CSCI361
Computer Security

Modes for block ciphers

Outline

- Modes.
  - Electronic codebook.
  - Cipher block chaining.
  - Cipher feedback.
  - Output feedback.
  - Counter.
- Advantages/Disadvantages.
- Padding.

Modes of operation for block ciphers

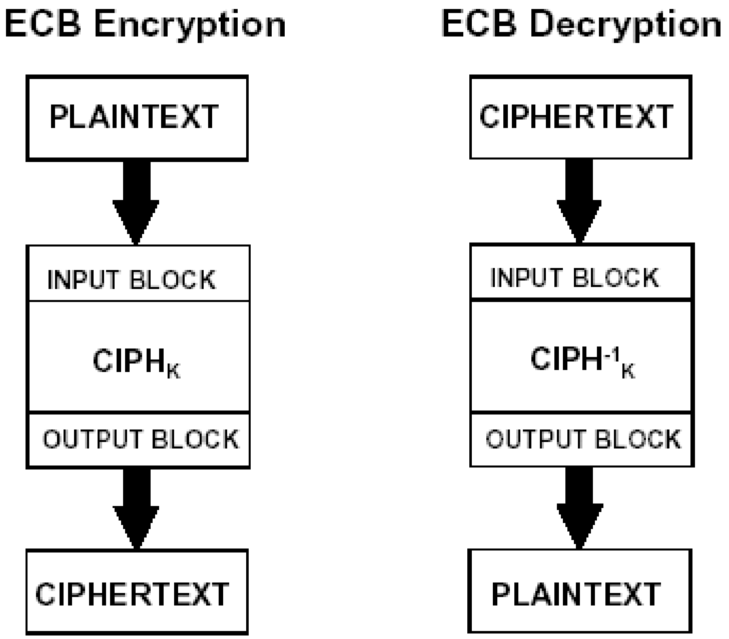
- A block cipher is defined to work on blocks of a particular size. A message will generally consist of multiple blocks. **Modes** describe the relationship between the application of the block cipher to different blocks.
- A block cipher can be used in different **modes**. The following four modes are standard modes which were recommended for DES.

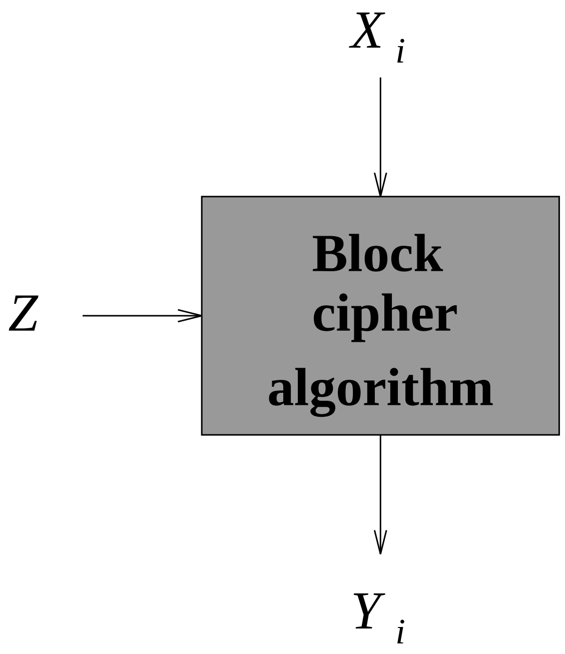
1. Electronic codebook mode (ECB).
2. Cipher block chaining mode (CBC).
3. Cipher feedback mode (CFB).
4. Output feedback mode (OFB).

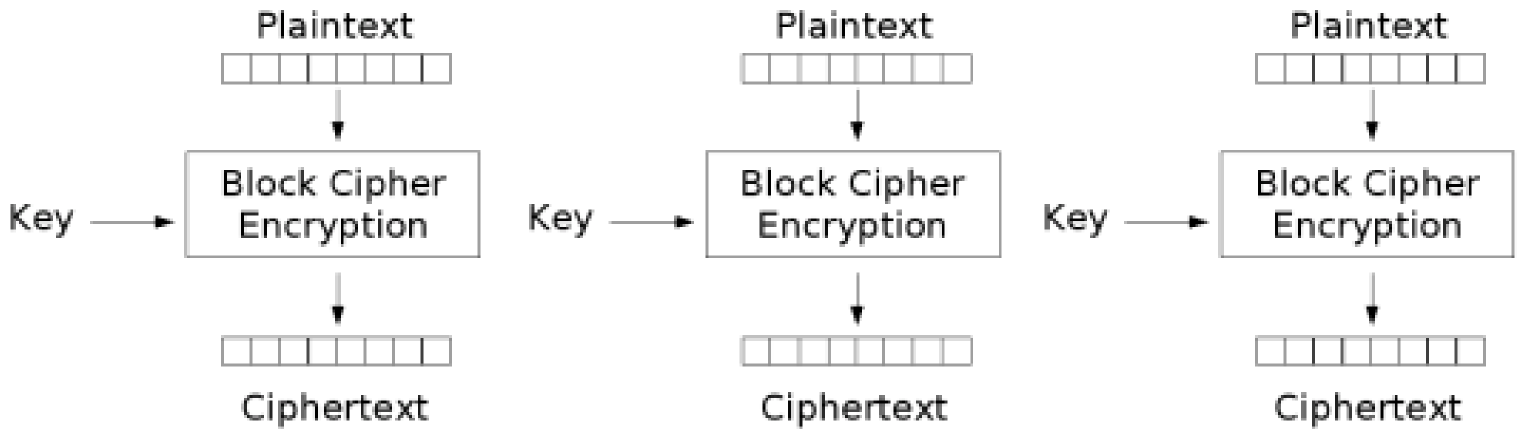
Although the modes can have different security properties, the length of the key remains the same, and therefore the upper bound on security stays the same also.

Electronic codebook mode (ECB)

- This is the basic mode of operation. A plaintext message is broken into blocks and each block is encrypted separately.

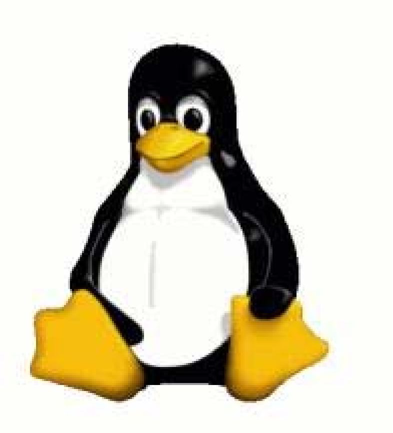


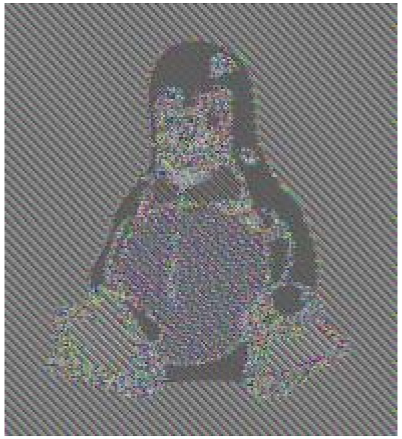




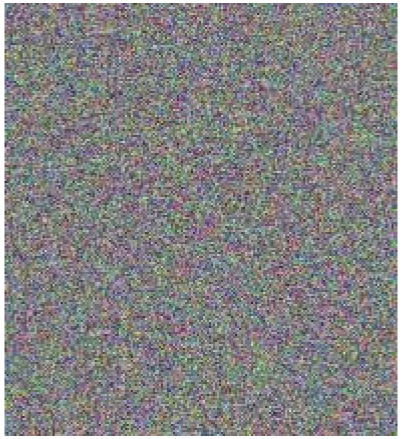
Electronic Codebook (ECB) mode encryption

- Electronic codebook mode has the **disadvantage** that if we encrypt two identical plaintext blocks, the resulting ciphertext blocks will be the same.
- This leaks some information to an eavesdropper.
- Formatting information, such as spaces between columns or paragraphs, and other plaintext details can also leak through the cipher system.





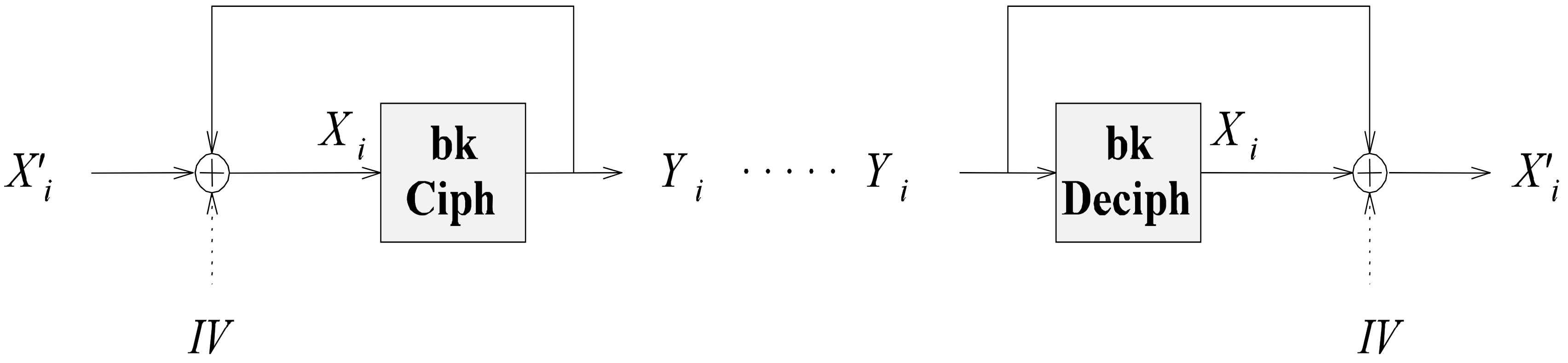
ECB



Other

http://en.wikipedia.org/wiki/Block_cipher_modes_of_operation

Cipher block chaining mode (CBC)



- In this mode a plaintext block is XORed with the ciphertext block of the previous round and then entered to the encryption algorithm.
- A strong dependency between consecutive blocks is created.

