Introduction to Symmetric Encryption

Ibéria Medeiros

Departamento de Informática

Faculdade de Ciências da Universidade de Lisboa

FBI Seeks US $38 Million to Help Defeat Encryption

- *The FBI is requesting US $38 million to address the problems associated with "going dark" (i.e., not being able to read communications). The money will be used to develop and purchase tools to access encrypted data during investigations, something Director James Comey has repeatedly warned is stymying investigations. That amount is 23 percent greater than the amount the agency spent last year in its battle with encryption*

NOTE: **under the proper conditions**, law enforcement agencies do need access to suspect communications

NOTE1: FBI and other agencies buying **tools and/or services to read encrypted communications** is less costly overall in the long run than **forcing weaknesses to be built into cryptography**

Implementing/Managing Cryptography is Hard!

- Cryptographic issues **ranked higher in prevalence** than historically common flaws like cross-site scripting, SQL injection and directory traversal. They included things like:
  - improper TLS (Transport Layer Security) certificate validation,
  - cleartext storage of sensitive information,
  - missing encryption for sensitive data,
  - hard-coded cryptographic keys,
  - inadequate encryption strength,
  - insufficient entropy,
  - non-random initialization vectors,
  - improper verification of cryptographic signatures,
  - … and more.

Read more in: https://info.veracode.com/state-of-software-security-report-volume6.html

Basic Concepts

- Encryption algorithms can be classified accordingly to:

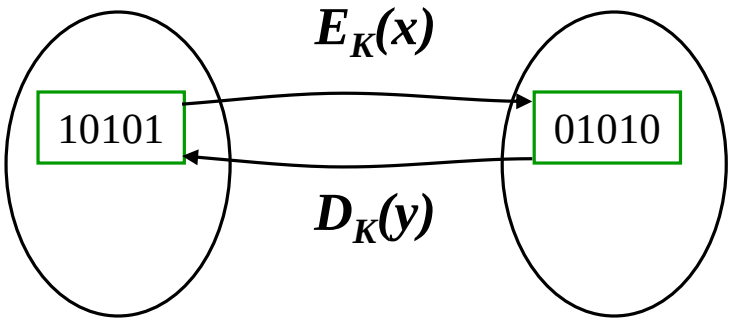
1. *Basic types of operations used to transform plaintext into ciphertext*
   - substitution
   - transposition (or permutation)

This class we will focus on …

2. *Number of keys*
   - symmetric cipher: 1 key
   - asymmetric (or public-key) cipher: 2 keys

3. *How to process the plaintext*
   - block cipher
   - stream cipher

Encryption Algorithms with Symmetric Key



1. The encryption function E takes an input x and produces and output y
2. E_K has to be bijective function, and each K should define a distinct bijective function

- Assumptions that one should consider for the adversary
  - has access to all encrypted data that gets transmitted
  - knows all details about the encryption/decryption function with the exception of the key (*Kerchhoffs hypothesis*)
- From these assumptions results that the security of the cryptographic system depends fundamentally on two things: the ***quality of the algorithm*** and the ***key size***

Attacks on the Algorithm (Cryptanalysis)

- **Cryptanalysis** is based on the nature of the algorithm plus some knowledge of the plaintext (e.g., statistical tests)
  - aims either to get the *plaintext* or the *key* given the ciphertext
  - general attack classes based on the amount of info known to the adversary
    - » *ciphertext-only attack* - *chosen-plaintext attack*
    - » *known plaintext* - *chosen-ciphertext attack* - *chosen text*
- **Unconditionally secure:** the generated ciphertext does not contain enough information to determine uniquely the corresponding plaintext, no matter how much ciphertext is available
  - only achieved with a ***one-time pad scheme***
- **Computationally secure** if either are true
  - the cost of breaking the cipher exceeds the value of the encrypted information, *or/and*
  - the time required to break the cipher exceeds the useful lifetime of the information

Attacks on the Key

- If one assumes that the encryption algorithm is secure, then the only way to attack the cryptographic system is by ***brute force***

  NOTE: there were several cases where this idea was shown to be invalid!

  Example: some concerns were raised due to some tweaking done by NSA on the Windows usage of DUAL_EC_DRNG, which is an implementation of a NIST standard specified in Special Publication 800-90

- **Attack on the key with known-plaintext**
  - one knows a few pairs of plaintext and the corresponding ciphertext *How?*
  - keys are tried (eventually all keys) until we find a key that converts the plaintext into ciphertext
  - the attack complexity for a key of size k bits is 2^k tests, with 50% probability of finding the right key with 2^{k-1} tries

    Example: consider a computer that can test a million keys per second; if the key has 56 bits, then it would take around 2285 years to discover the key

Attack Cost Estimates (Stallings)

One recent very fast PC

One recent supercomputer

Key size (bits)	Cipher	Number of Alternative Keys	Time Required at 10^9 decryptions/s	Time Required at 10^{13} decryptions/s
56	DES	$2^{56} \approx 7.2 \times 10^{16}$	2^{55} ns = 1.125 years	1 hour
128	AES	$2^{128} \approx 3.4 \times 10^{38}$	2^{127} ns = 5.3×10^{21} years	5.3×10^{17} years
168	Triple DES	$2^{168} \approx 3.7 \times 10^{50}$	2^{167} ns = 5.8×10^{33} years	5.8×10^{29} years
192	AES	$2^{192} \approx 6.3 \times 10^{57}$	2^{191} ns = 9.8×10^{40} years	9.8×10^{36} years
256	AES	$2^{256} \approx 1.2 \times 10^{77}$	2^{255} ns = 1.8×10^{60} years	1.8×10^{56} years
26 characters (permutation)	Monoalphabetic	$26! = 4 \times 10^{26}$	2×10^{26} ns = 6.3×10^{9} years	6.3×10^{6} years

1. estimated time of the universe is **10^{10} years**
2. due to Moore´s law, the cost should go down by a factor of 10 every 5 years
3. if the encryption algorithm is not perfect (which is normal), then all these numbers stop making sense

Some Examples of Brute Force Attacks

- Exploit CPU lost cycles: utilize the unused computer time to test in parallel many keys

  (Example: 400 computers, each one with 32000 encryptions per second, would take one day to find a key if it has 40 bits (the key size for the exported software from the USA a few years ago))

- Worm: develop a worm that instead of encrypting the disk, it tests a set of keys and if it finds the correct key it sends it to the adversary

  (Example: for the same problem, a million computers would take 1/2 minute)

- Chinese lotery machine: the Chinese government installs in each radio and television a chip capable of testing keys

(Example: the chip tests 1 million encryptions per second)

Country	Population	# of Televisions/Radios	Time to Break	
			56-bit	64-bit
China	1,190,431,000	257,000,000	280 seconds	20 hours
U.S.	260,714,000	739,000,000	97 seconds	6.9 hours
Iraq	19,890,000	4,730,000	4.2 hours	44 days
Israel	5,051,000	3,640,000	5.5 hours	58 days
Wyoming	470,000	1,330,000	15 hours	160 days
Winnemucca, NV	6,100	17,300	48 days	34 years

(All data is from the *1995 World Almanac and Book of Facts.*)

How big should be the key?

