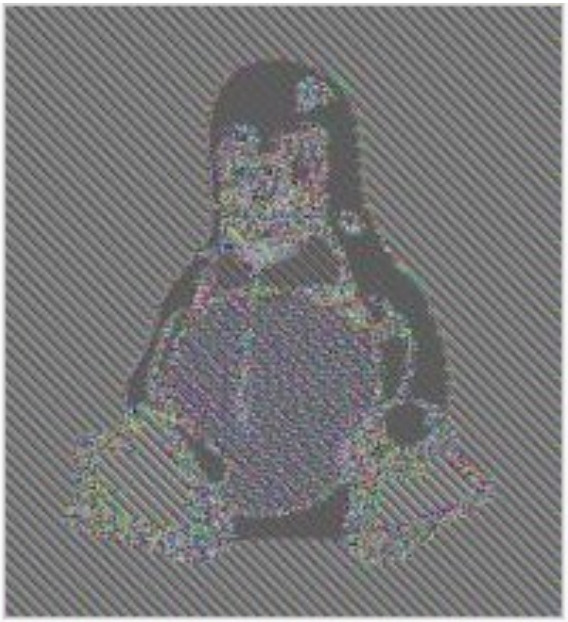
- Break the piece of plaintext into **64-bit blocks ;** pad the last block to 64 bit
- Basic Mode of Operation: Use **Electronic Code Book** (ECB)
  - Each block of plaintext is encrypted *independently* using the same key
  - Repeated plaintext block will produce the same ciphertext block

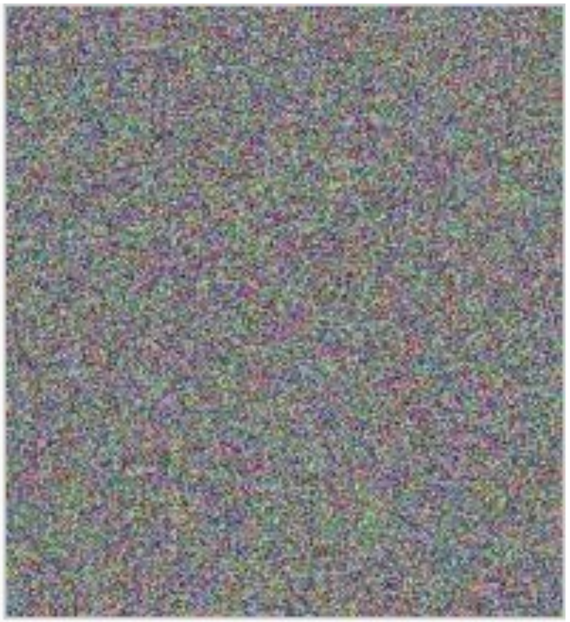
    => can leak information
  - Blockwise swapping of ciphertext may still produce meaningful output upon decryption
  - Mainly used for sending a small number of blocks of information only



Information Leakage with ECB


Original image


Encrypted using ECB mode


Modes other than ECB result in pseudo-randomness

Source: *https://words.filippo.io/the-ecb-penguin/*

Cipher Block Chaining (CBC) Mode

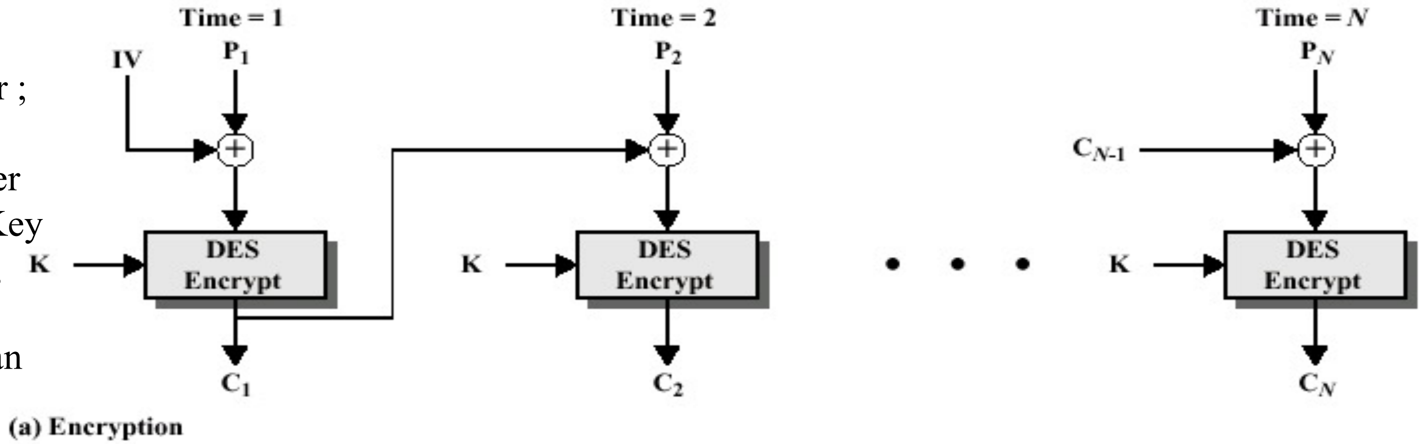
- Input to algorithm is the **XOR** of current plaintext block and preceding ciphertext block
- **Repeating patterns** are **not** exposed
- But what if the ciphertext is corrupted or lost during transmission ?