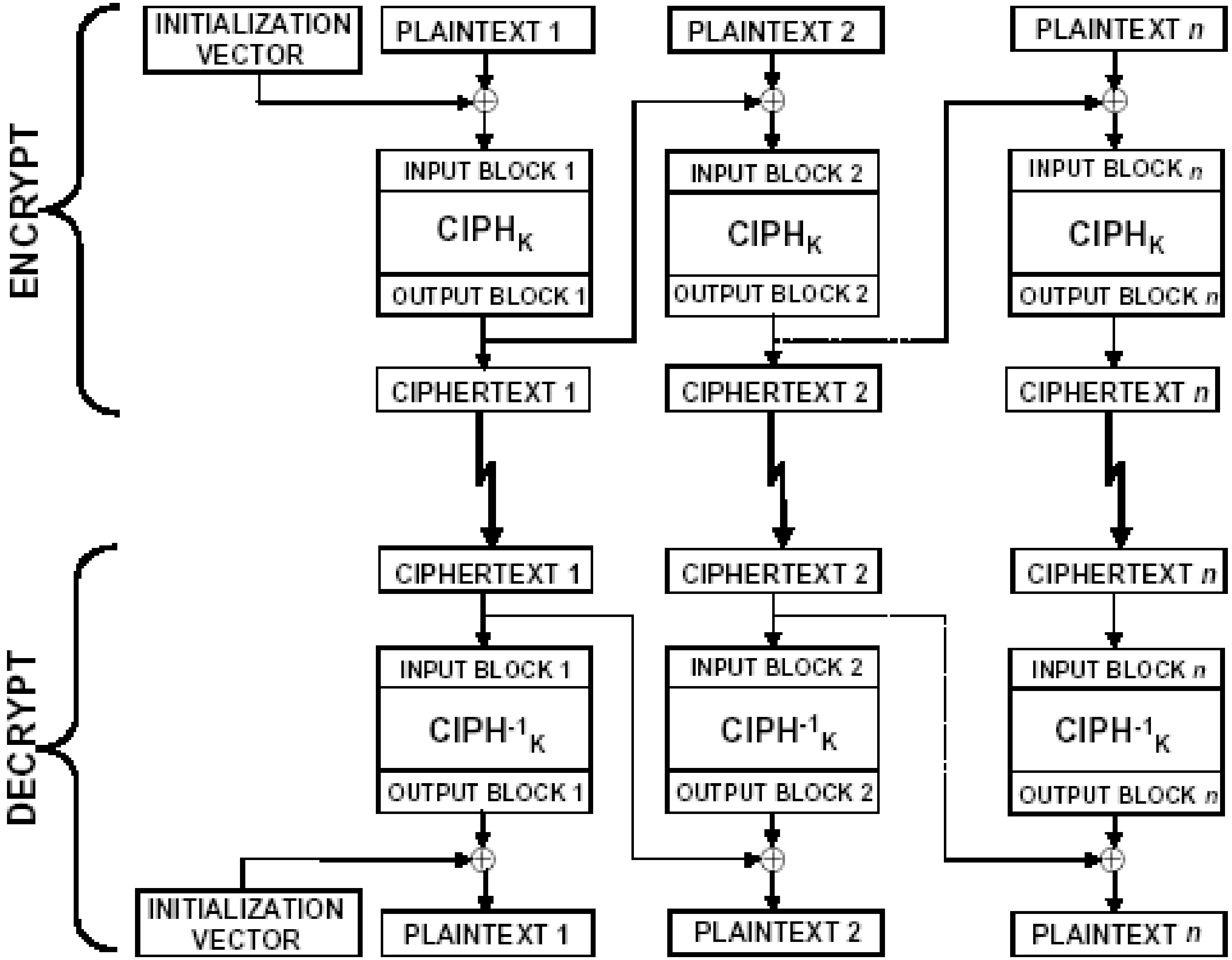
$$X_1 = X_1' \oplus IV$$

$$X_2 = X_2' \oplus Y_1$$

$$\ldots$$

$$X_i = X_i' \oplus Y_{i-1}$$

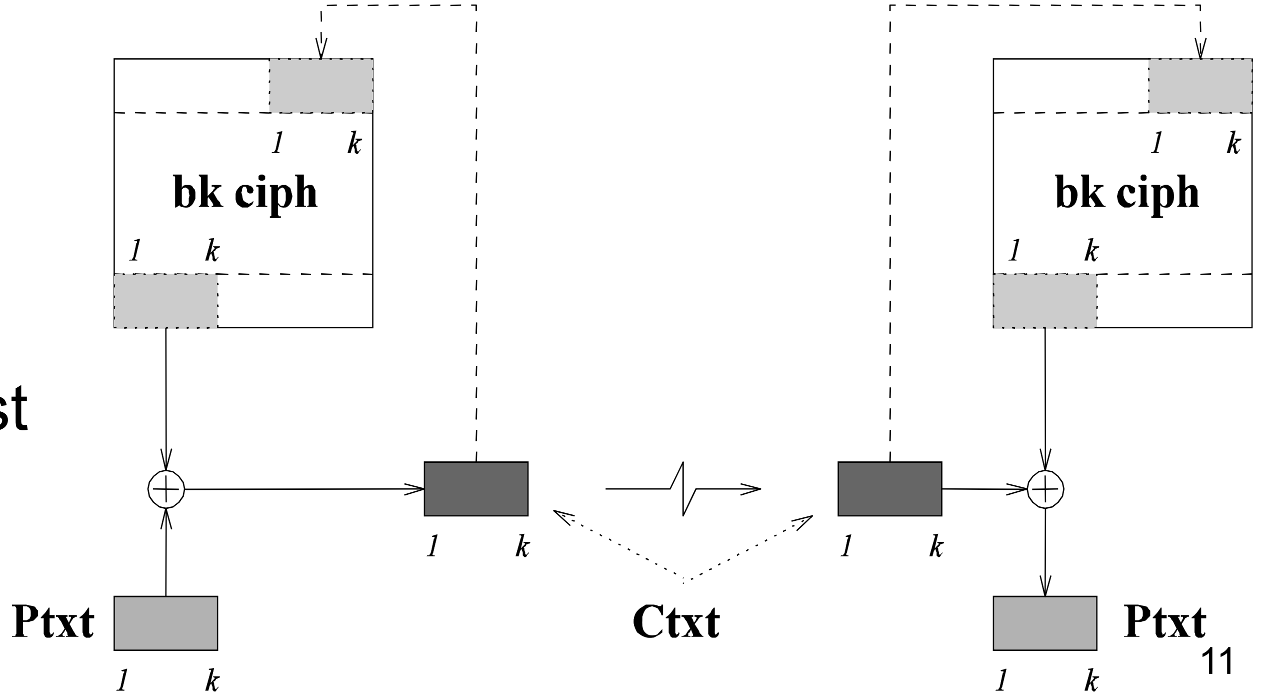
- For the first block, since ciphertext from the previous block does not exist, a random initial vector, called the IV, is used. IV is not secret but it needs to be unpredictable.
- CBC mode produces different ciphertext blocks even if plaintext blocks are the same.
- The strong dependency between blocks suggests that CBC can also be used to provide integrity.

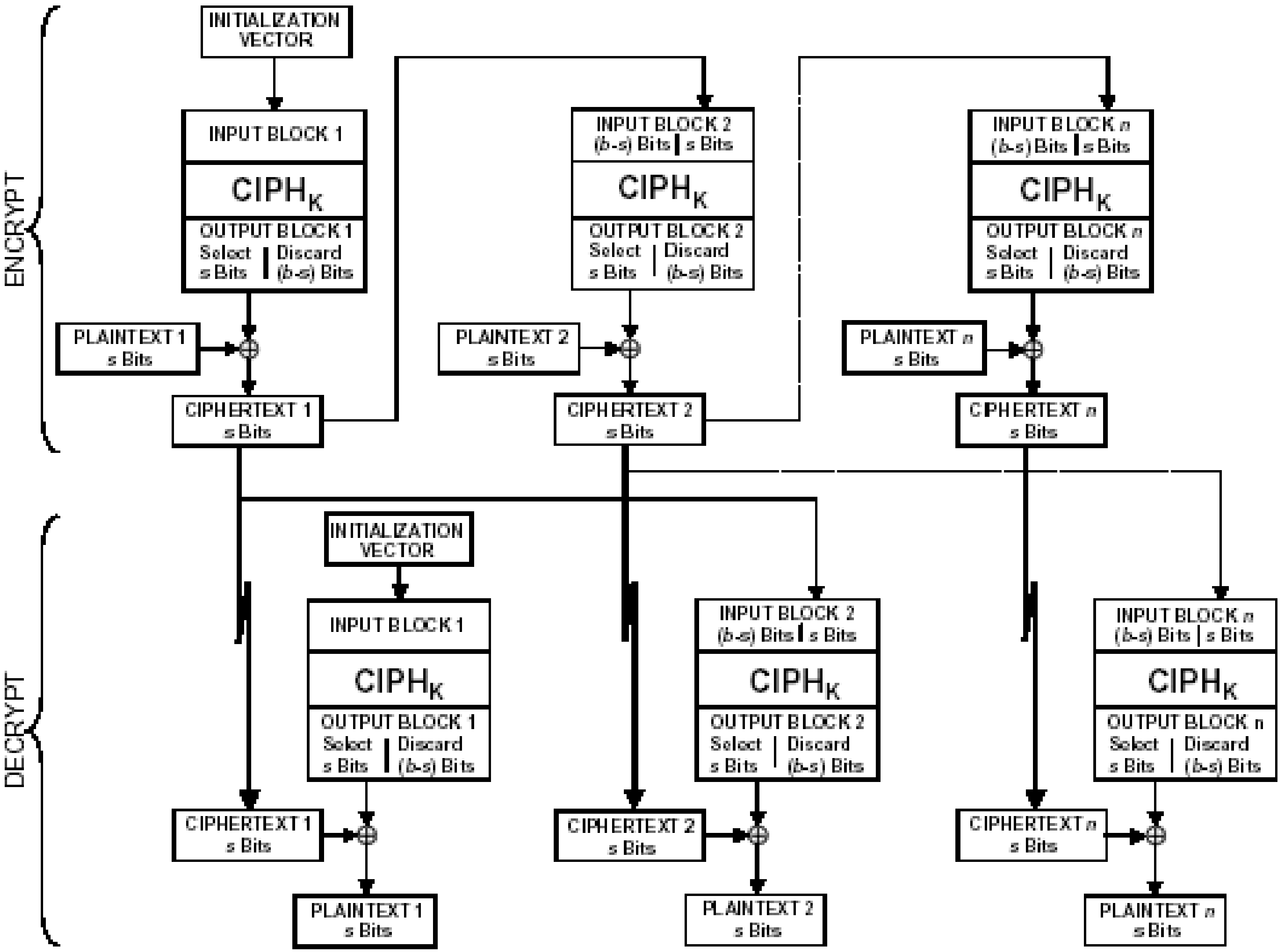

ENCRYPT
INITIALIZATION VECTOR
PLAINTEXT 1
PLAINTEXT 2
PLAINTEXT n
INPUT BLOCK 1
INPUT BLOCK 2
INPUT BLOCK n
CIPH_K
CIPH_K
CIPH_K
OUTPUT BLOCK 1
OUTPUT BLOCK 2
OUTPUT BLOCK n
CIPHERTEXT 1
CIPHERTEXT 2
CIPHERTEXT n
DECRYPT
CIPHERTEXT 1
CIPHERTEXT 2
CIPHERTEXT n
INPUT BLOCK 1
INPUT BLOCK 2
INPUT BLOCK n
CIPH-1_K
CIPH-1_K
CIPH-1_K
OUTPUT BLOCK 1
OUTPUT BLOCK 2
OUTPUT BLOCK n
INITIALIZATION VECTOR
PLAINTEXT 1
PLAINTEXT 2
PLAINTEXT n

(k-bit) Cipher feedback mode (CFB)