- It depends on factors like
  - how valuable is the data
  - for how long does it have to be kept secret
  - who is our opponent

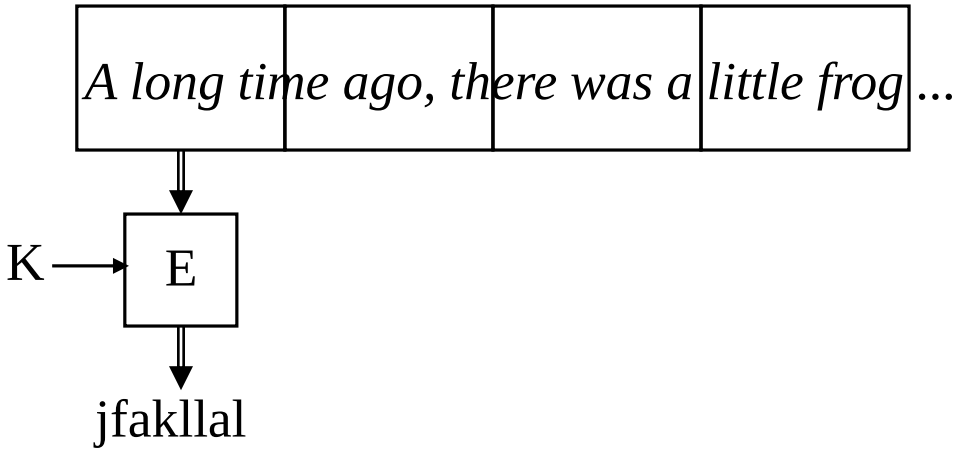
Type of Traffic	Lifetime	Minimum Key Length
Tactical military information	minutes/hours	56–64 bits
Product announcements, mergers, interest rates	days/weeks	64 bits
Long-term business plans	years	64 bits
Trade secrets (e.g., recipe for Coca-Cola)	decades	112 bits
H-bomb secrets	>40 years	128 bits
Identities of spies	>50 years	128 bits
Personal affairs	>50 years	128 bits
Diplomatic embarrassments	>65 years	at least 128 bits
U.S. census data	100 years	at least 128 bits

What is the largest size of key that makes sense to consider?

(SYMMETRIC) BLOCK CIPHER

Principles of Block Cipher Algorithms

The basic algorithm encrypts block by block of the plaintext



- Defines a transformation of a block of *n* bits into a block of *n* bits
- Each one of the 2^n different blocks has to be transformed into a distinct block in order to be possible to invert the transformation

Reversible Transformation

Plaintext	Ciphertext
00	11
01	10
10	00
11	01

Irreversible Transformation

Plaintext	Ciphertext
00	11
01	10
10	**01**
11	**01**

Basic Principles (cont.)

- If we can use all possible distinct reversible transformations that exist (which are $2^n!$ transformations) then we have an ***ideal block cipher algorithm***

- What is a good value of ***n*** (i.e., how big should be the block)?
  - for a small *n* there is a reduced number of available transformations

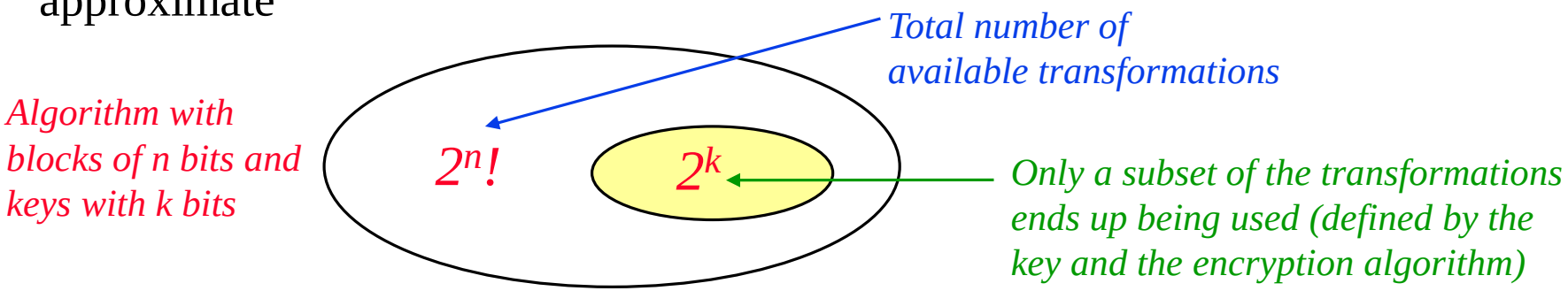
    $\Rightarrow$ one can perform a brute force attack where all transformations are tried

    $\Rightarrow$ statistical attacks can be performed because encryption $\approx$ substitution
  - to ensure an acceptable level of security, *n* has to have a reasonable number of bits $\Rightarrow$ the size of the block cannot be too small
  - for a very large *n* there is waste of space for small plaintexts $\Rightarrow$ the size of the block cannot be too big

- How can we specify the transformation?
  - one possibility is through a substitution table with 2^n entries that maps each plaintext block to the corresponding ciphertext block (but it is *not scalable*)

Basic Principles (cont.)

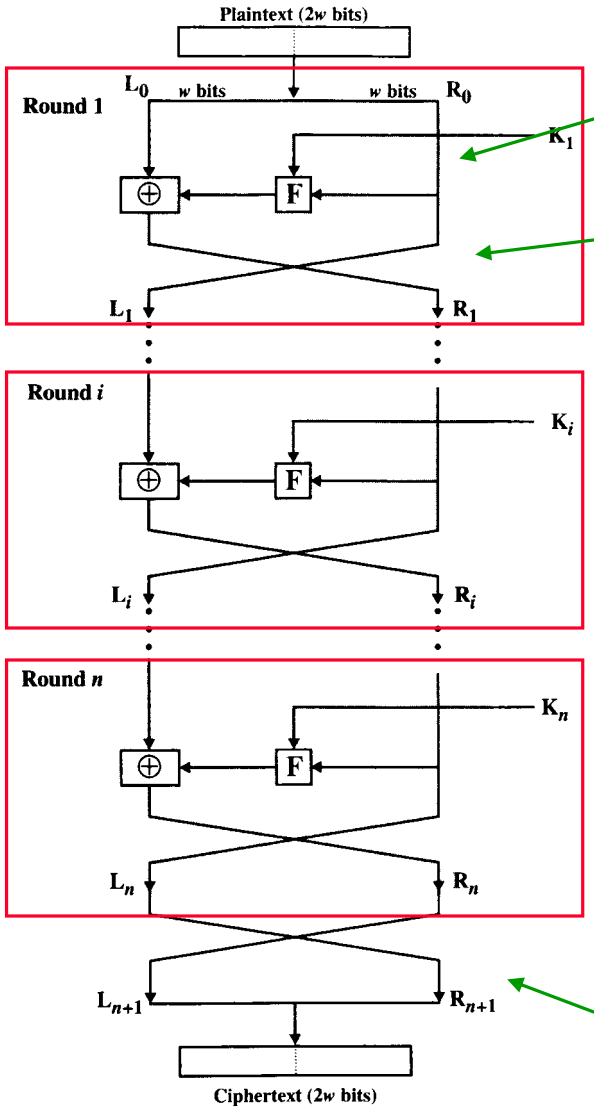
- In practice it is not possible to use a substitution table, and therefore one needs to approximate



- Substitution cipher: substitutes an element by another one (e.g., aac → BBD)
- Transposition / permutation cipher: rearrange the elements (e.g., aac → CAA)
- Product cipher: combination of two (or more) basic transformations such that the resulting transformation is cryptographically stronger than the individual ones
- Feistel cipher : repeats a number of times a product of ciphers based on the basic transformations of substitution and transposition

  NOTE: many of the past encryption algorithms are based on the Feistel cipher

Feistel Structure



Substitution is applied only to half of the block, and is based on the operation $F(R_i, K_i) \oplus L_i$