IV stands for Initialization Vector ;

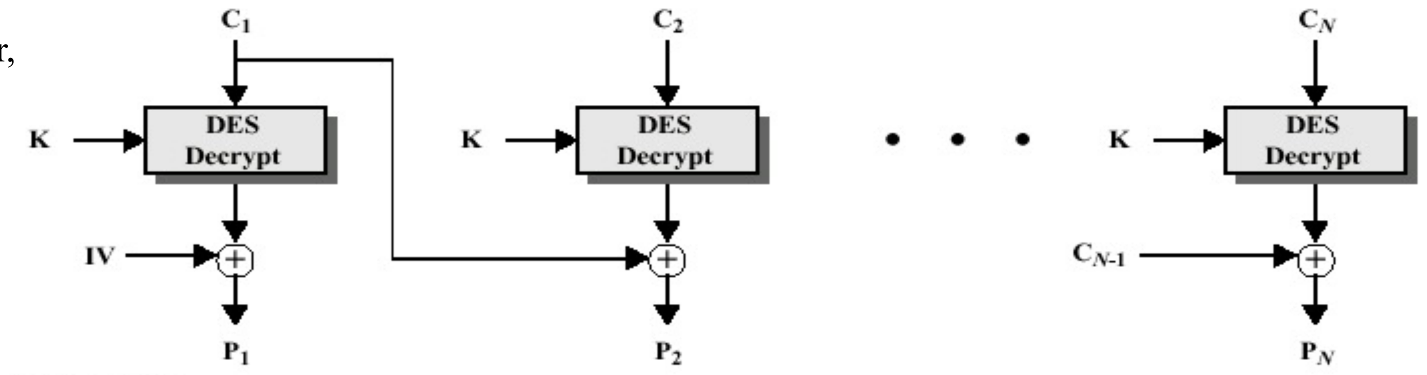
Sender and Receiver Need to share the Key and IV in advance ;

An encrypted IV can be Sent in advance

If IV is sent in clear, Attacker can manipulate P1 and compensate by modifying the IV



(a) Encryption



(b) Decryption

Cipher Block Chaining (CBC) Mode (cont'd)

- The last encrypted block C_N, aka the CBC residue, can be used as a "Message Authentication Code" (MAC) for a message as follows:

1. The sender transmits the original message in plaintext together with the the CBC residue (but NOT the key, of course)
2. The receiver, who knows the key in advance, can then encrypt the plaintext upon its arrival using CBC mode. If the message has been tampered with during transmission, the CBC residue won't match !