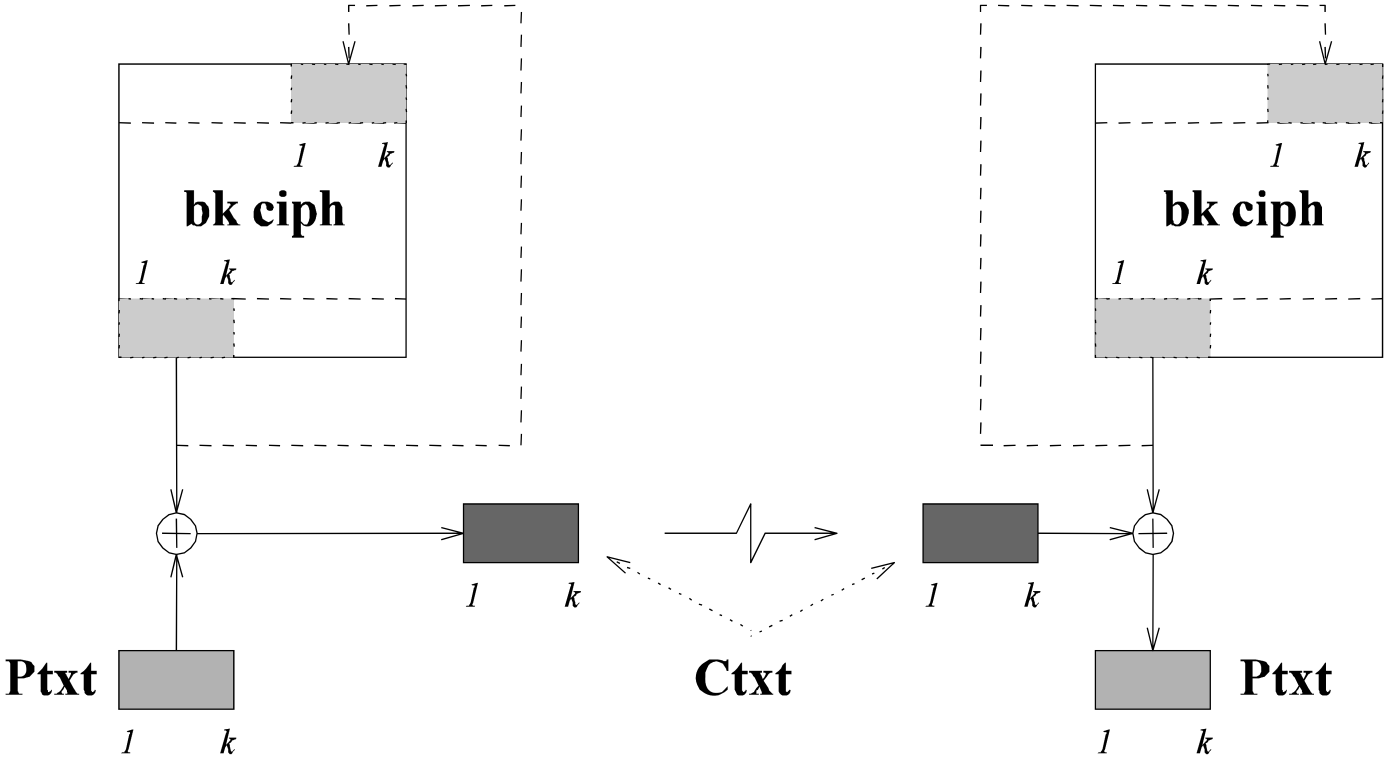
- The block cipher algorithm is effectively turned into a pseudorandom generator that produces a *k*-bit pseudorandom number in every execution of the algorithm.
- For decryption a similar generator is used to remove the masking pseudo-noise data.
- The input to the cipher “at the top” is the obtained by concatenating the input to the previous round, left shifted by *k*-bits, with the *k*-bit feedback.

Using smaller k in general produces a **more secure** pseudorandom stream, with the cost of lowering speed.



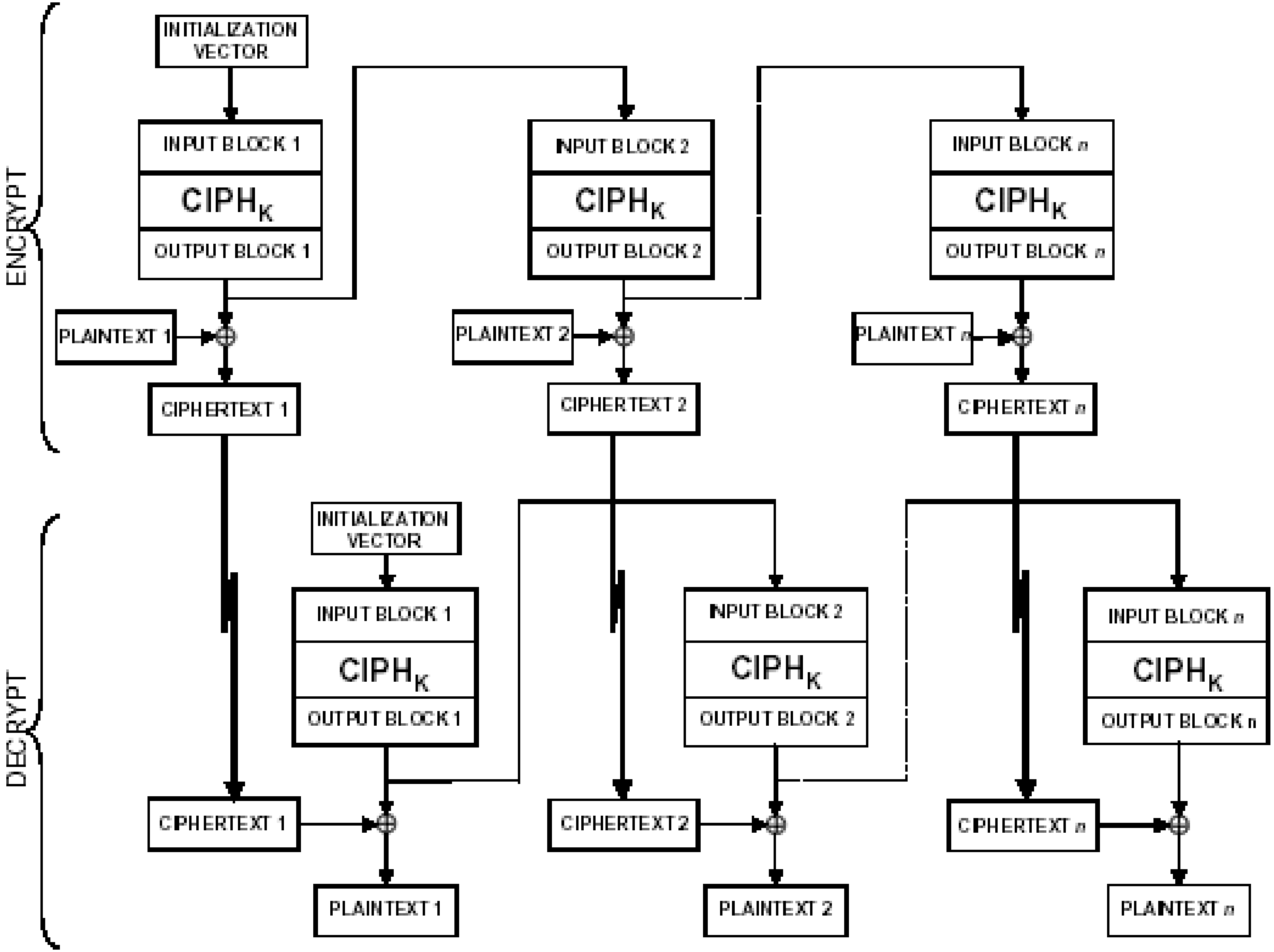

ENCRYPT
INITIALIZATION VECTOR
INPUT BLOCK 1
CIPH$_K$
OUTPUT BLOCK 1
Select s Bits
Discard (b-s) Bits
PLAINTEXT 1 s Bits
CIPHERTEXT 1 s Bits
INPUT BLOCK 2
(b-s) Bits | s Bits
CIPH$_K$
OUTPUT BLOCK 2
Select s Bits
Discard (b-s) Bits
PLAINTEXT 2 s Bits
CIPHERTEXT 2 s Bits
INPUT BLOCK n
(b-s) Bits | s Bits
CIPH$_K$
OUTPUT BLOCK n
Select s Bits
Discard (b-s) Bits
PLAINTEXT n s Bits
CIPHERTEXT n s Bits
DECRYPT
INITIALIZATION VECTOR
INPUT BLOCK 1
CIPH$_K$
OUTPUT BLOCK 1
Select s Bits
Discard (b-s) Bits
CIPHERTEXT 1 s Bits
PLAINTEXT 1 s Bits
INPUT BLOCK 2
(b-s) Bits | s Bits
CIPH$_K$
OUTPUT BLOCK 2
Select s Bits
Discard (b-s) Bits
CIPHERTEXT 2 s Bits
PLAINTEXT 2 s Bits
INPUT BLOCK n
(b-s) Bits | s Bits
CIPH$_K$
OUTPUT BLOCK n
Select s Bits
Discard (b-s) Bits
CIPHERTEXT n s Bits
PLAINTEXT n s Bits

(k-bit) Output feedback mode (OFB)



- This is very similar to CFB, but the feedback is independent of the transmitted data.
- This means the pseudorandom stream can be pre-generated prior to any transmission.

A special case of OFB: k=block size.



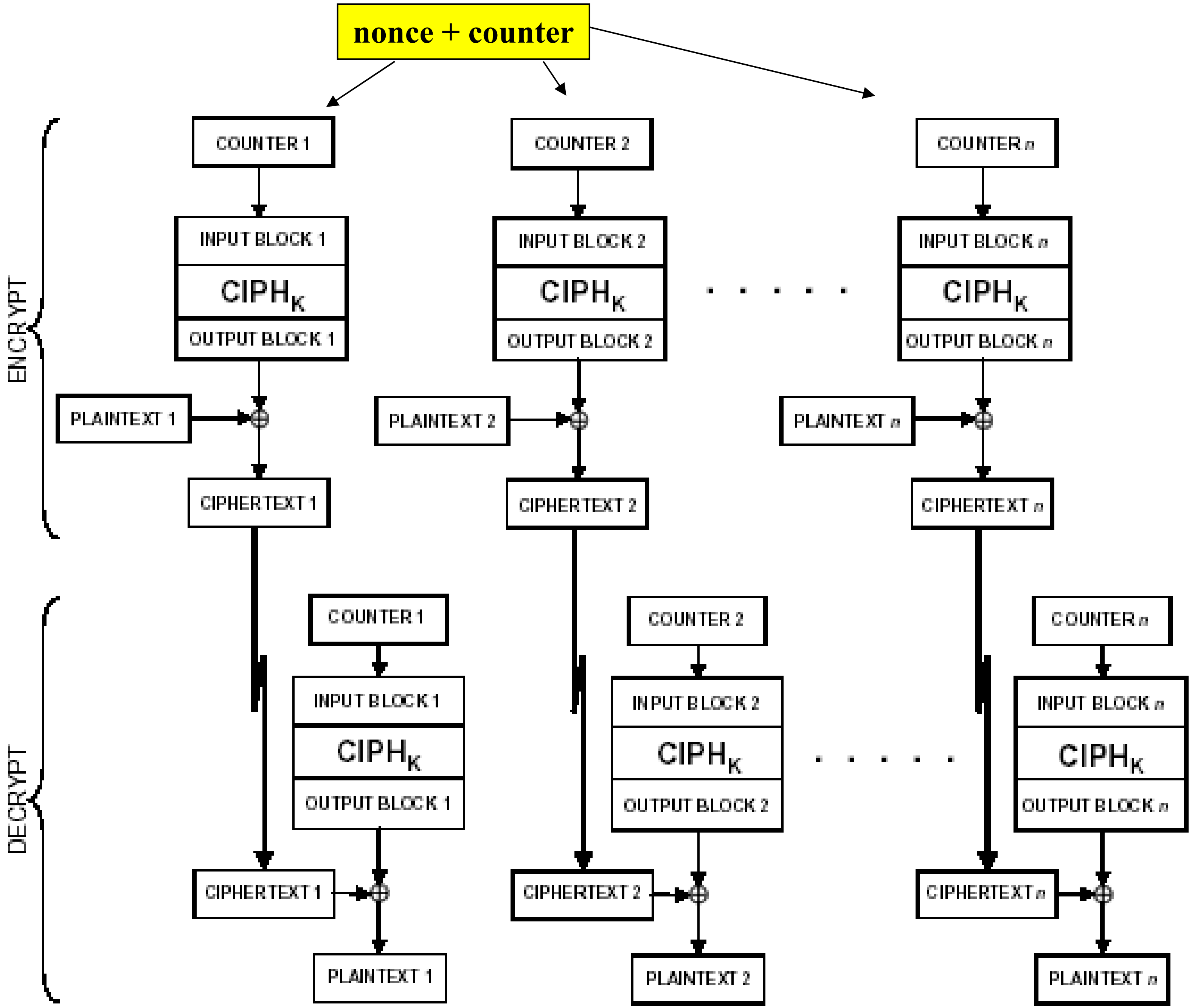
Advantages of CBC, CFB and OFB

1. All hide patterns in plaintext since $X_{i+n}=X_i$ does not imply that $Y_{i+n}=Y_i$.
   - Two identical plaintext blocks at two different locations in a ciphertext will produce two distinct ciphertexts. This will stop the leakage of information which occurs in ECB mode.
2. Prevent data tampering (CFB).
   - The dependency between ciphertext blocks can be used to detect tampering of the ciphertext. That is, a single bit change in a ciphertext block will affect decryption of all the following ciphertext blocks.
3. The k-bit ciphers (CFB and OFB) can be tuned, with respect to k, to allow for the user's data format, required level of security and resources.
4. The OFB cipher can be performed in parallel, with pre-computation of the pseudorandom stream.