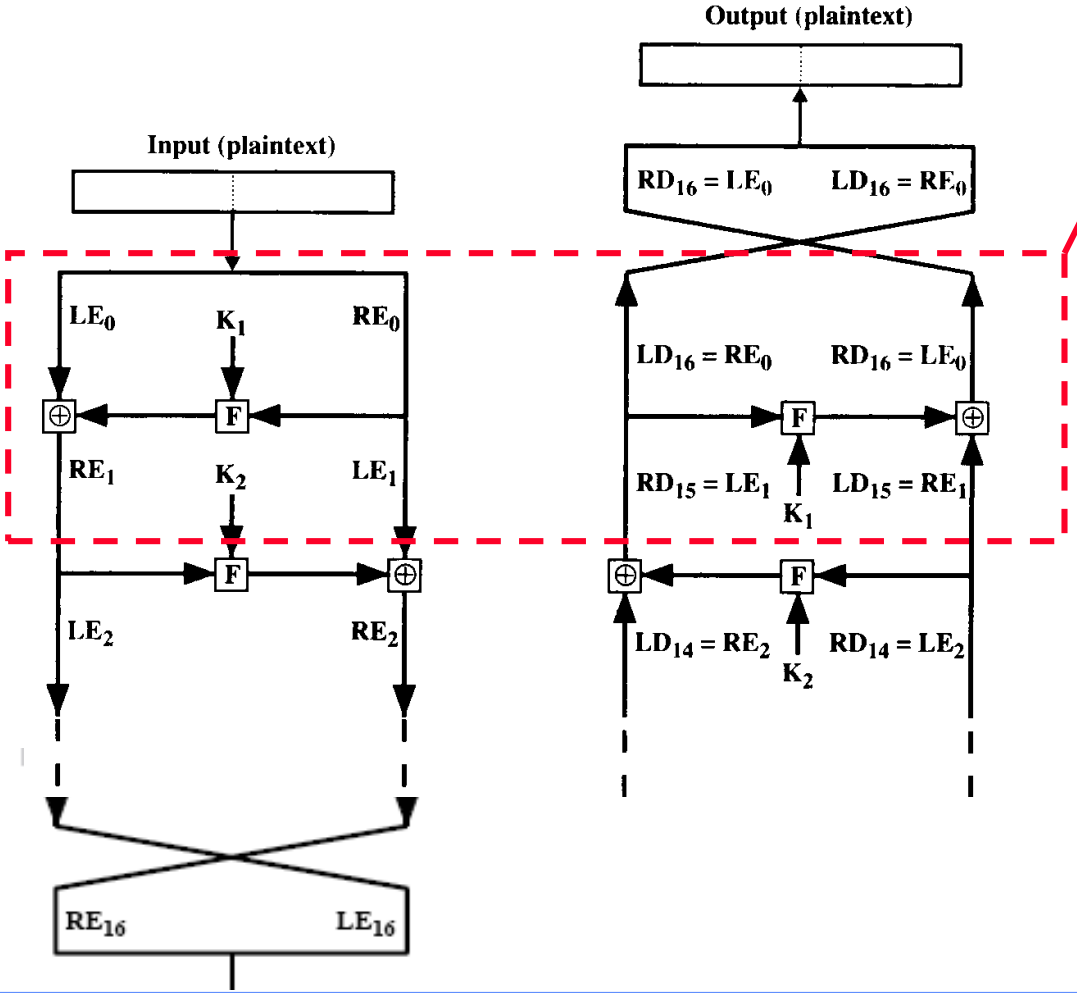
Permutation is basically the exchange of the two halves of the block (F() can also contain other permutations)

- The exact implementation depends
  - *block size*: larger results in higher security but in slower execution
  - *key size*: larger improves security but may increase execution time
  - *number of iterations*: larger gives more security but reduces performance
  - *sub-key generation algorithm*
  - *function F()*

Last permutation to simplify decryption

Encrypt and Decrypt with a Feistel Cipher

- The decryption procedure is essentially the same as to encrypt, with the difference that sub-keys are utilized in the reverse order



LE1 = RE0
RE1 = LE0 ⊕ F(RE0, K1)

Therefore, taking into consideration the last permutation in the encryption:

LD16 = RD15 = LE1 = RE0
RD16 = LD15 ⊕ F(RD15, K1)
= RE1 ⊕ F(LE1, K1) =
= [LE0 ⊕ F(RE0, K1)] ⊕ F(RE0, K1)
= LE0

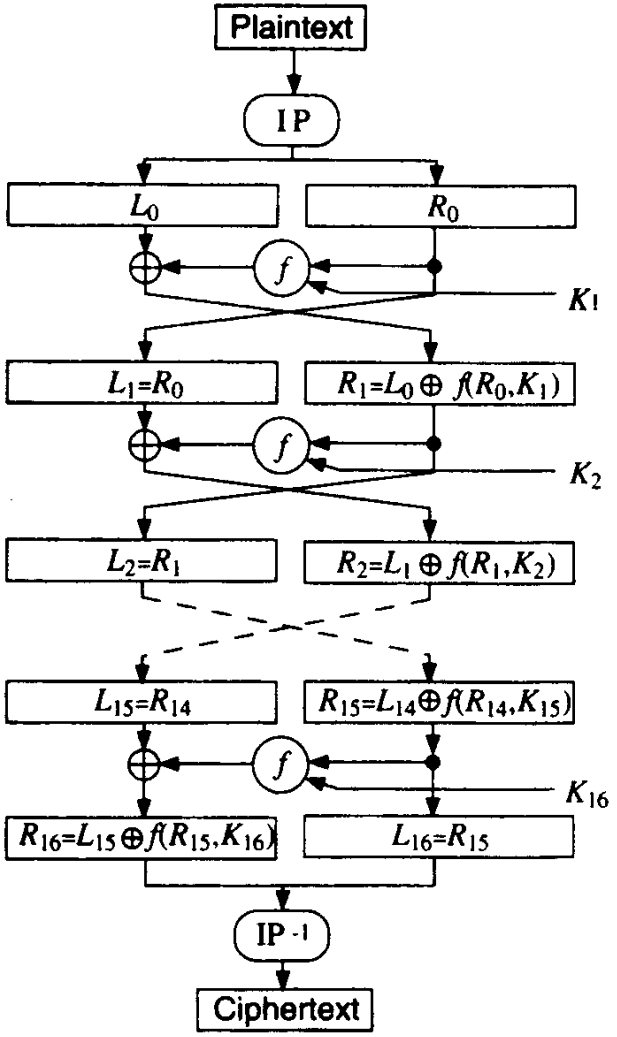
With the last permutation in the decryption we obtain the original data