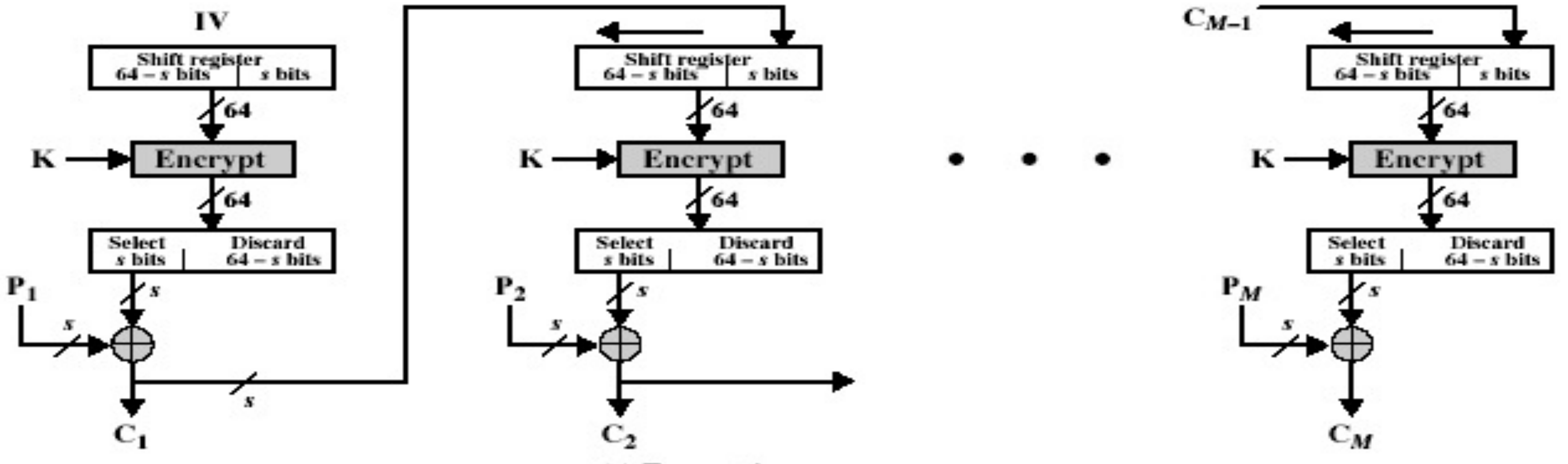
- Notice in this case, CBC is used for MAC purpose and does NOT provide secrecy at all ;
- If both secrecy and message-authenticity (tamper-proof) is required, we need to do CBC twice in 2 passes with 2 different keys:
  - 1^{st} pass for encryption,
  - 2^{nd} pass to generate the CBC-residue for MAC.
- Why is it insufficient to just append the CBC residue of the 1^{st} pass as the MAC ?

Insecurity of MAC-then-Encrypt mode of Cipher Block Chaining (CBC)

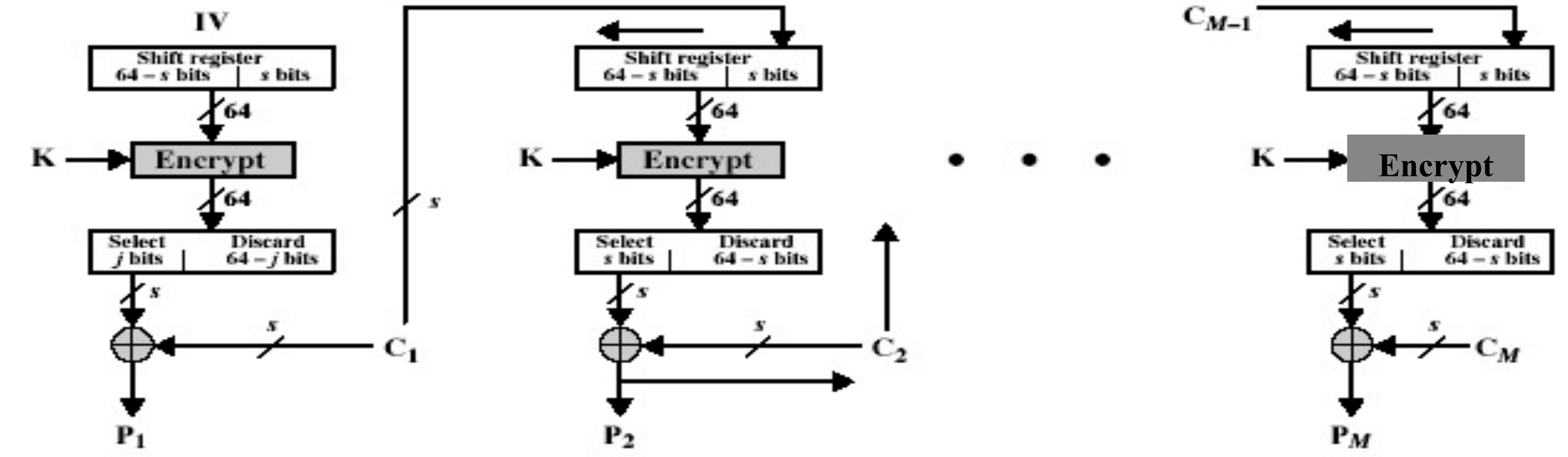
- Which one is better: MAC-then-Encrypt or Encrypt-then-MAC ?
- TLS Ver1.1 chose MAC-then-Encrypt (MtE), namely
  1. Authenticate (protect the integrity of) the plaintext (using HMAC) ;
  2. Add the HMAC code at the end of the message ;
  3. Pad the message length to the required block-length ;
  4. Encrypt the resultant block using a block cipher, e.g. AES ;
- **Unfortunately, such implementations of MAC-then-Encrypt mode of CBC modes, in TLS-1.1, 1.2, are subject to various variants of Padding-Oracles+Timing attacks,**
  - **e.g. Vaudenay Padding Oracle (2002), Lucky 13 (2013), BEAST (2011), etc,**

    https://blog.cloudflare.com/padding-oracles-and-the-decline-of-cbc-mode-ciphersuites/
  - **CBC MtE modes have been depreciated (i.e. removed, disallowed) by new TLS standards (v1.3)**
  - https://www.cloudflare.com/learning-resources/tls-1-3/