Counter Mode (CTR)

- The cipher is applied to a set of input blocks, called **counters**.
- The sequence of output blocks are XORed with the plaintext to produce the ciphertext.
- The sequence of counters must satisfy the following property:

Each block in the sequence is different from every other block, across all of the messages that are encrypted under the given key.


nonce + counter
ENCRYPT
COUNTER 1
INPUT BLOCK 1
CIPH_K
OUTPUT BLOCK 1
PLAINTEXT 1
CIPHERTEXT 1
COUNTER 2
INPUT BLOCK 2
CIPH_K
OUTPUT BLOCK 2
PLAINTEXT 2
CIPHERTEXT 2
COUNTER n
INPUT BLOCK n
CIPH_K
OUTPUT BLOCK n
PLAINTEXT n
CIPHERTEXT n
DECRYPT
COUNTER 1
INPUT BLOCK 1
CIPH_K
OUTPUT BLOCK 1
CIPHERTEXT 1
PLAINTEXT 1
COUNTER 2
INPUT BLOCK 2
CIPH_K
OUTPUT BLOCK 2
CIPHERTEXT 2
PLAINTEXT 2
COUNTER n
INPUT BLOCK n
CIPH_K
OUTPUT BLOCK n
CIPHERTEXT n
PLAINTEXT n

- In both CTR encryption and decryption the ciphers can be performed in parallel.
  - The plaintext block that corresponds to any particular ciphertext block can be recovered independently from the other plaintext blocks.
  - The ciphers can be applied to the counters prior to the availability of the plaintext or ciphertext data.

Mode examples: ECB

- Let us consider a 3-bit block cipher with the following mapping:

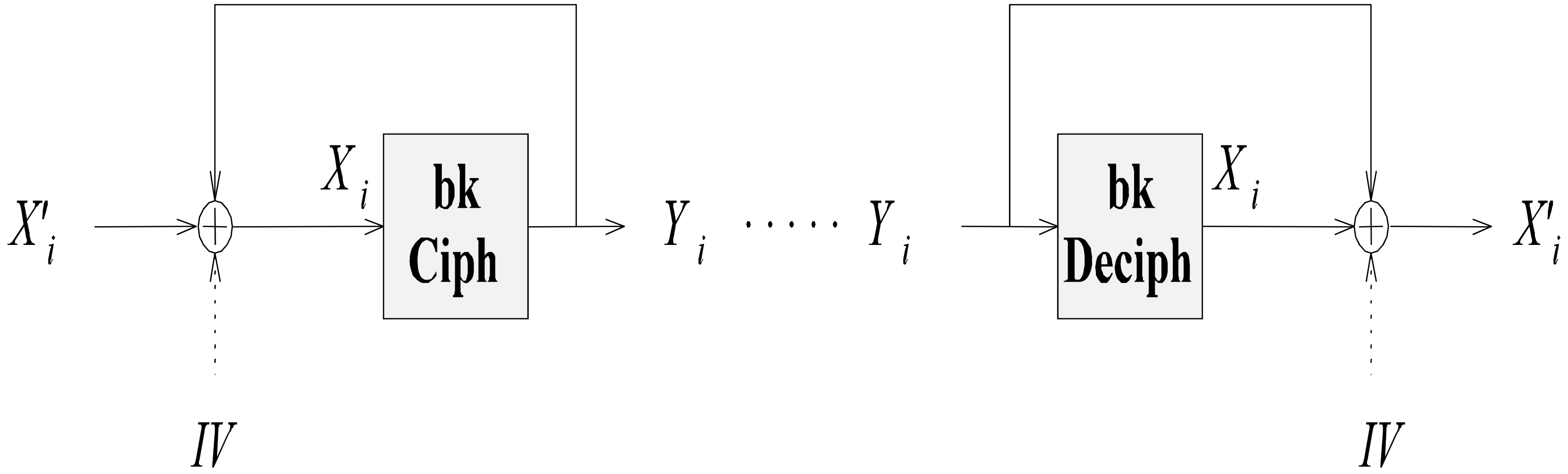
Input	000	001	010	011	100	101	110	111
Output	111	110	011	100	001	000	101	010

ECB

Plaintext	101	101	110	010
Ciphertext	000	000	101	011

Input	000	001	010	011	100	101	110	111
Output	111	110	011	100	001	000	101	010

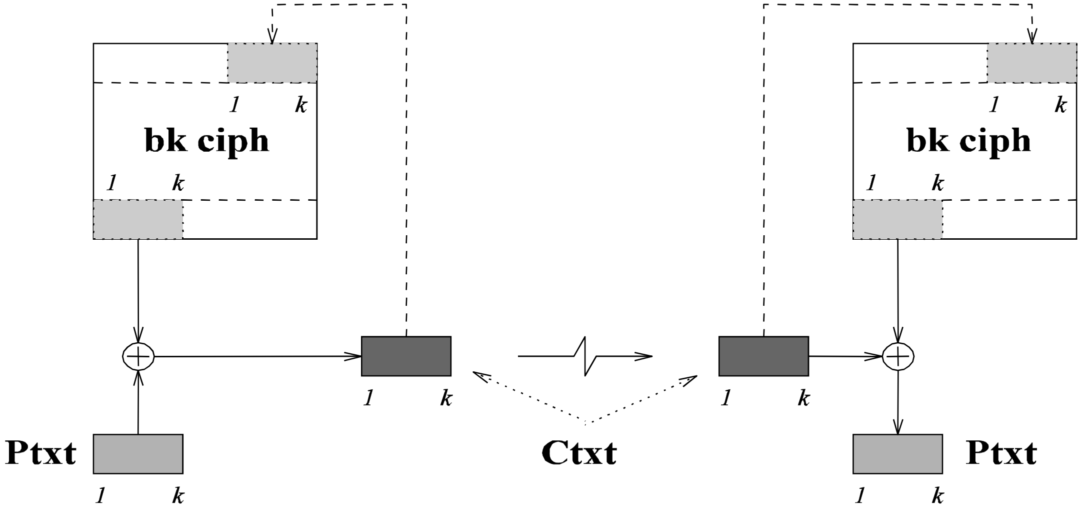
$\text{XOR}_0 = \text{Plaintext}_0 \oplus \text{IV}$ $\text{XOR}_i = \text{Plaintext}_i \oplus \text{Ciphertext}_{i-1}$



CBC
IV=111

Plaintext	101	101	110	010
XOR	010	110	011	110
Ciphertext	011	101	100	101

Input	000	001	010	011	100	101	110	111
Output	111	110	011	100	001	000	101	010

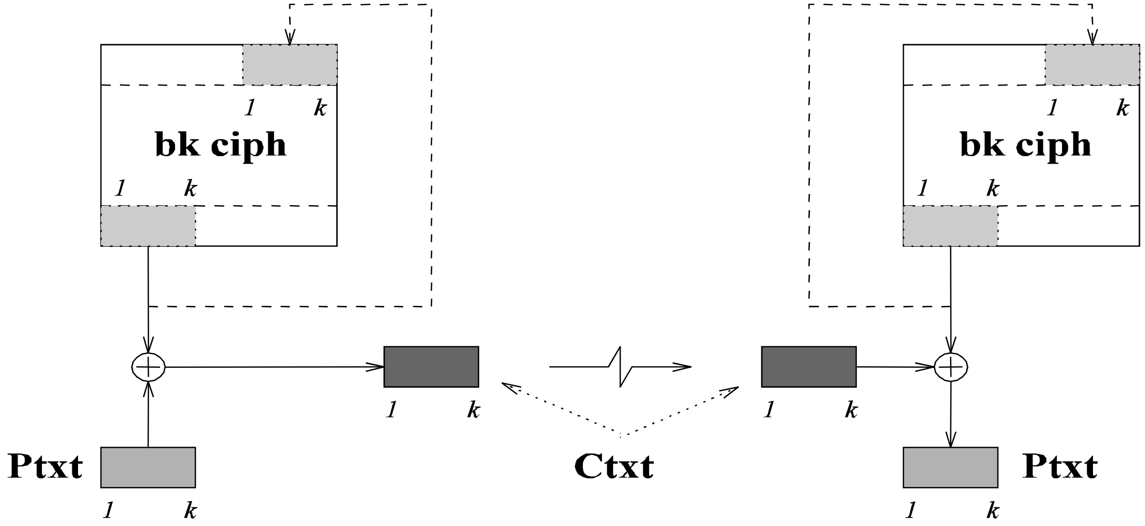


2-bit CFB
IV=111

Plaintext	10	11	01	11	00	10
Input	111	111	110	011	101	100
E(Input)	**01**0	**01**0	**10**1	**10**0	**00**0	**00**1
Ciphertext	11	10	11	01	00	10

Input	000	001	010	011	100	101	110	111
Output	111	110	011	100	001	000	101	010

2-bit OFB
IV=111



Plaintext	10	11	01	11	00	10
Input	111	101	100	000	011	110
E(Input)	**010**	**000**	**001**	**111**	**100**	**101**
Ciphertext	11	11	01	00	10	00

Input	000	001	010	011	100	101	110	111
Output	111	110	011	100	001	000	101	010

CTR
IV=000

$$\mathbf{Counter_i = Counter_{i-1} + 1}$$

Plaintext	101	101	110	010
IV	000	001	010	011
E(IV)	111	110	011	100
Ciphertext	010	011	101	110

+1 +1 +1

enc enc enc enc

XOR

Padding

- We have discussed block ciphers and described modes which tell us how to deal with multiple blocks.
- One issue we have not considered so far is that the last block of plaintext is, in general, a partial block of size *u* bits, this being less than the blocksize of the cipher.
- One needs to apply a **padding rule** to allow the last block to be suitably encrypted.
- That is, the block ciphers and modes mostly require full blocks.

- Padding example: Add a string 1000…0 to fill out the last block to the correct length. For decryption take the plaintext as being read back to the least significant 1.
  - Note that there are two possible plaintext values, that back to the least significant 1 or that with the full block (if padding hasn’t been added).
- One could use an arbitrary method, essentially adding random data with a length for the last block included.
- For CTR mode one can simply use only the most significant u bits of the last output block for the XOR operation. This reveals the length of the message though, which is not always desirable.

