chiffrement symétrique (suite et fin)

Nicolas Ollinger

M1 informatique — 2024/2025

Chiffrement par blocs

Chiffrement par blocs

$$K \in \{0,1\}^m \quad \text{clé}$$
$$M \in \{0,1\}^n \quad \text{message}$$

$$E_K : \{0,1\}^n \to \{0,1\}^n$$
$$D_K : \{0,1\}^n \to \{0,1\}^n$$

$$C = E_K(M) \quad \text{message chiffré}$$
$$M = D_K(C) \quad \text{message déchiffré}$$

AES illustré avec Mini-AES

AES

Algorithme Rijndael, vainqueur en 2000 de l’appel d’offre du NIST pour remplacer DES.

Blocs de 128 bits

Clés de 128, 192 ou 256 bits

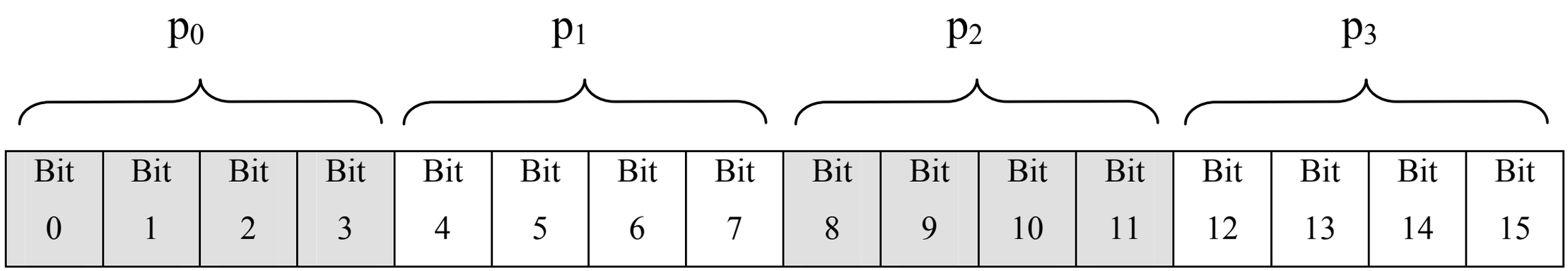
10, 12 ou 14 rondes composant 4 opérations élémentaires (*ByteSub*, *ShiftRow*, *MixColumn* et *KeyAddition*)

MiniAES

Version pédagogique très simplifiée d'AES.

Mots et clé de 16 bits vus comme 4 *nibbles*.

2 rondes.



$$P = \begin{array}{|c|c|} \hline p_0 & p_2 \\ \hline p_1 & p_3 \\ \hline \end{array}$$

Le corps de Galois GF(2^4)

- Corps fini composé de tous les nibbles (0000, 0001, 0010, …, 1100, 1101, 1110, 1111) vu comme des polynômes à coefficients binaires

$$x^3 + x + 1$$

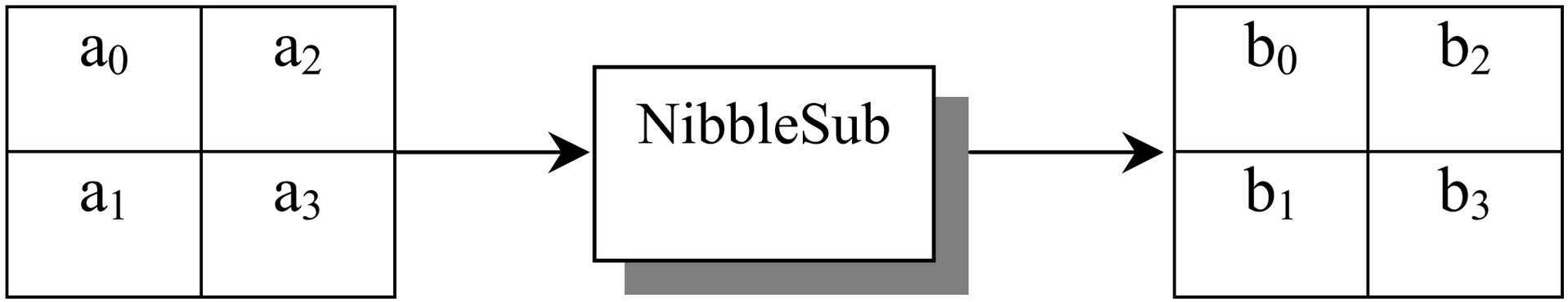
- Addition $\oplus$: c'est le XOR !