Cipher FeedBack (CFB) Mode



(a) Encryption



(b) Decryption

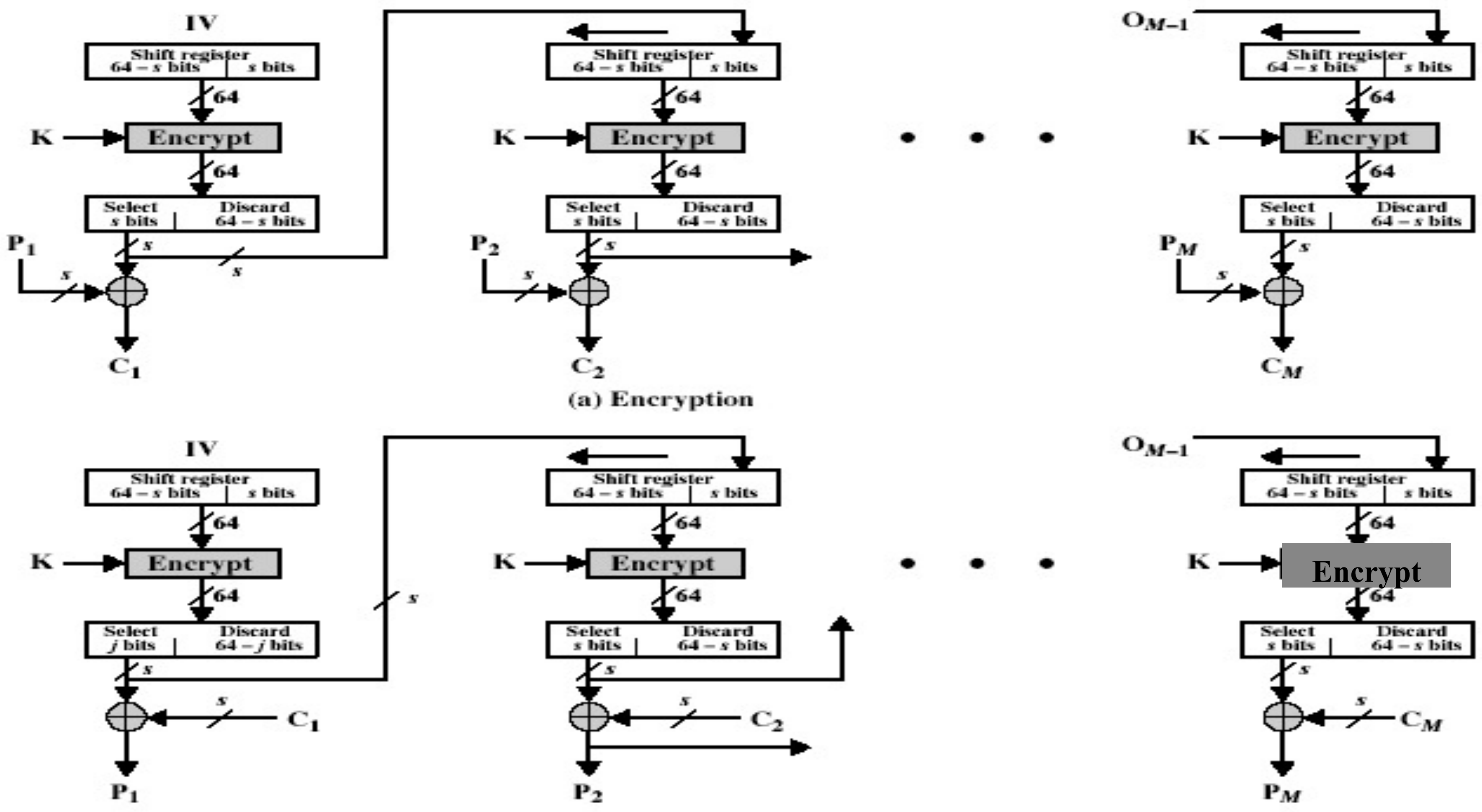
Cipher Feedback Mode

- Convert DES into a **stream cipher**
- **Eliminates** need to **pad** a message
- Operates in **real time**
- Each character can be **encrypted** and **transmitted immediately**
- message is treated as a stream of bits
- added to the output of the block cipher
- result is feed back for next stage (hence name)
- standard allows any number of bit (1,8 or 64 or whatever) to be feed back
  - denoted CFB-1, CFB-8, CFB-64 etc
- is most efficient to use all 64 bits (CFB-64)
- uses: stream data encryption, message authentication