General Principals for the Design

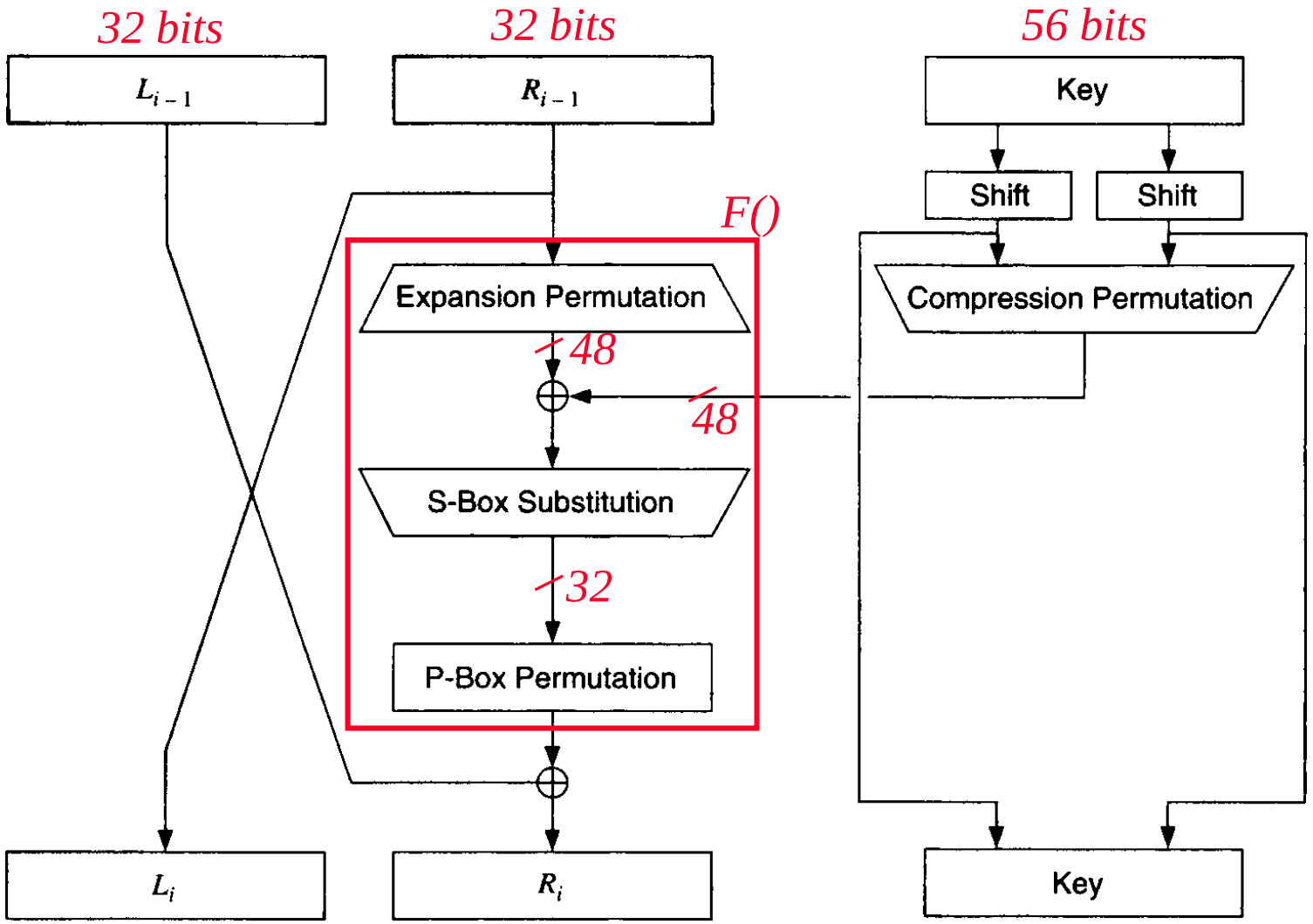
- Number of rounds
  - should be chosen in such a way that the cryptanalytic effort requires greater effort than simple brute-force key search
  - in the case of DES, *differential cryptanalysis* requires $2^{55.1}$ operations, whereas brute-force requires 2^{55}
- Design of function F
  - should be difficult to find how to undo the substitution, e.g., should be non-linear
  - good avalanche properties, meaning that the change of one bit in the input should produce the change of many bits in the output & the change of output bits *j and k* should be independent when any input bit *i* is flipped
- Key schedule algorithm
  - should make difficult to find the *key* or other *subkeys*
  - good avalanche properties as above should be guaranteed

Example: *Data Encryption Standard (DES)*

- Basically the same algorithm is used to encrypt and decrypt (there are only a few minor differences)
- All security is determined by the key
- The algorithm is based on substitution and permutation operations
- Each block of 64-bits of data is encrypted into 64-bits of ciphertext
- The key is 56-bits (there is small set of "weak" keys)
- The algorithm is executed in 16 iterations



DES Fundamental Block



DES Variants

- **Double DES**

  $C = E_{K2}(E_{K1}(P))$

  $P = D_{K1}(D_{K2}(C))$

  – if the keys do not form a group (i.e., $E_{K2}(E_{K1}(P)) \neq E_{K3}(P)$) then the brute force attack becomes much harder to perform (for a key with *k*-bits one needs 2^{2k} tests)

Meet-in-the-Middle Attack

• we will save more information in memory but we will not need to encrypt the data so many times

- needs 2^{k+1} encryptions (instead of 2^{2k}) and to save 2^k blocks in memory

• *Two pairs (P1, C1) and (P2, C2) are known in advance*
• *For each key K1 calculate and save in memory* $E_{K1}(P1)$
• *For each key K2 calculate* $D_{K2}(C1)$ *and compare with data saved in memory*
• *When there is a match, verify K1 and K2 with P2 and C2; in case it does not give the correct result, restart in the previous step*

DES Variants (cont)

- Triple DES with Two Keys
  - to perform a brute force attack one needs to try 2^{112} keys
  - if K1=K2 then it becomes equivalent to simple DES (reason E-D-E)
  - there have been some attacks described, but none is practical; for example: 2^k pairs of chosen plaintext-ciphertext, 2^{k+1} encryptions and 2^k blocks in memory

$$C = E_{K1}(D_{K2}(E_{K1}(P)))$$
$$P = D_{K1}(E_{K2}(D_{K1}(C)))$$

- Triple DES with Three Keys

$$C = E_{K3}(D_{K2}(E_{K1}(P)))$$
$$P = D_{K1}(E_{K2}(D_{K3}(C)))$$

- Triple DES with Minimal Key

$$K1 = E_{X1}(D_{X2}(E_{X1}(T1)))$$
$$K2 = E_{X1}(D_{X2}(E_{X1}(T2)))$$
$$K3 = E_{X1}(D_{X2}(E_{X1}(T3)))$$

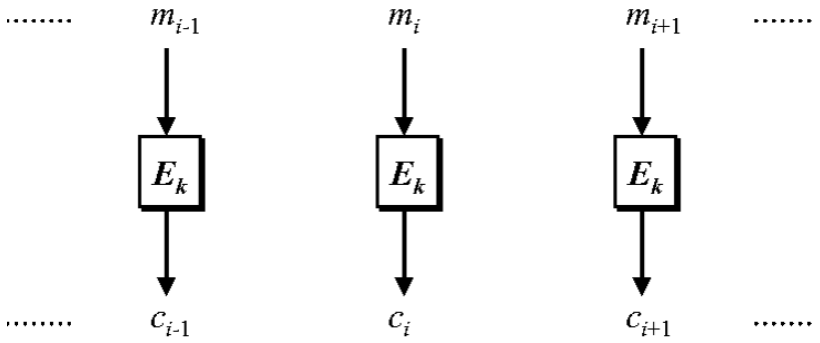
- *X1, X2 are the keys*
- *T1, T2, T3 are constants that do not need to be secret*
- *K1, K2, K3 are the keys to use in the above algorithm*

Cipher Modes

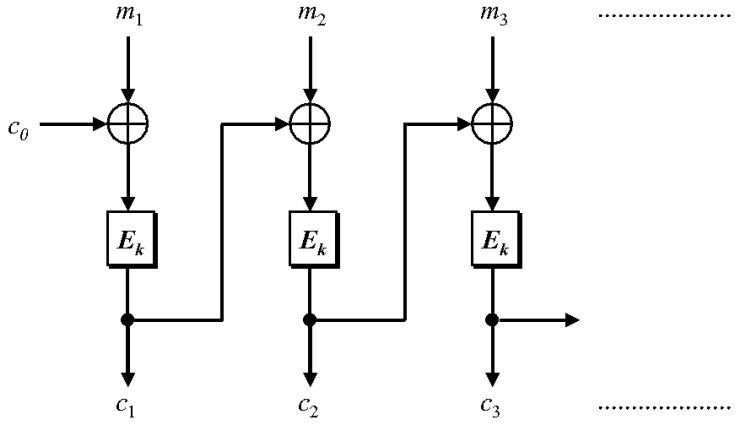
- Take advantage of an existing cryptographic algorithm and adapt its use to particular applications; all security is based on the original algorithm
- Some of the traditional modes of encryption