$$(x^3 + x + 1) \oplus (x^2 + x + 1) = x^3 + x^2$$

- Multiplication $\otimes$: modulo x^4+x+1

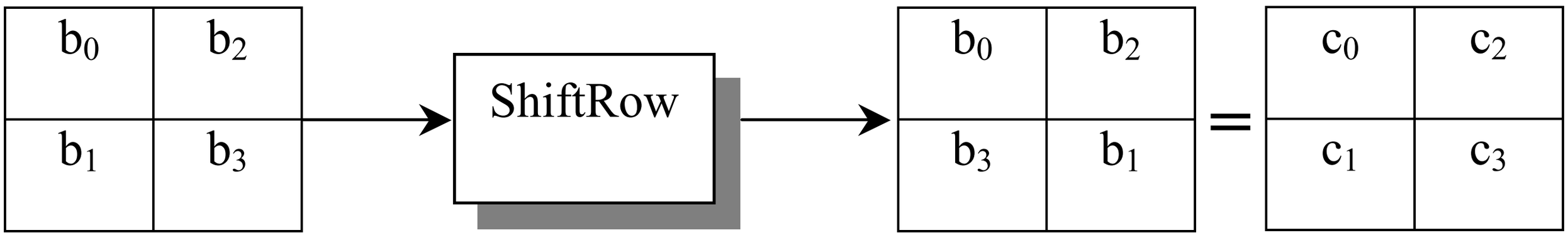
$$(x^3 + x + 1) \otimes (x^2 + x + 1) = x^5 + x^4 + 1 \pmod{x^4 + x + 1}$$
$$= x^2$$

NibbleSub γ



Input	Output	Input	Output
0000	1110	1000	0011
0001	0100	1001	1010
0010	1101	1010	0110
0011	0001	1011	1100
0100	0010	1100	0101
0101	1111	1101	1001
0110	1011	1110	0000
0111	1000	1111	0111

ShiftRow π



MixColumn θ

$$\begin{bmatrix} d_0 \\ d_1 \end{bmatrix} = \begin{bmatrix} 3 & 2 \\ 2 & 3 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} 3 & 2 \\ 2 & 3 \end{bmatrix} \begin{bmatrix} c_2 \\ c_3 \end{bmatrix}$$

Calculs dans GF(2^4)

KeyAddition σ_{Ki}

$$\begin{array}{|c|c|}\hline d_0 & d_2 \\ \hline d_1 & d_3 \\ \hline \end{array} \oplus \begin{array}{|c|c|}\hline k_0 & k_2 \\ \hline k_1 & k_3 \\ \hline \end{array} = \begin{array}{|c|c|}\hline e_0 & e_2 \\ \hline e_1 & e_3 \\ \hline \end{array}$$