Advantages and Limitations of CFB

- appropriate when data arrives in bits/bytes
- most common stream mode
- limitation is need to stall while do block encryption after every s-bits
- note that the block cipher is used in **encryption** mode at **both** ends
- errors propagate for several blocks after the error

Output FeedBack (OFB)



(a) Encryption

(b) Decryption

Output FeedBack (OFB)

- message is treated as a stream of bits
- output of cipher is added to message
- output is then feed back (hence name)
- feedback is independent of message
- can be computed in advance
- uses: stream encryption over noisy channels
- How about message authentication ?

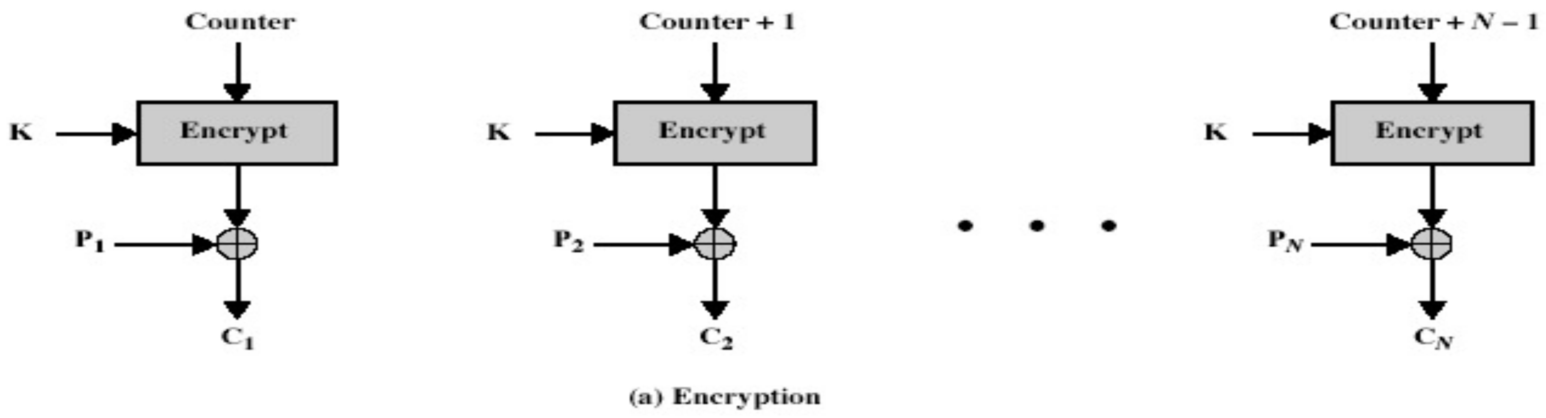
Advantages and Limitations of OFB

- used when error feedback a problem or where need to encryptions before message is available
- superficially similar to CFB
- but feedback is from the output of cipher and is independent of message
- must **never** reuse the same sequence (key+IV)
- sender and receiver must remain in sync, and some recovery method is needed to ensure this occurs
- originally specified with m-bit feedback in the standards
- subsequent research has shown that only **OFB-64** should ever be used

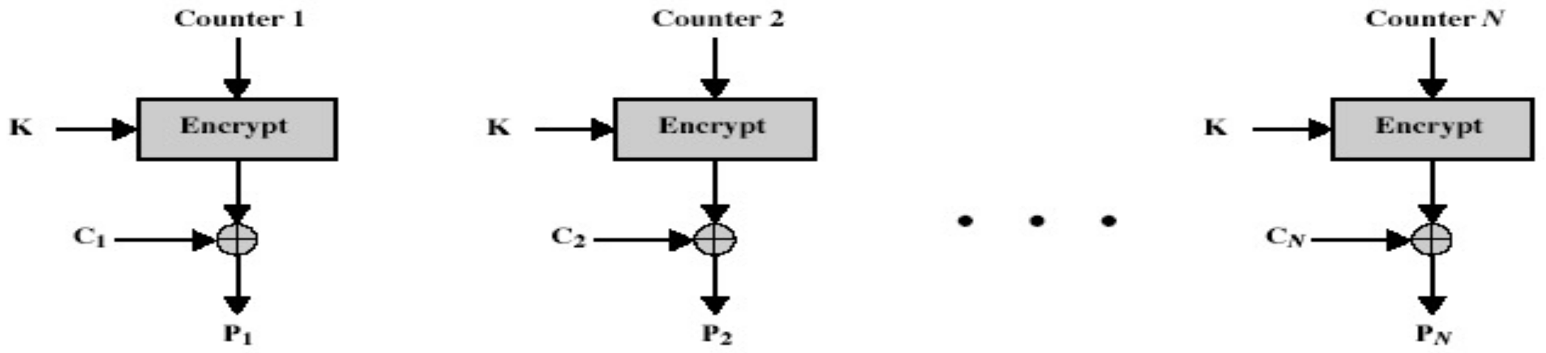
Counter (CTR)