CSCI361

Computer Security

Other block ciphers

Outline

- FEAL
- LOKI91
- Blowfish
- IDEA 1990
- TEA
- RC5

FEAL

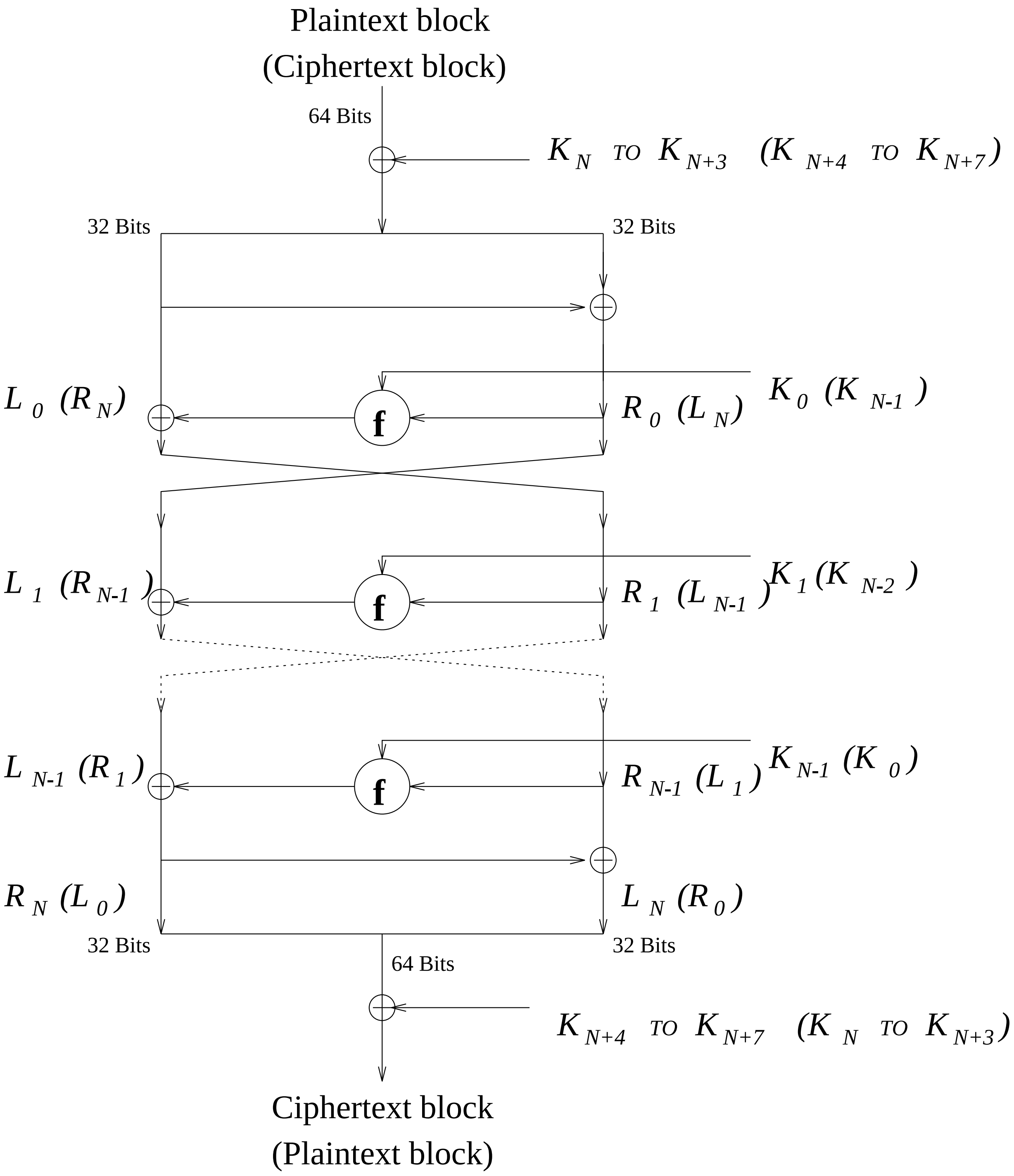
- The **F**ast **D**ata **E**ncryption **AL**gorithm (FEAL) was developed for high speed software implementation, at Nippon Telegraph and Telephone (NTT).
- The same algorithm is used for decryption, but the sub-keys are used in reverse order.
- FEAL-4 is a 64 bits block cipher with 64 bits key, proposed in 1987 (Miaguchi). It was successfully cryptanalysed with a chosen-plaintext attack in 1988 (den Boer).
- FEAL-8 was successfully cryptanalysed in 1989 (Biham-Shamir).
- FEAL-N/NX was developed in 1990.

- Differential-linear cryptanalysis can break FEAL-8 with only **12 chosen plaintext blocks**.

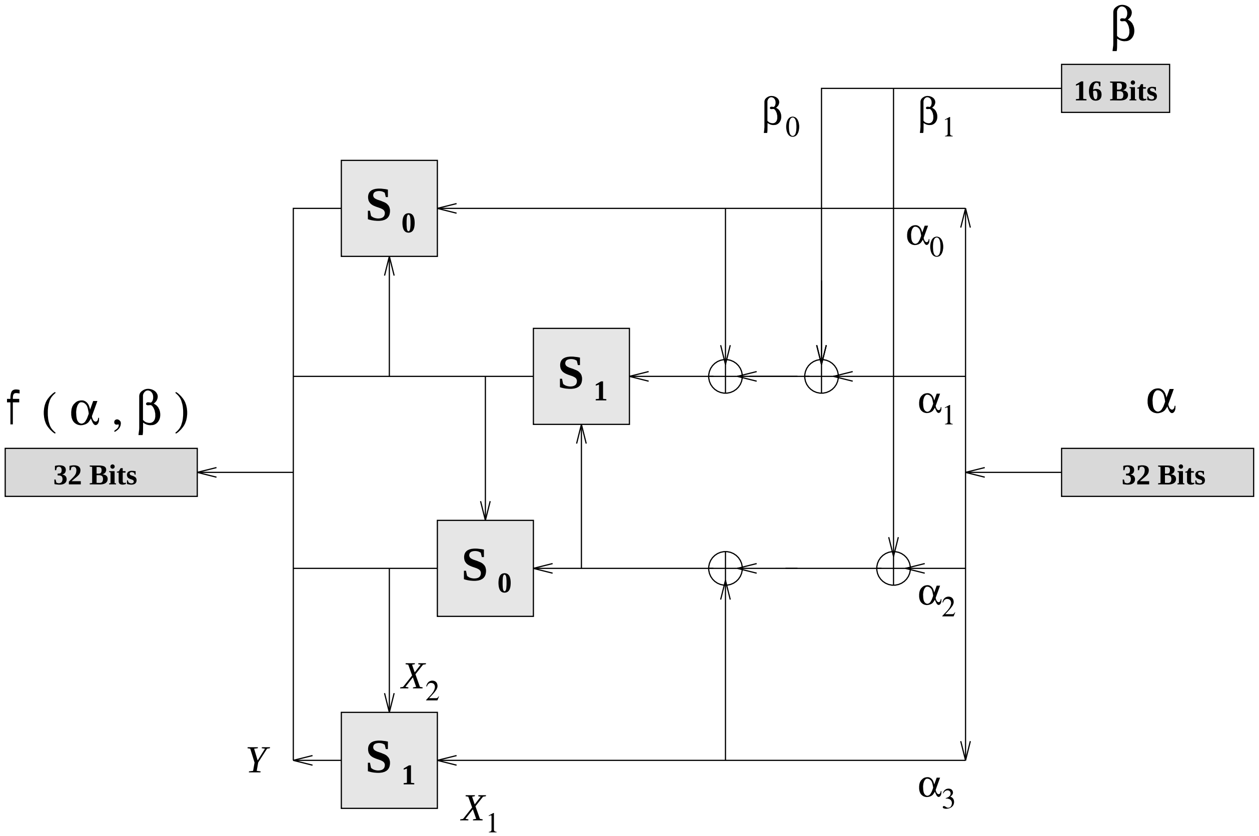
- FEAL-N has a variable number of rounds. Biham-Shamir showed that for N<32 the algorithm can be broken more quickly than 2_{64} chosen plaintext encryptions.

- FEAL-NX is a modification of FEAL that takes a 128 bit key. Biham-Shamir showed that the increase in key length does not add much to the security.

FEAL Encryption (Decryption)



FEAL f function



The two S boxes act as follows:

$Y = S_0(X1, X2) = \text{Rot2}((X1+X2) \quad \text{mod } 256$

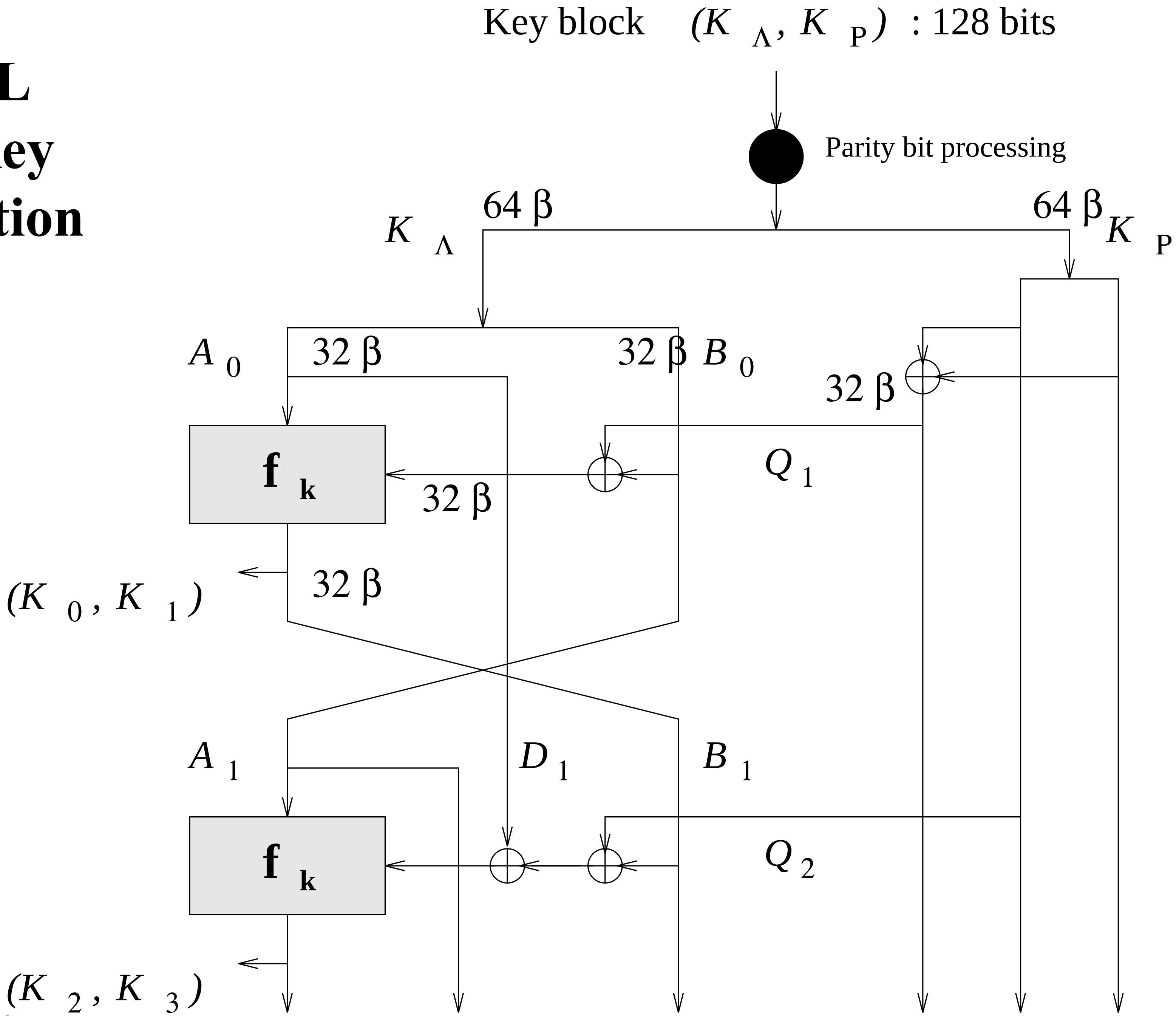
$Y = S_1(X1, X2) = \text{Rot2}((X1+X2+1) \text{ mod } 256$

X1 and X2 are 8 bit inputs.

Y is an 8 bit output.