Calcul des clés de rondes

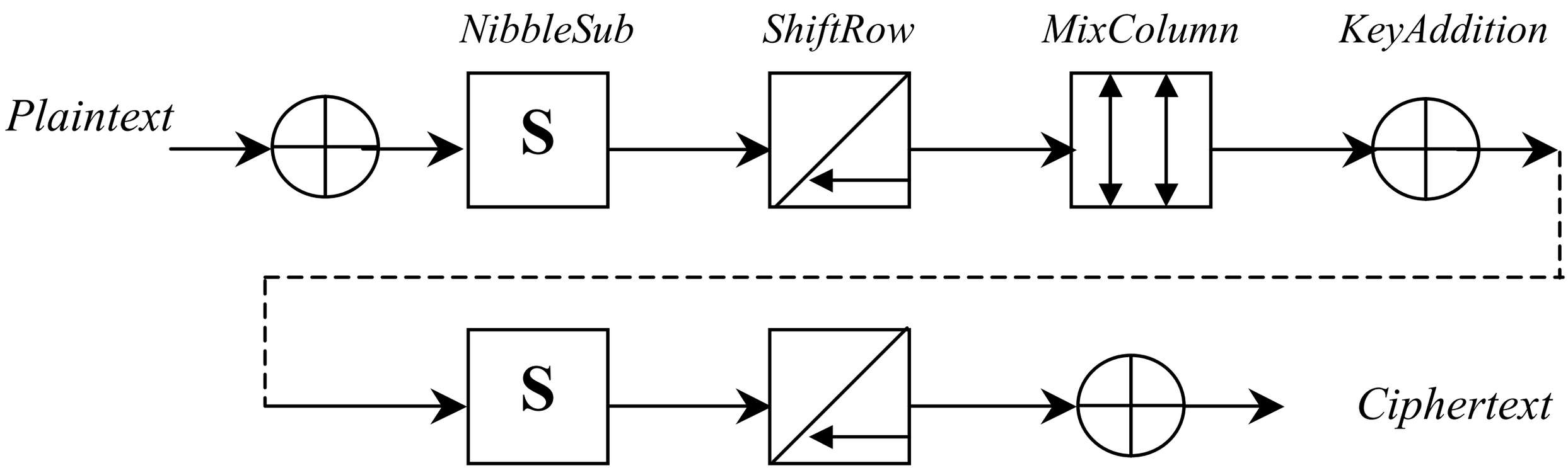
$$K_0 \begin{cases} w_0 = k_0 \\ w_1 = k_1 \\ w_2 = k_2 \\ w_3 = k_3 \end{cases}$$

$$K_1 \begin{cases} w_4 = w_0 \oplus \text{NibbleSub}(w_3) \oplus 1 \\ w_5 = w_1 \oplus w_4 \\ w_6 = w_2 \oplus w_5 \\ w_7 = w_3 \oplus w_6 \end{cases}$$

$$K_2 \begin{cases} w_8 = w_4 \oplus \text{NibbleSub}(w_7) \oplus x \\ w_9 = w_5 \oplus w_8 \\ w_{10} = w_6 \oplus w_9 \\ w_{11} = w_7 \oplus w_{10} \end{cases}$$

Chiffrement

$$\text{Enc} = \left(\sigma_{K_2} \circ \pi \circ \gamma\right) \circ \left(\sigma_{K_1} \circ \theta \circ \pi \circ \gamma\right) \circ \sigma_{K_0}$$



Déchiffrement

$$\begin{aligned}
\mathrm{Dec} &= \left(\left(\sigma_{K_2} \circ \pi \circ \gamma\right) \circ \left(\sigma_{K_1} \circ \theta \circ \pi \circ \gamma\right) \circ \sigma_{K_0}\right)^{-1} \\
&= \sigma_{K_0}^{-1} \circ \left(\sigma_{K_1} \circ \theta \circ \pi \circ \gamma\right)^{-1} \circ \left(\sigma_{K_2} \circ \pi \circ \gamma\right)^{-1} \\
&= \sigma_{K_0}^{-1} \circ \left(\gamma^{-1} \circ \pi^{-1} \circ \theta^{-1} \circ \sigma_{K_1}^{-1}\right) \circ \left(\gamma^{-1} \circ \pi^{-1} \circ \sigma_{K_2}^{-1}\right) \\
&= \sigma_{K_0} \circ \left(\gamma^{-1} \circ \pi \circ \theta \circ \sigma_{K_1}\right) \circ \left(\gamma^{-1} \circ \pi \circ \sigma_{K_2}\right)
\end{aligned}$$

Input	**Output**		**Input**	**Output**
0000	1110		1000	0111
0001	0011		1001	1101
0010	0100		1010	1001
0011	1000		1011	0110
0100	0001		1100	1011
0101	1100		1101	0010
0110	1010		1110	0000
0111	1111		1111	0101

Modes opératoires

Modes opératoires

Pour le chiffrement par flot, pour chiffrer un message de longueur *n*, il suffit générer *n* bits de flot et d'en faire le XOR avec le message.

Et dans le cas du chiffrement par blocs ?

Rq Même problématique pour le chiffrement par flot avec un séquence de messages.