- a “new” mode, though proposed early on
- similar to OFB but encrypts counter value rather than any feedback value
- must have a different key & counter value for every plaintext block (never reused)
- uses: high-speed network encryptions

Counter (CTR)



(a) Encryption



(b) Decryption

Advantages and Limitations of CTR

- efficiency
  - can do parallel encryptions
  - in advance of need
  - good for bursty high speed links
- random access to encrypted data blocks
- provable security (good as other modes)
- but must ensure never reuse key/counter values, otherwise could break (cf OFB)

Birthday Attacks on CBC, CFB, OFB, CTR

- For a lot of modes of operations for Block Cipher, we must ensure not to reuse key/ counter values for different plaintexts, otherwise information can be leaked ;

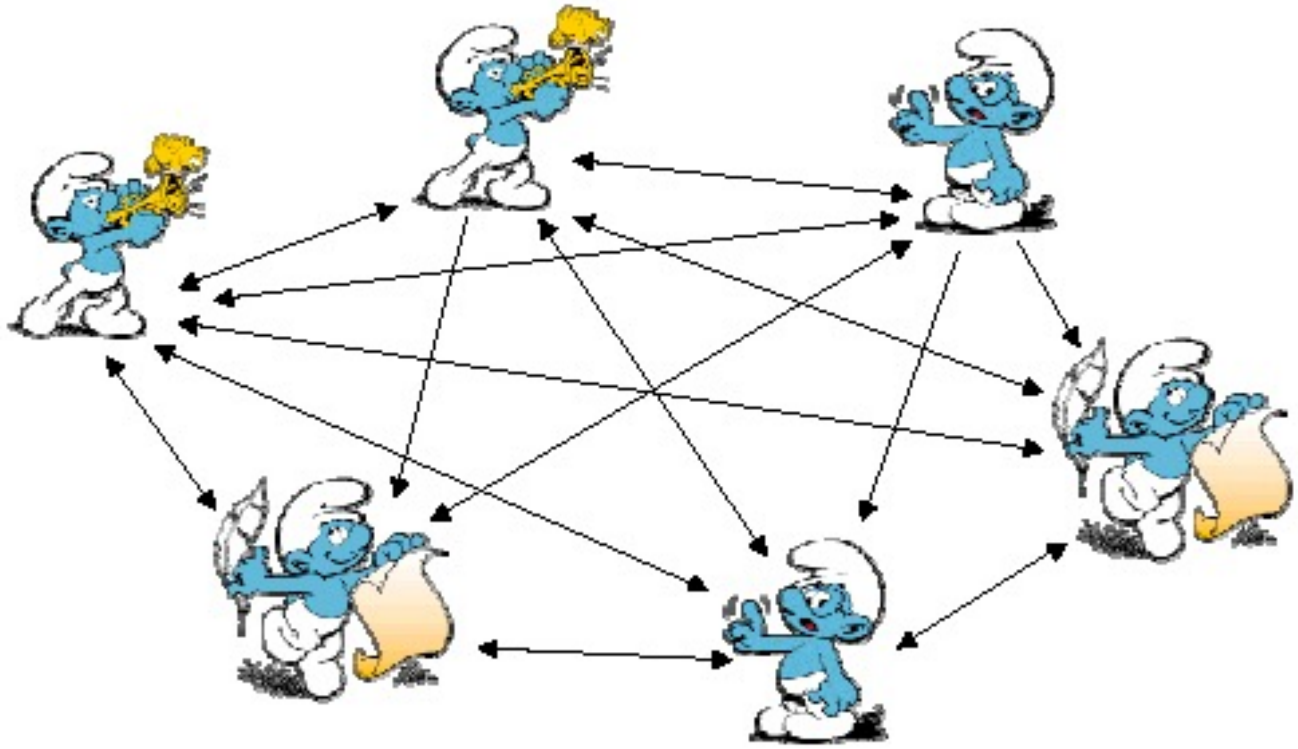
=> CBC, CFB, OFB and CTR modes of operation for Block ciphers are subject to so-called Birthday Attacks:

  - To stay safe, re-keying is required before encrypting $2^{(\text{block-size} / 2)}$ of input blocks, e.g. for DES, or 3DES, it means rekeying more frequent than 2^{32} input blocks

  Refer to the following for further details:

  - Sweet32 paper in CCS 2016 https://sweet32.info/SWEET32_CCS16.pdf
  - D. McGrew 2012 paper https://eprint.iacr.org/2012/623

Key Distribution Problem for Secret Key Crypto-systems



N - Users ⇨ $\frac{N \cdot (N-1)}{2}$ Keys

Users	Keys
100	5,000
1000	500,000

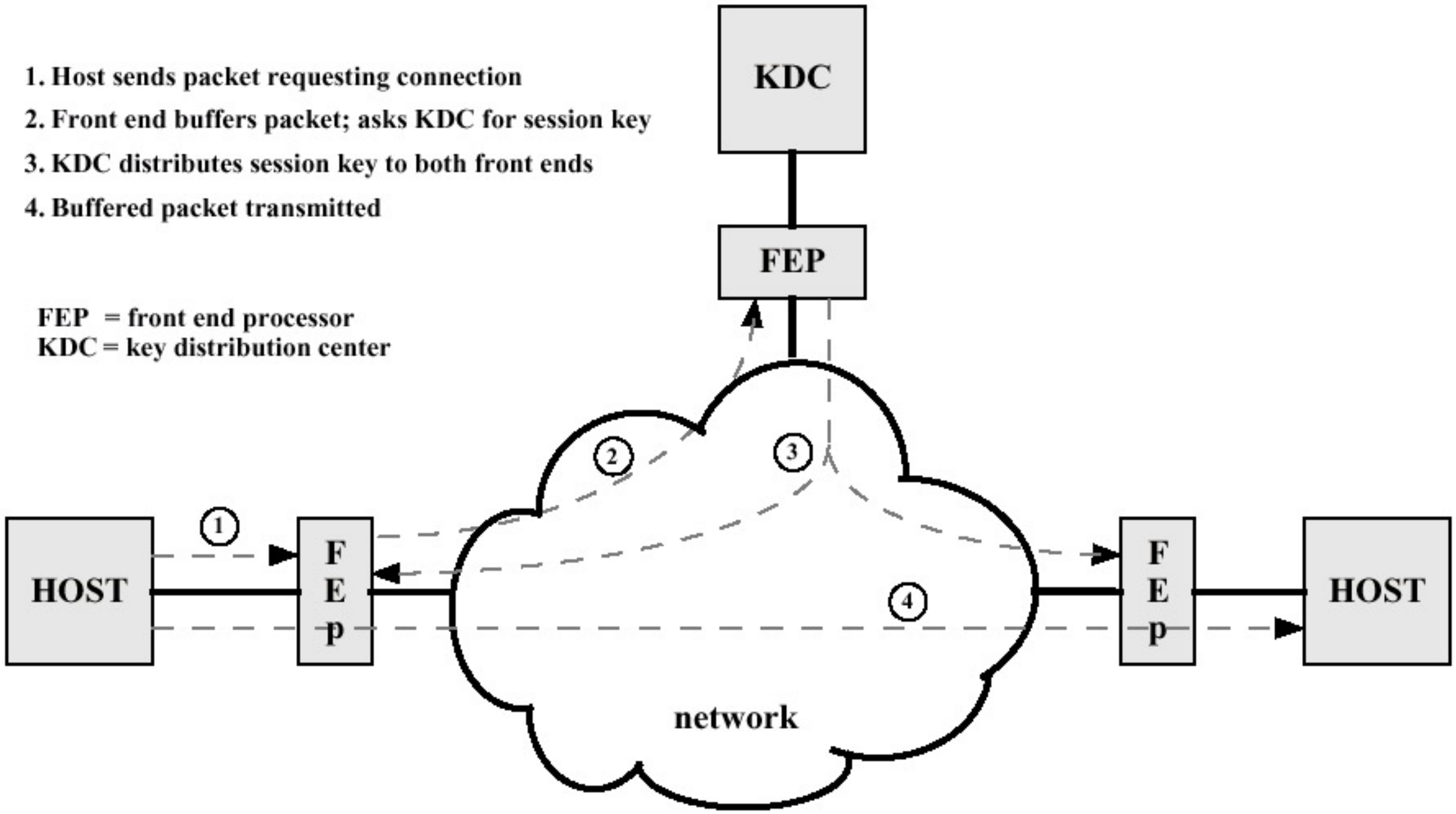
Key Distribution

- **Both parties** must have the **secret key**
- Key is **changed frequently**
- Requires either manual **delivery** of keys, or a third-party encrypted channel
- Most effective method is a **Key Distribution Center** (e.g. Kerberos)
- More later in the course…