Mode	Description	Typical Application
Electronic Codebook (ECB)	Each block of plaintext bits is encoded independently using the same key.	•Secure transmission of single values (e.g., an encryption key)
Cipher Block Chaining (CBC)	The input to the encryption algorithm is the XOR of the next block of plaintext and the preceding block of ciphertext.	•General-purpose block-oriented transmission •Authentication

ECB



CBC

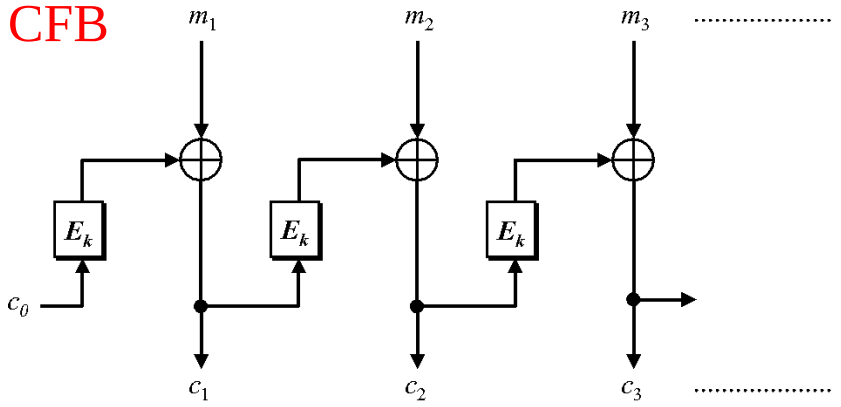


Cipher Modes

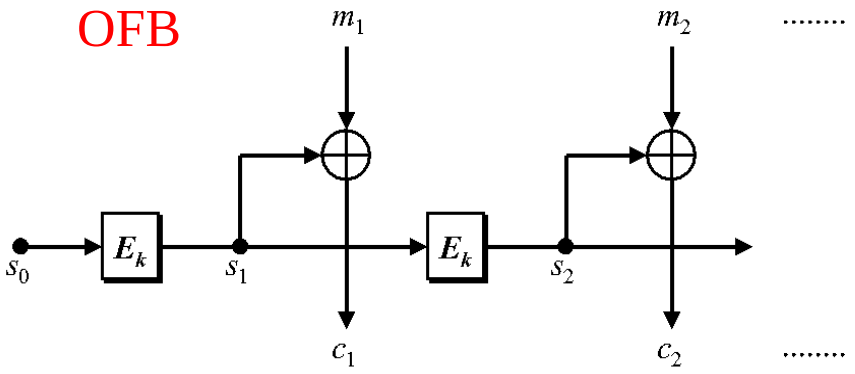
- Take advantage of an existing cryptographic algorithm and adapt its use to particular applications; all security is based on the original algorithm
- Some of the traditional modes of encryption

Mode	Description	Typical Application
Cipher Feedback (CFB)	Input is processed *s* bits at a time. Preceding ciphertext is used as input to the encryption algorithm to produce pseudorandom output, which is XORed with plaintext to produce next unit of ciphertext.	•General-purpose stream-oriented transmission •Authentication
Output Feedback (OFB)	Similar to CFB, except that the input to the encryption algorithm is the preceding encryption output, and full blocks are used.	•Stream-oriented transmission over noisy channel (e.g., satellite communication)

CFB



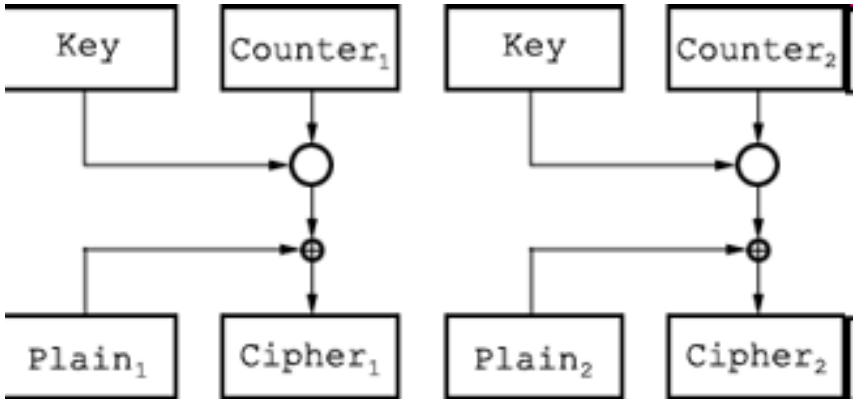
OFB



Cipher Modes

- Take advantage of an existing cryptographic algorithm and adapt its use to particular applications; all security is based on the original algorithm
- Some of the traditional modes of encryption

Mode	Description	Typical Application
Counter (CTR)	Each block of plaintext is XORed with an encrypted counter. The counter is incremented for each subsequent block.	•General-purpose block-oriented transmission •Useful for high-speed requirements