NIST Special Publication 800-38A
2001 Edition

Recommendation for Block Cipher Modes of Operation

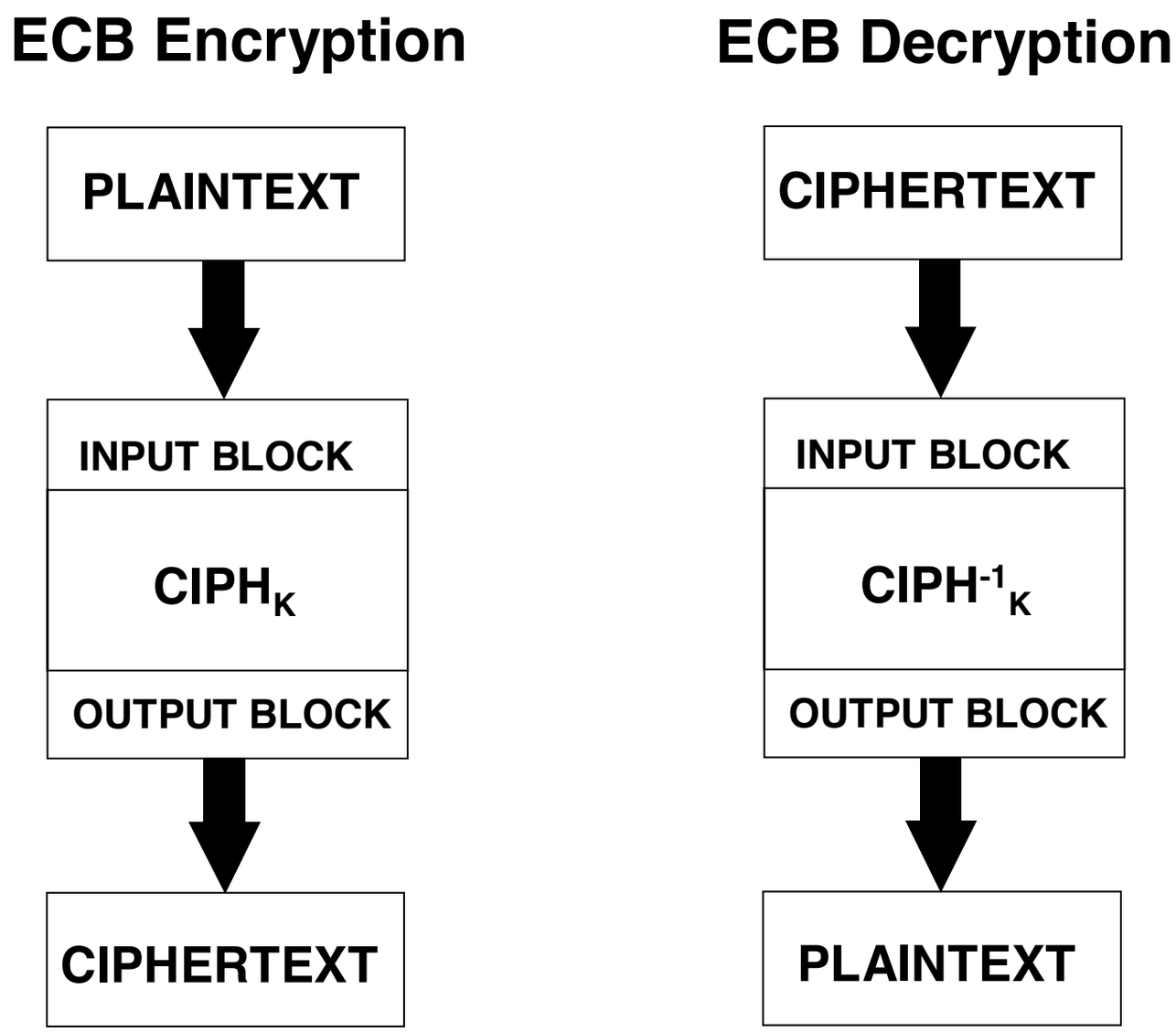
Methods and Techniques

Morris Dworkin

National Institute of Standards and Technology
Technology Administration
U.S. Department of Commerce

C O M P U T E R S E C U R I T Y

ECB (electronic codebook)



Fuite d'information : deux blocs identiques sont toujours chiffrés à l'identique !