Key Distribution

1. Host sends packet requesting connection
2. Front end buffers packet; asks KDC for session key
3. KDC distributes session key to both front ends
4. Buffered packet transmitted

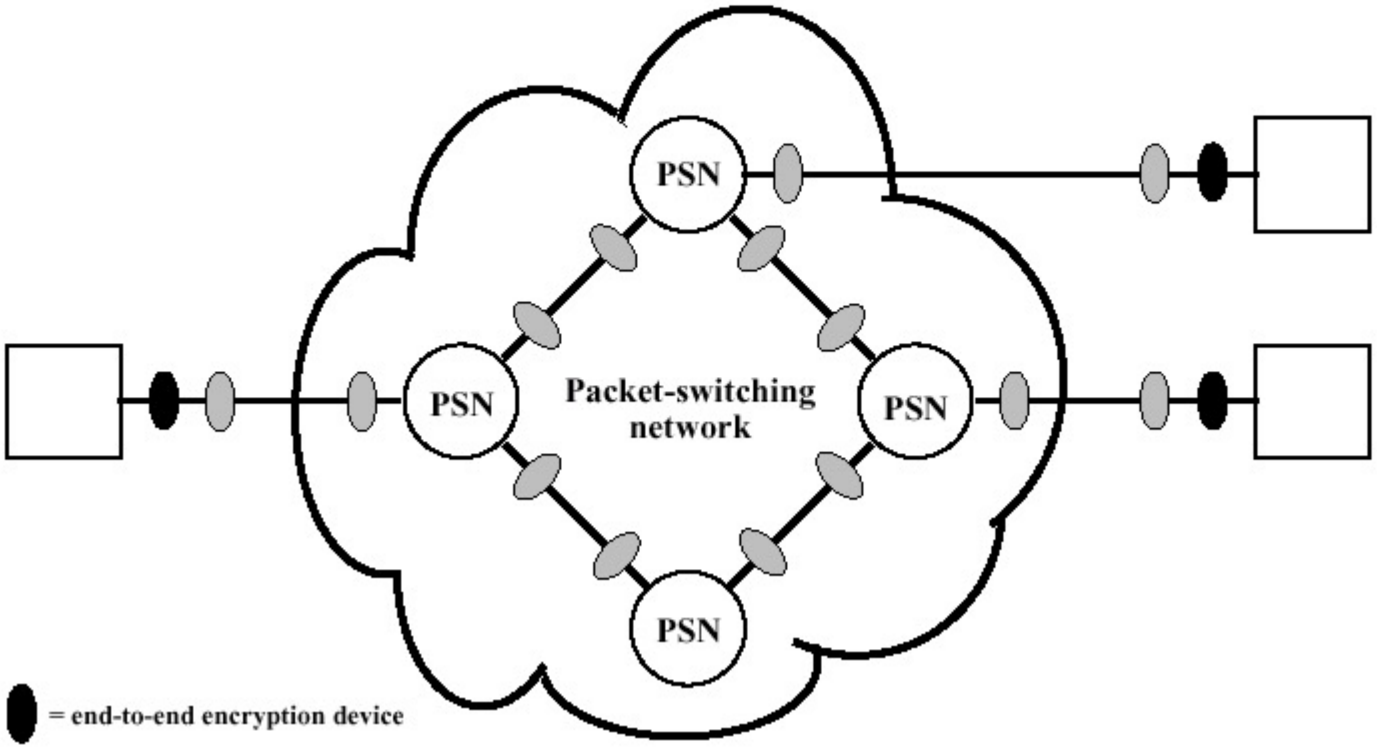
FEP = front end processor
KDC = key distribution center



Location of Encryption Devices

- **Link Encryption**
  - Each vulnerable communications link is equipped on both ends with an encryption device
  - All traffic over all communications links is secured
  - Vulnerable at each switch
- **End-to-end Encryption**
  - The encryption process is carried out at the two end systems
  - Encrypted data are transmitted unaltered across the network to the destination, which shares a key with the source to decrypt the data
  - Packet headers cannot be secured

Location of Encryption Devices



● = end-to-end encryption device

⬮ = link encryption device

PSN = packet switching node