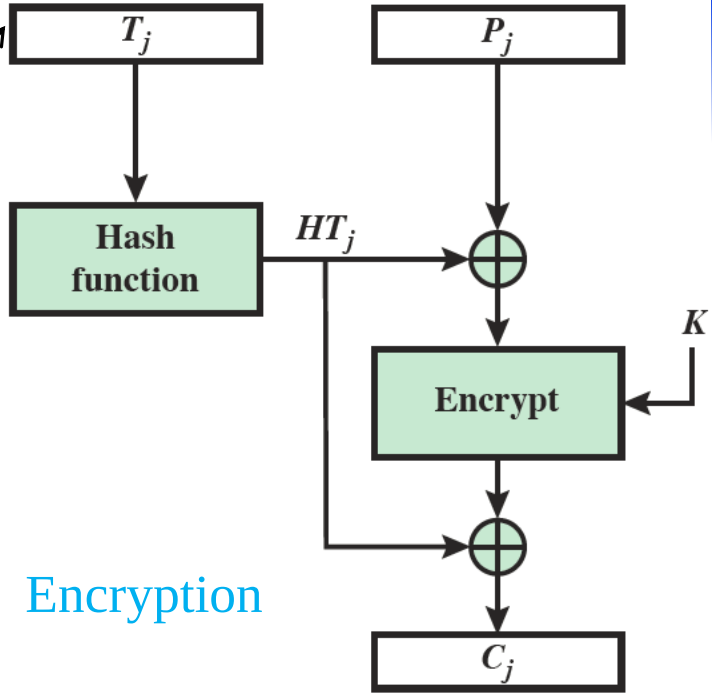


Overview of mode XTS-AES

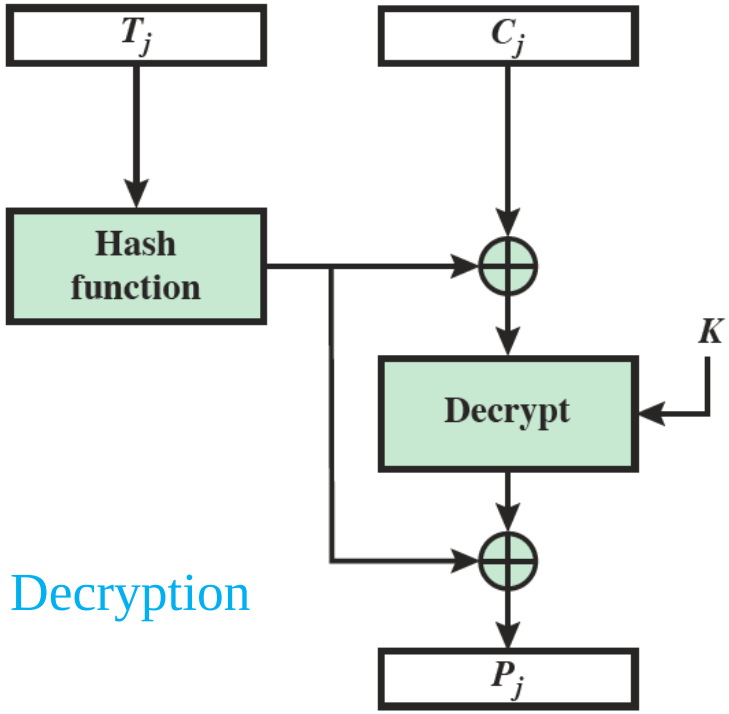
- Used to protect data in sector-based storage devices, where it is assumed that the adversary might have access to the cyphertext
- Based on the concept of ***tweakable block cipher***, but the standard is more complex

C = E(K,T,P)

Tweak may be public but should change in each encrypted block



Encryption



Decryption

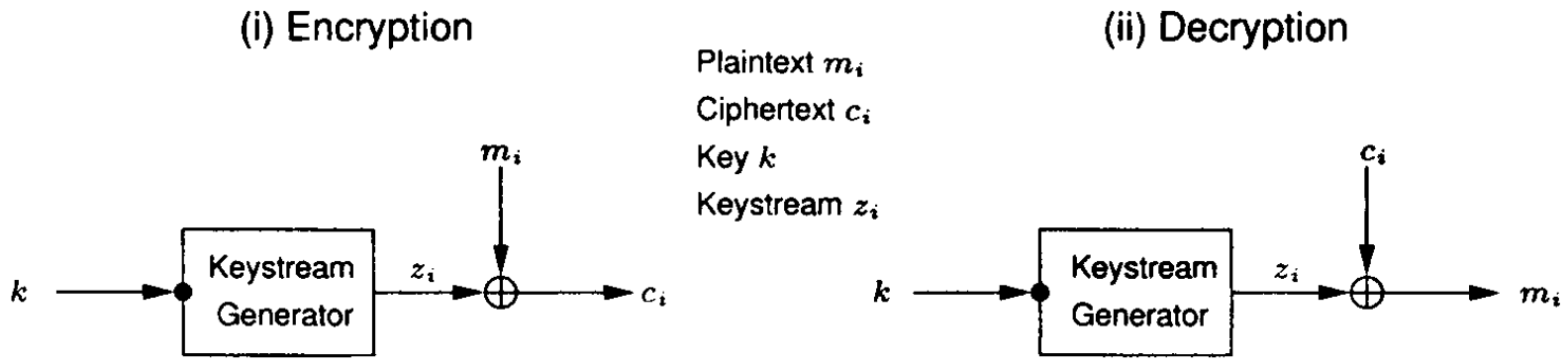
Properties:
- the same plaintext gives different ciphertext

STREAM CIPHERS

Principles of Stream Cipher Algorithms

- Encrypt individual symbols (bits or bytes) of the plaintext using a transformation that changes through time
- Their main advantages are
  - usually they are *faster and simpler* than block cipher algorithms when implemented in hardware
  - they require *very little space in memory* (which can be important for certain telecommunication equipments)
  - they have a *very small error propagation* (which is relevant for environments with lots of noise)
- Their main problems are
  - in many cases they result in inefficient software implementations
  - many algorithms are secret
  - historically several algorithms were crypto analyzed (i.e., broken)

Working Mode

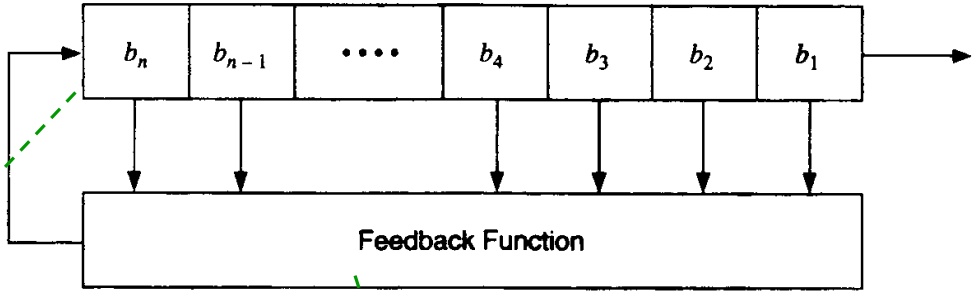


- The ***Keystream Generator*** should
  1. create a sequence of bits with a very large period
  2. the sequence of bits should have properties similar to a *true* random sequence of bits (e.g., number of 1's should be similar to the number of 0)
  3. the input key should have at least 128 bits to avoid brute force attacks

NOTE: The keystream has some similarities to a **one-time pad**

LFSR – *Linear Feedback Shift Registers*

- The majority of hardware implemented stream cipher algorithms are based in LFSR
- A *Feedback Shift Register (FSR)* is composed of two basic elements



Generated bits

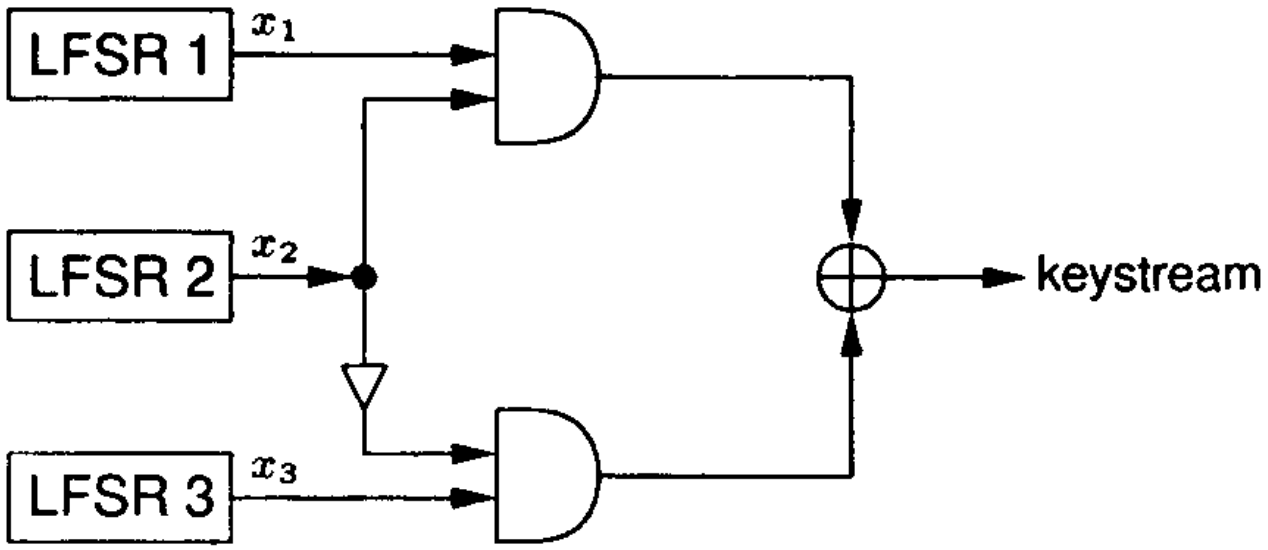
Register that in each iteration moves the bits one position to the right