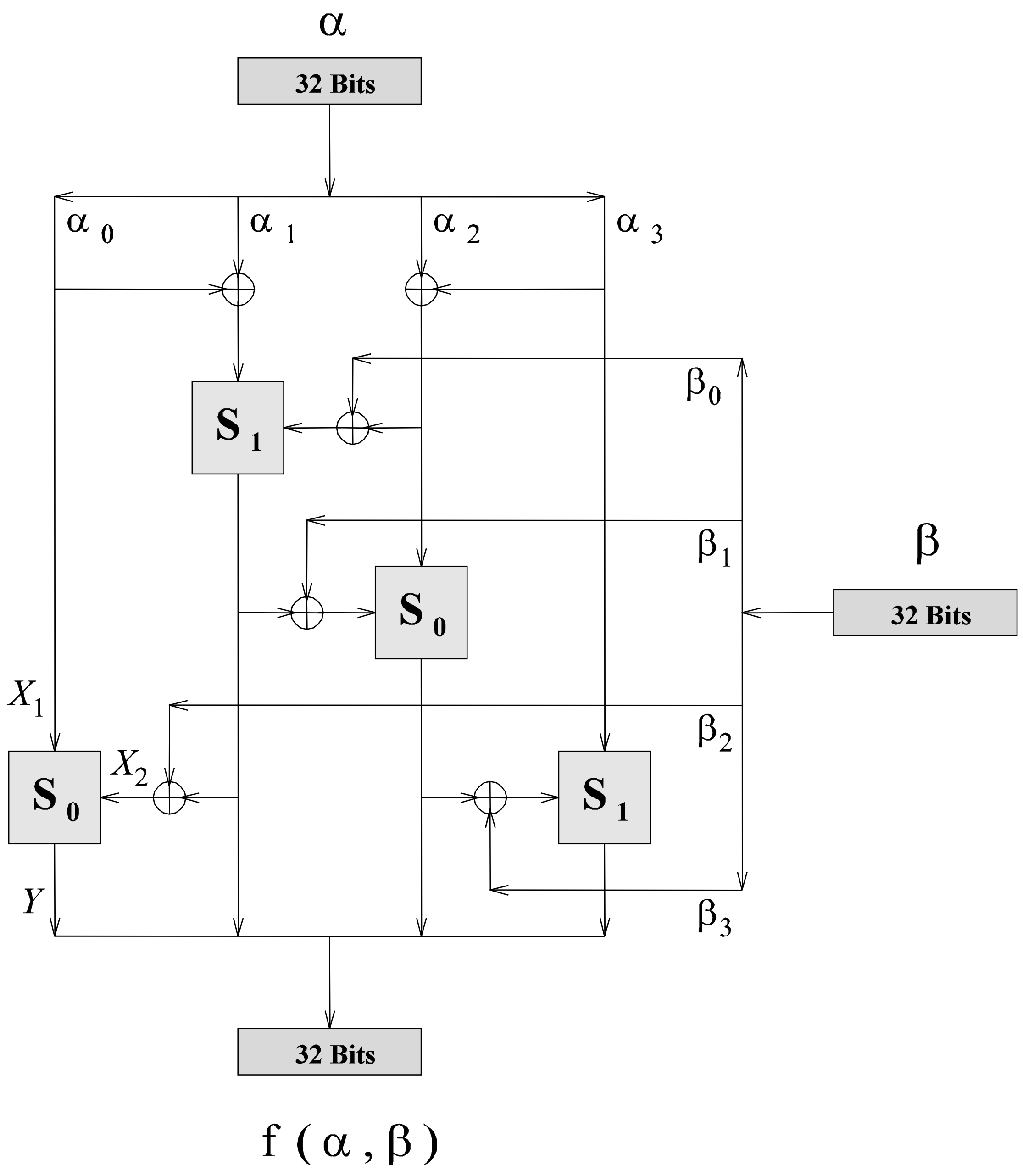
Rot2(X) is a 2-bit left rotation on 8-bit data X.

FEAL Sub-key generation



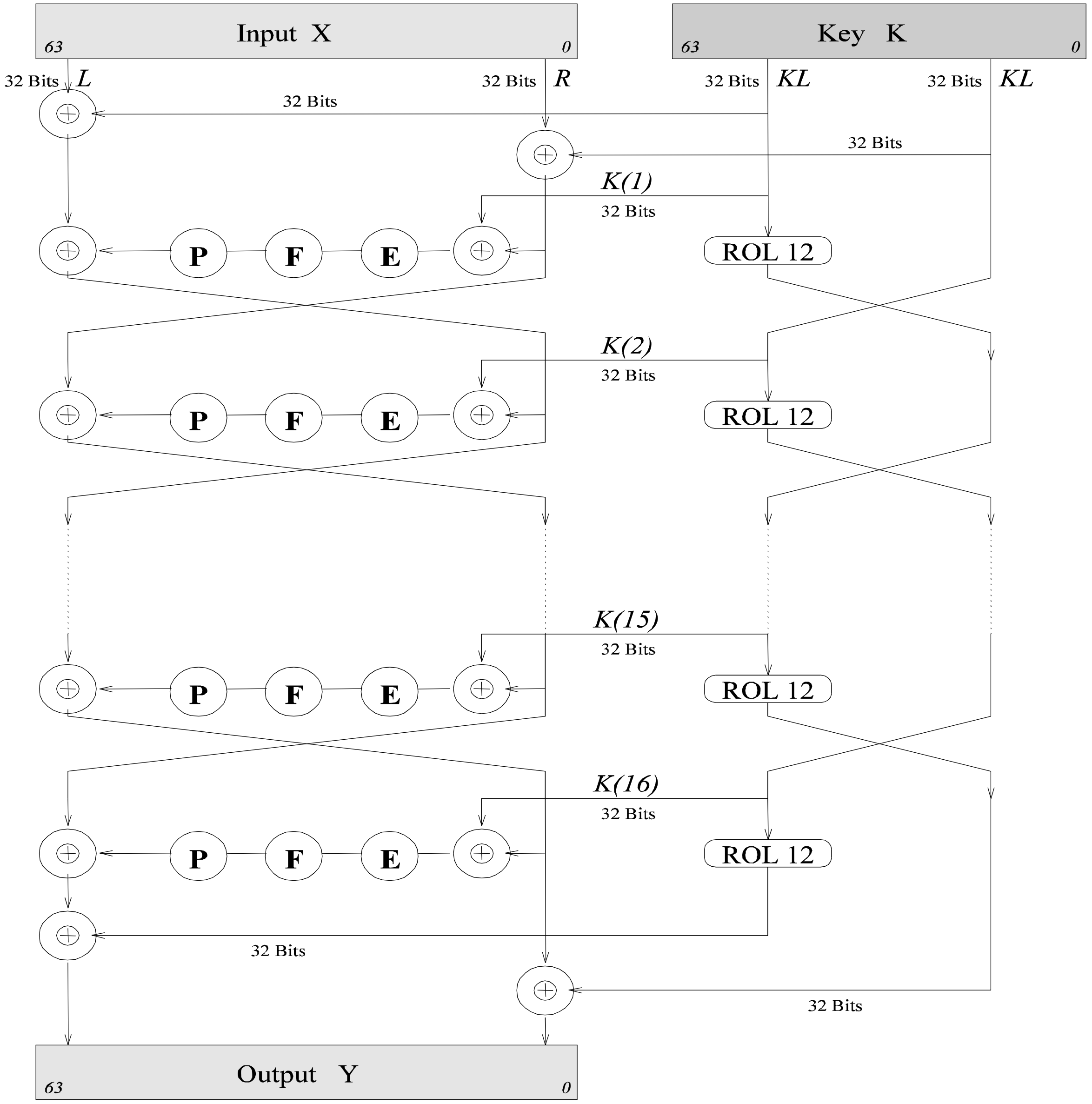
FEAL $\mathbf{f_k}$

- The S boxes, S0 and S1, are defined in the same way as for the FEAL **f** function.
- Note that **f** and $\mathbf{f_k}$ are necessarily different, since the key and data input sizes differ.



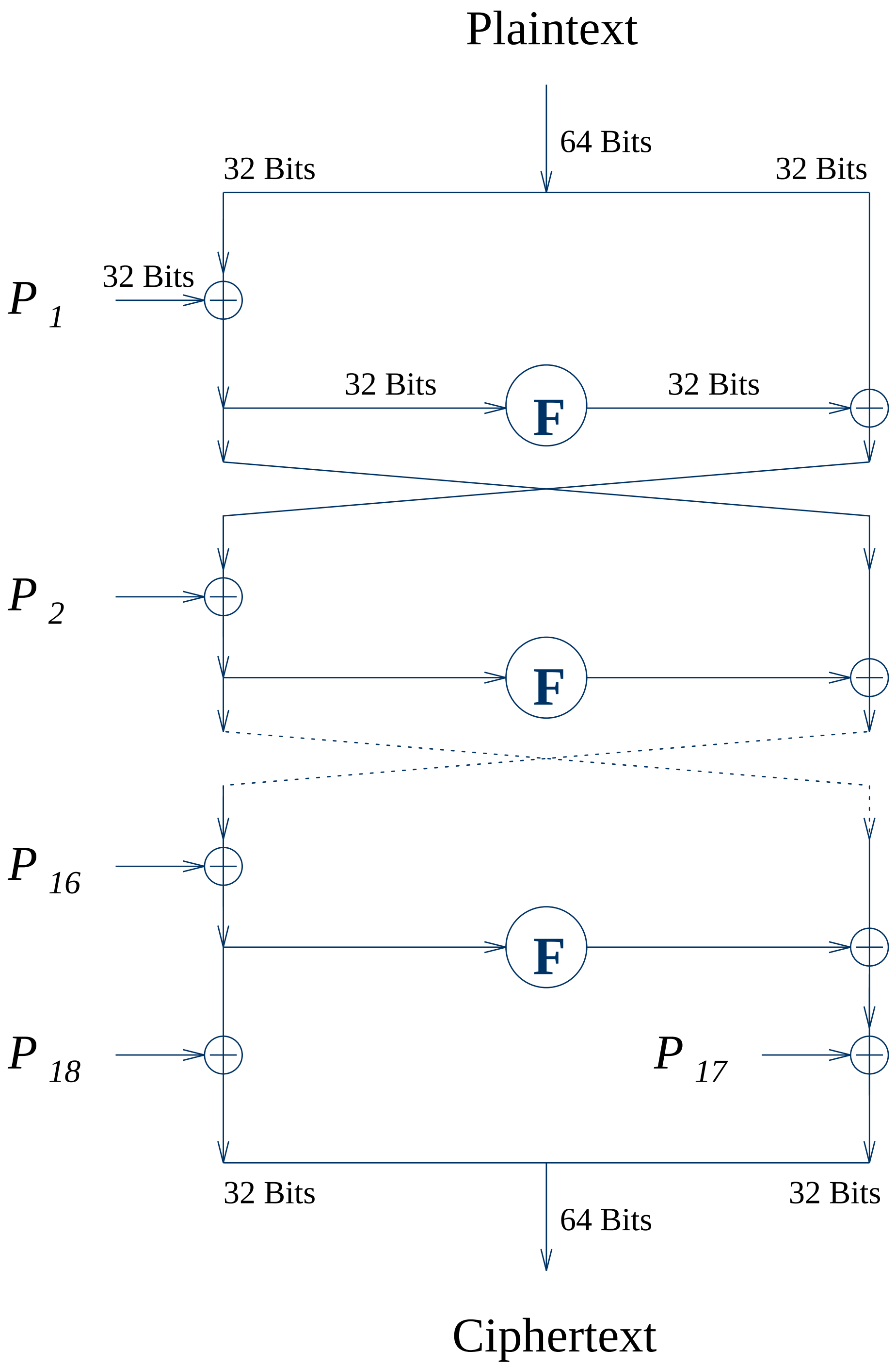
LOKI91

- Brown, Pieprzyk & Seberry.
- 64-bit block cipher.
- 64-bit key.
- 16 rounds.
- Secure against differential cryptanalysis.
  - LOKI89 wasn't.
- Weakness in key-scheduling allows **related-key chosen-plaintext attack**.
- LOKI97.