Function that in each iteration receives as input the current bits in the register, and generates a new bit to be included in the leftmost position of the register

- A *Linear FSR* is a FSR where the function is an XOR of some bits of the register
- A *LFSR* with a register with *n* bits can in theory generate a sequence with $2^n - 1$ pseudo-random bits before it repeats itself

Example: The Geffe Generator

- Over the years, many proposals have appeared that combined the LFSR in different ways to produce a keystream (ideally, more difficult to attack)



RC4 Algorithm

- This algorithm was developed by Ron Rivest in 1987
- Fundamental characteristics
  - works at byte level
  - uses a key with a variable size from 1 to 256 bytes (8 to 2048 bits)
  - very fast in software (8 to 16 machine instructions per each encrypted byte)
  - the period for the pseudo-random bytes (until they return to the beginning) is in the order of 10^{100}
- The algorithm is based in two procedures
  - *Key Scheduling Algorithm* : initializes an array using the key
  - *Pseudo Random Generation Algorithm* : generates the pseudo-random bytes using the initial array

Security of RC4

- It is especially vulnerable
  - when the beginning of the output keystream is not discarded, ***or***
  - when nonrandom or related keys are used

- Based on these ideas, some communication protocols that used RC4 were broken
  - WEP and WPA-TKIP have efficient attacks
  - TLS should also be attackable

- IETF has published RFC 7465 (2015) to prohibit the use of RC4 in TLS
  - Google, Mozilla and Microsoft are stopping the support of RC4

Key Scheduling Algorithm (KSA)

KSA(K)
  Initialization:
    for i = 0 ... N-1
      S[i] = i
    j = 0
  Scrambling:
    for i = 0 ... N-1
      j = (j + S[i] + K[i mod len]) mod N
      Swap(S[i], S[j])

Initializes the array with an increasing number in each entry

S[]	0	1	2	3	4	5	6	.....

Exchanges the position of the stored values using the key

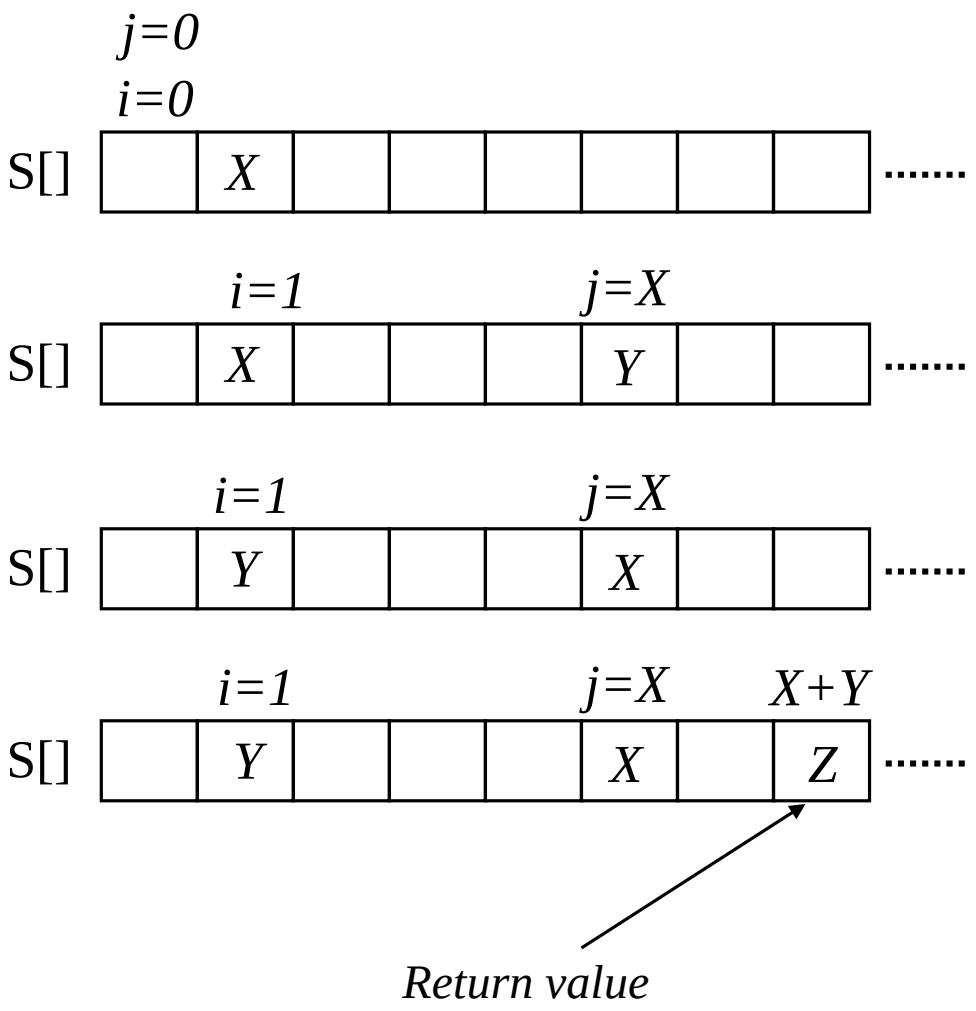
S[]	9	99	55	122	4	35	207	.....

where:

n – number of bits of a byte (8 bits = 1byte)
N – 2^n (256 entries)
K – key (40 to 256 bits = 5 to 32 bytes)
len – number of bytes of the key (5 to 32)
S[] – array with N entries (contains the state of the algorithm)
Swap(x, y) – exchange the values in entries x and y

Pseudo Random Generation Algorithm (PRGA)

PRGA(K)
  Initialization:
    i = 0
    j = 0
  Generation loop:
    while(true)
      i = (i + 1) mod N
      j = (j + S[i]) mod N
      Swap(S[i], S[j])
      return S[(S[i] + S[j]) mod N]



Spritz by Ron Rivest and Jacob Schuldt (2014)

- Change the PRGA with a new version
- **w** is *relatively prime* to the size of the S array
- after 256 iterations of the loop, the value *i* takes on all possible values 0 ... 255 and every byte in the S array has been swapped at least once

All arithmetic is performed modulo N, i.e., 256

```
PRGA(K)
  Initialization:
    i = 0
    j = 0
    k = 0
  Generation loop:
    while(true)
      i = i + w
      j = k + S[j + S[i]]
      k = k + i + S[j]
      Swap(S[i], S[j])
      return  z = S[j + S[i + S[z + k]]]
```

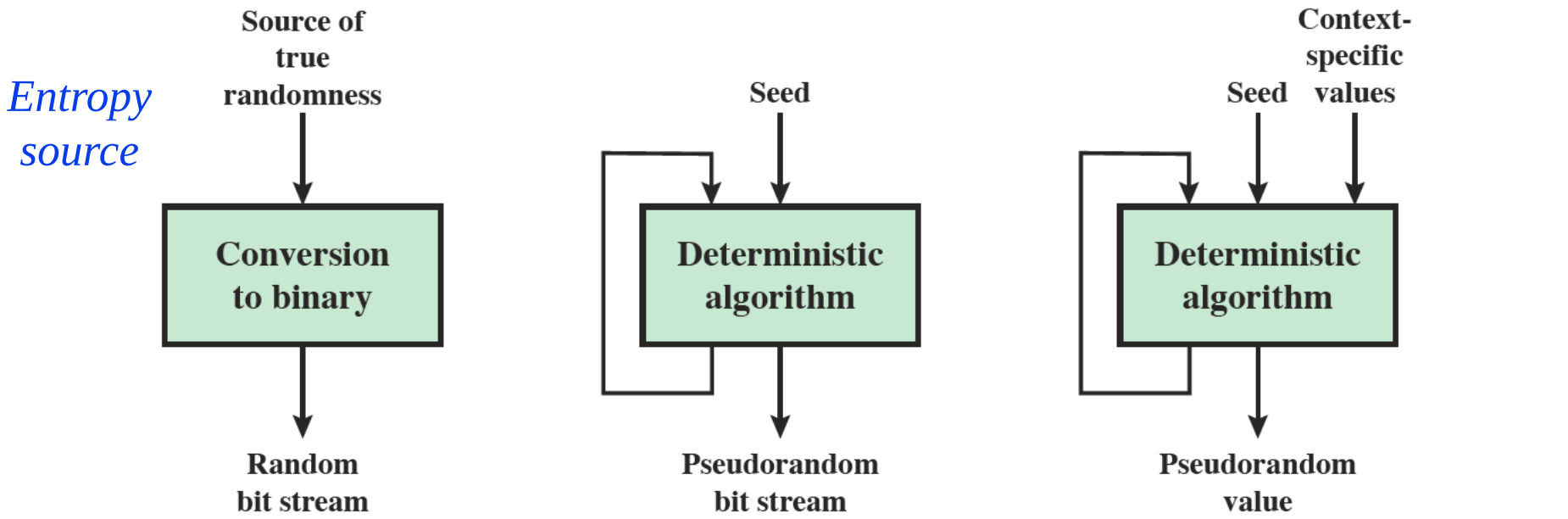
(PSEUDO) RANDOM NUMBERS

Pseudo-Random Generators (PRNG)

- PRNG are used in many applications
  - generation of *nonces* in authentication & key agreement protocols
  - creation of *session keys* and *RSA keys*
  - produce the *keystream* of symmetric stream cyphers

- The sequence of produced random numbers should fulfill some well-known statistical properties
  - *Uniform distribution* : the frequency of 0 and 1 in the sequence is similar
  - *Independence* : a subsequence of the sequence can not be inferred from the other subsequences
- In security, it makes sense to consider an extra property
  - *Unpredictability* : the successive elements of the sequence that are produced are unpredictable
- Although there several tests to assess these properties, **there is none** that can "prove" for instance the independence

Types random generators

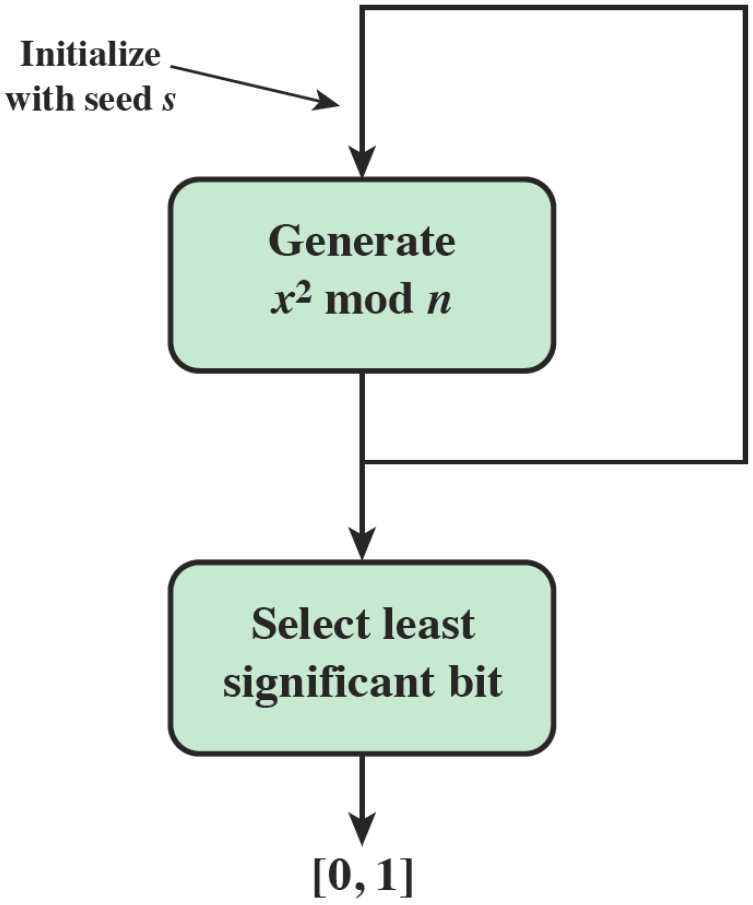


- Entropy source is typically taken from the physical environment (e.g., keystroke timing patterns, disk electrical activity)
- Seed is a fixed input value that often is obtained from a TRNG
- PRNG and PRF differ in: the first produces bits, and the second a string with a fixed number of bits and uses some context specific values (e.g., user ID)

Blum Blum Shub Pseudorandom Generator

- Choose two large prime numbers p and q, such that they have remainder 3 when divided by 4
- Calculate : n = p * q
- Choose a random number s relatively prime to n

$X_0 = s^2 \bmod n$
for $i=1$ *to* ∞
$X_i = (X_{i-1})^2 \bmod n$
select bit: $B_i = X_i \bmod 2$

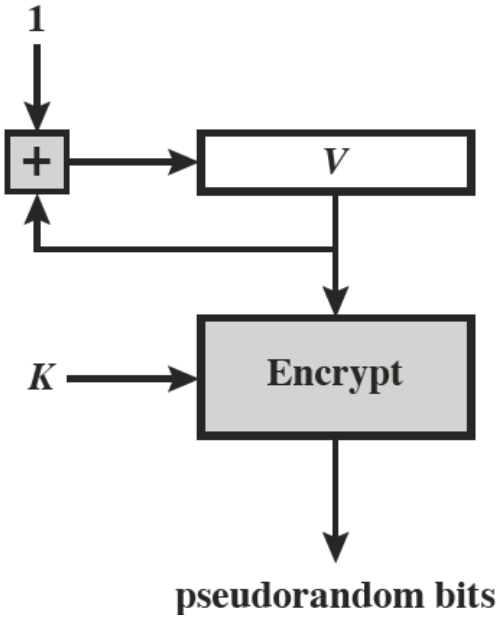


One of the most used generators, with very strong guarantees of cryptographic strength

Pseudorandom Generation with a Block Cipher

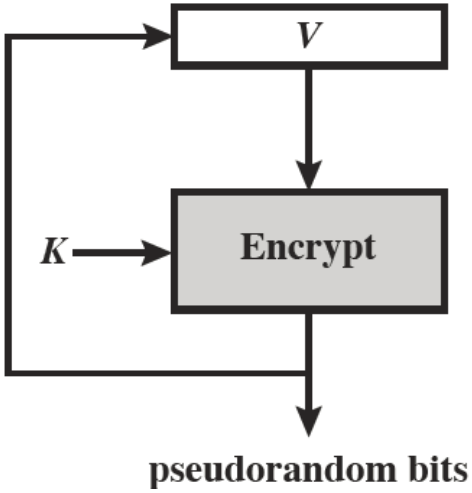
- Two example approaches are based on
  - Counter mode: NIST SP 800-90, ANSI standard X9.82 and RFC 4086
  - Output-feedback mode: ANSI standard X9.82 and RFC 4086

Counter mode



K and V come from the seed

Output-feedback mode



Bibliography

- Stallings, 6 edition : 48-53, 82-92, (92-100 for DES), 195-200, 211-213, 239-243 (stream ciphers/RC4), 223-239 and 243-247

- Stallings, 5 edition : 56-62, 91-101, (101-120 for DES), 216-222, 256-261 (stream ciphers/RC4)
- Stallings, 4 edition : 29-35, 63-72, (72-89 for DES), 175-180, 189-194 (for RC4)