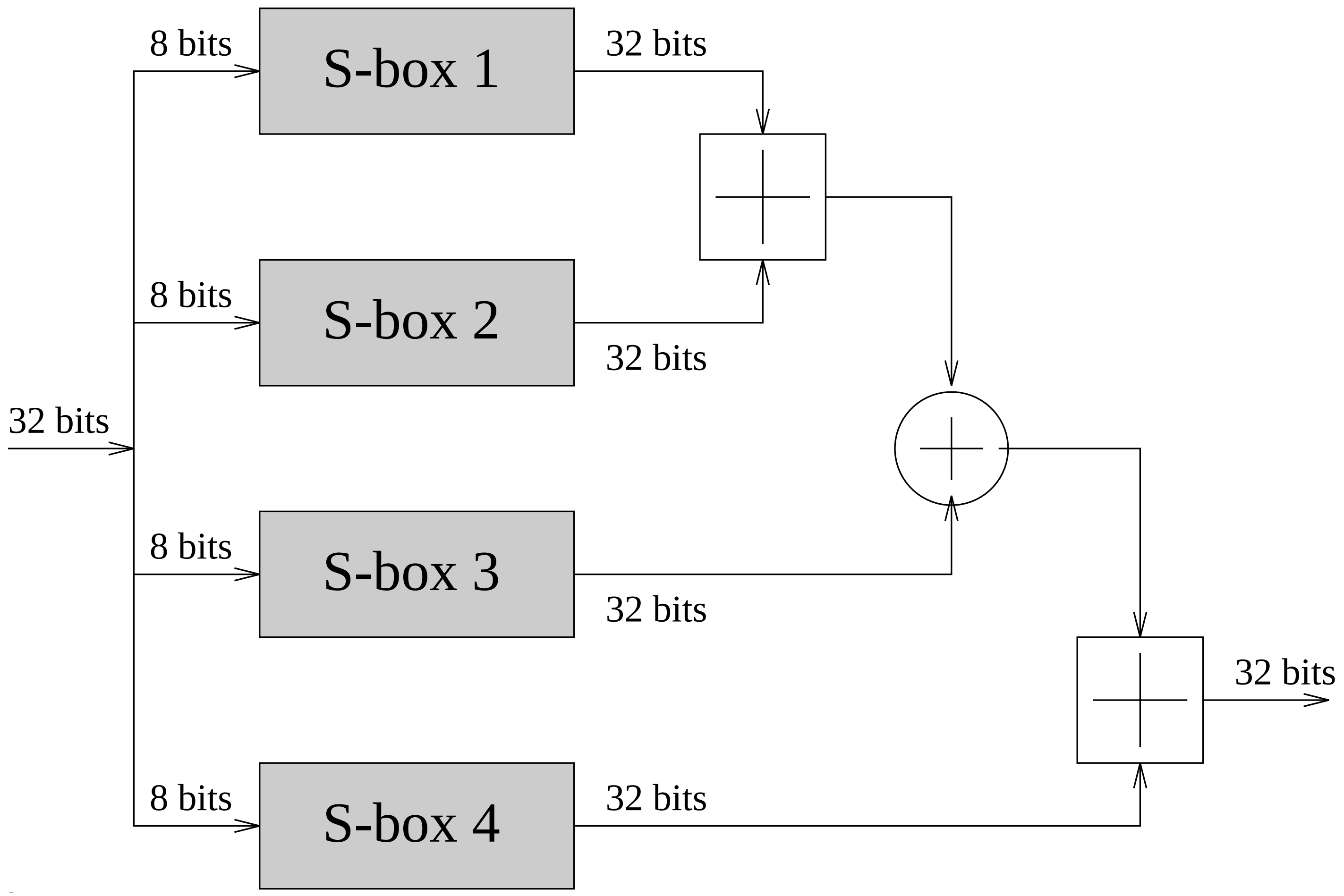


Blowfish

- Schneier (1993).
- 64-bit block cipher.
- 32→448-bit key.
- 16 round Feistel.
- One of the fastest block ciphers.
- Considered secure for key lengths>64-bits.
  - Exhaustive otherwise.
- There are attacks against reduced rounds.
- Key initialisation is relatively expensive so it isn't as useful where keys have a short lifetime.
- Unpatented.



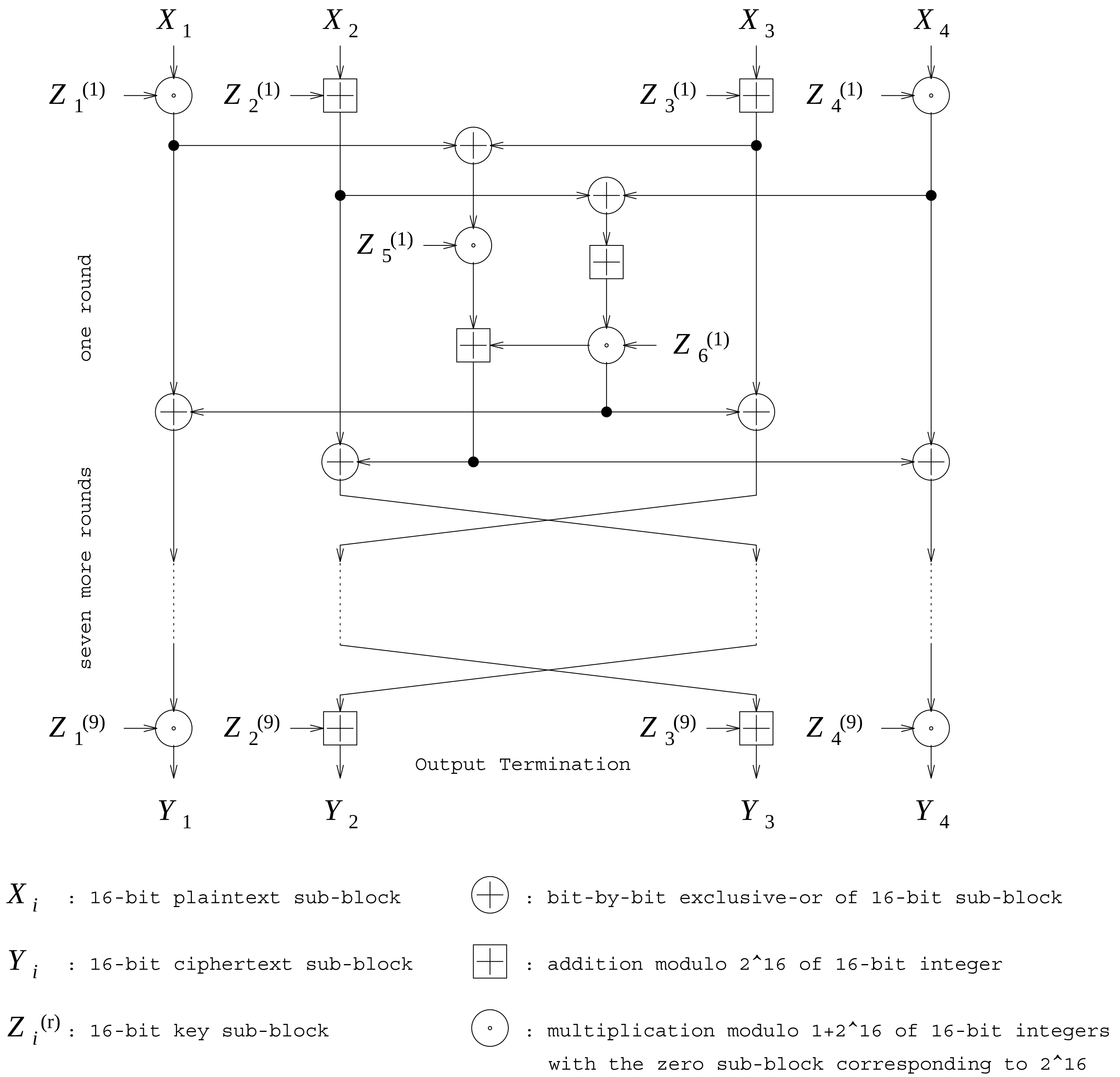
Blowfish f



- Consists of key dependent permutation and key and data dependent substitution.
- The input X is divided into 8-bit blocks *a, b, c* and *d*.
- F[X] = ((S$_{1,a}$ + S$_{2,b}$) (mod 2^{32}) ⊕ S$_{3,c}$) + S$_{4,d}$ mod 2^{32}

IDEA 1990

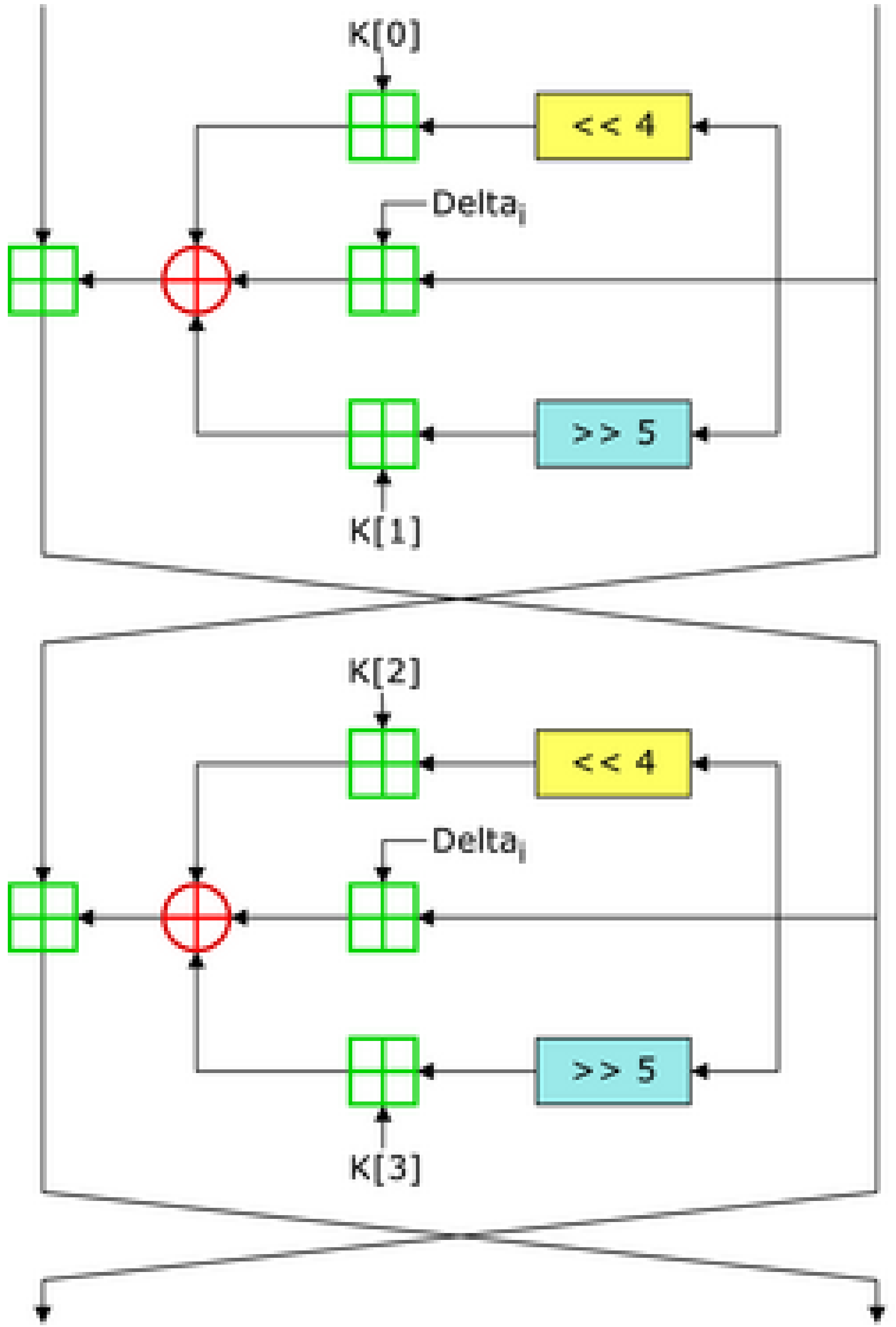
- **I**nternational **D**ata **E**ncryption **A**lgorithm (1990).
- Lai and Massey, ETH-Zurich (Swiss Federal Institute of Technology).
- Based on theoretical foundations.
  - It was designed to be resistant against differential cryptanalysis.
  - It can be shown that after 4 rounds it is resistant.
- 64-bit block cipher.
- 128-bit key.
- Operations used:
  - XOR
  - Addition modulo 2^{16}
  - Multiplication modulo $2^{16}+1$
- All operations are on 16-bit blocks: no bit permutation.
- IDEA with 8 rounds is considered “safe”.
  - There are fast attacks against reduced rounds (for example 5).
- Some weak keys have been identified, but they are few enough to not really be an issue.


X_1
X_2
X_3
X_4
$Z_1^{(1)}$
$Z_2^{(1)}$
$Z_3^{(1)}$
$Z_4^{(1)}$
$Z_5^{(1)}$
$Z_6^{(1)}$
one round
seven more rounds
$Z_1^{(9)}$
$Z_2^{(9)}$
$Z_3^{(9)}$
$Z_4^{(9)}$
Output Termination
Y_1
Y_2
Y_3
Y_4
X_i : 16-bit plaintext sub-block
Y_i : 16-bit ciphertext sub-block
$Z_i^{(r)}$: 16-bit key sub-block
⊕ : bit-by-bit exclusive-or of 16-bit sub-block
⊞ : addition modulo 2^16 of 16-bit integer
⊙ : multiplication modulo 1+2^16 of 16-bit integers with the zero sub-block corresponding to 2^16

TEA – Tiny Encryption Algorithm

- Designed by Needham and Wheeler (1994).
- TEA is unusual in that, as the name suggests, the description and implementation of the algorithm is relatively simple.
- 64-bit block cipher.
- 128-bit key.
- 64-round Feistel structure, with the rounds being paired into 32 “cycles”.
- Uses XOR, shift operations and modular addition.
- Simple (effectively trivial) key scheduling.

TEA



TEA Encryption

```
void encrypt(unsigned long* v, unsigned long* k) {
   unsigned long v0=v[0], v1=v[1], sum=0, i;                /* setup */
   unsigned long delta=0x9e3779b9;     /* key schedule constant */
   unsigned long k0=k[0], k1=k[1], k2=k[2], k3=k[3]; /* cache key */
   for (i=0; i < 32; i++) {                          /* basic cycle start */
        sum += delta;
        v0 += (v1<<4)+k0 ^ v1+sum ^ (v1>>5)+k1;
        v1 += (v0<<4)+k2 ^ v0+sum ^ (v0>>5)+k3;      /* end cycle */
   }
   v[0]=v0; v[1]=v1;
}
```

TEA Decryption

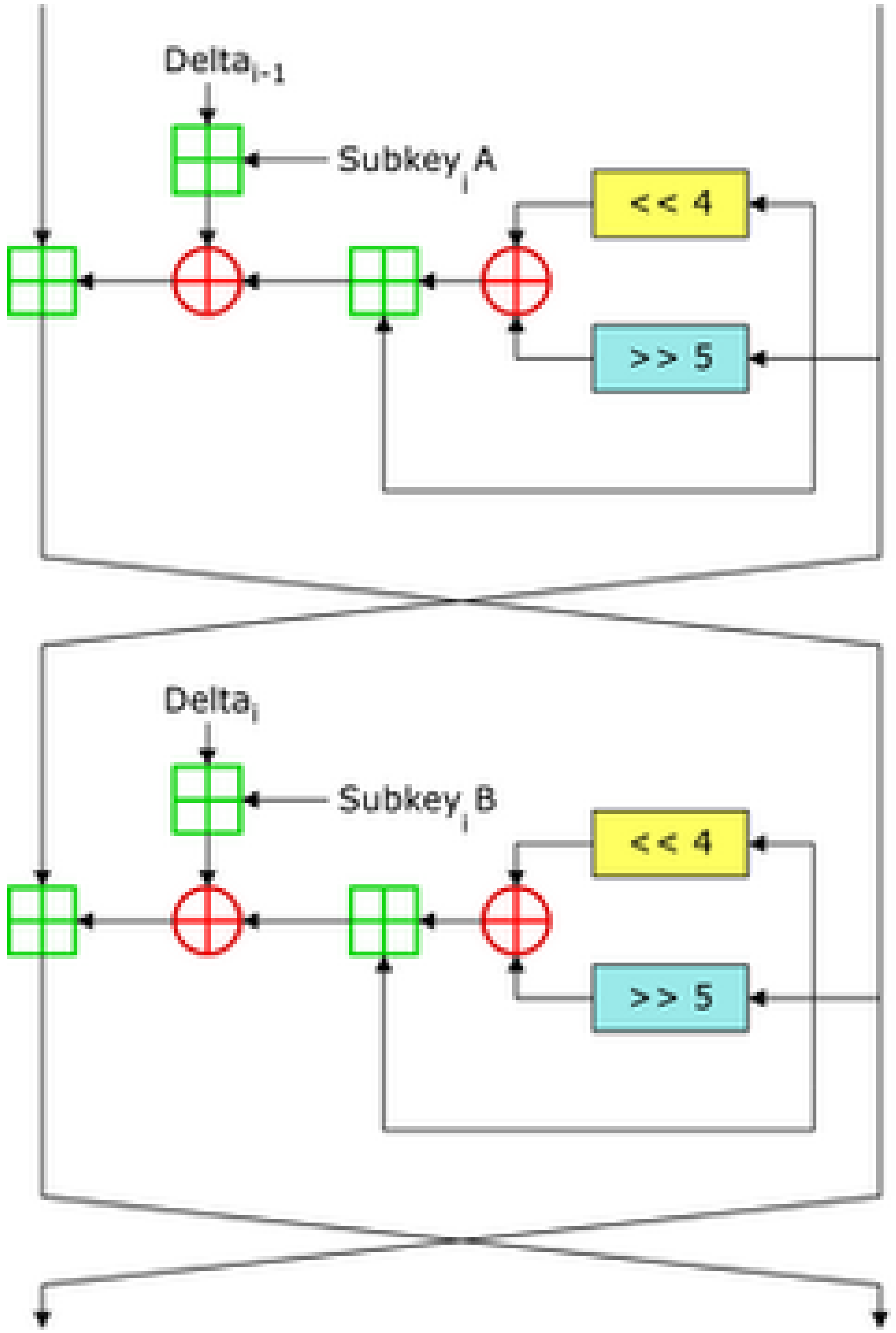
```
void decrypt(unsigned long* v, unsigned long* k) {
   unsigned long v0=v[0], v1=v[1], sum=0xC6EF3720, i; /* setup */
   unsigned long delta=0x9e3779b9;     /* key schedule constant */
   unsigned long k0=k[0], k1=k[1], k2=k[2], k3=k[3]; /* cache key */
   for (i=0; i<32; i++) {                         /* basic cycle start */
        v1 -= (v0 << 4)+k2 ^ v0+sum ^ (v0 >> 5)+k3;
        v0 -= (v1 << 4)+k0 ^ v1+sum ^ (v1 >> 5)+k1;
        sum -= delta;                                    /* end cycle */
   }
   v[0]=v0; v[1]=v1;
}
```

- TEA has some problems, in particular with key equivalencies.
  - Each key is equivalent to three others. So the effective key space is 126 bits.
  - TEA was used in the Xbox, not for encryption purposes but for hashing. We will discuss hashing later in the course but the key equivalences in TEA made it possible to corrupt the Xbox. Basically TEA was used to check in memory had been tampered with.
- TEA is also vulnerable to related-key attacks. With 2^{23} chosen plaintexts, encrypted under two related keys, 2^{32} computations are enough to break TEA.

TEA variants

- Due to the weaknesses of TEA, mainly the related-key attack, several variants have been proposed.
- XTEA (1997), Block TEA (1997), XXTEA (1998).
- The best attack against XTEA is a 27-round related-key differential attack requiring $2^{20.5}$ chosen plaintexts under a related key-pair and required $2^{115.15}$ 27-round XTEA encryptions.

XTEA

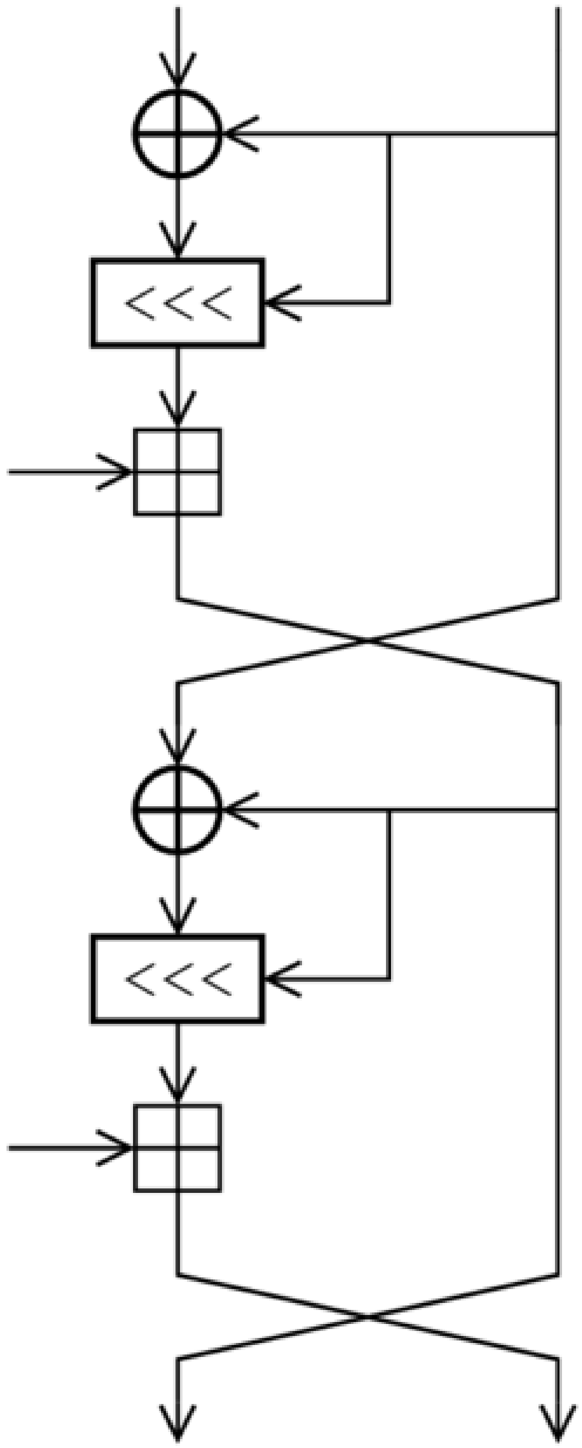


Delta_{i-1}
Subkey_iA
<< 4
>> 5
Delta_i
Subkey_iB
<< 4
>> 5

RC5

- Designed by Ron Rivest (1994).
- Another “simple” cipher.
- Also based on XOR, additions and shifts.
- 32, **64** or 128-bit block cipher.
- 0-2040 bit key, 128-bit suggested.
- A Feistel network of 1-255 rounds, with 12 suggested.
- The shifts, or rotations, are actually data-dependent. This is unusual.

RC5



The <<< are shifts.

The square boxes are modular additions.

The crossed circles are XOR operations.

Breaking RC5, the hard way ☺

- There is a competition, with prize money, for breaking various levels of RC5. This is provided by RSA Security.
  - A 56-bit RC5 was cracked in 1997.
  - Tens of thousands of computers.

  “A search of approximately 34,225 trillion keys at a peak rate of over 7 billion keys per second. With over 72 quadrillion possible keys (72,057,594,037,927,936), the winning key was reported to RSA after searching a little more than 47% of the total.”

- The RC5-64 challenge was solved by distributed.net in about four years.
- 331,252 volunteers and their machines.
- Worth $10,000.
- Finished September 2002.