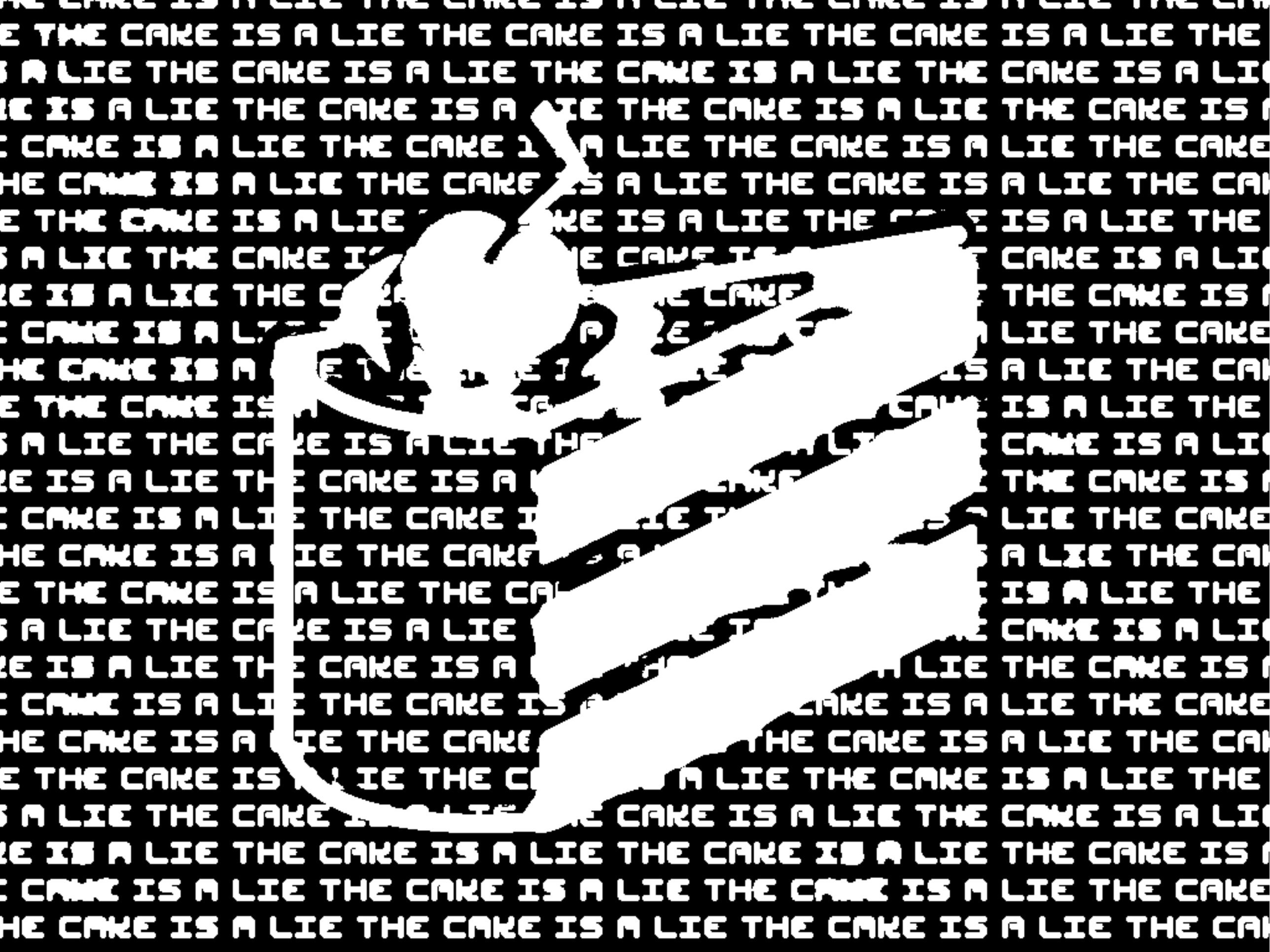
Padding

Pour les modes de chiffrement qui fonctionnent par blocs (ECB, CBC, CFB), il est nécessaire de compléter le message en clair.

Le padding doit être réversible.

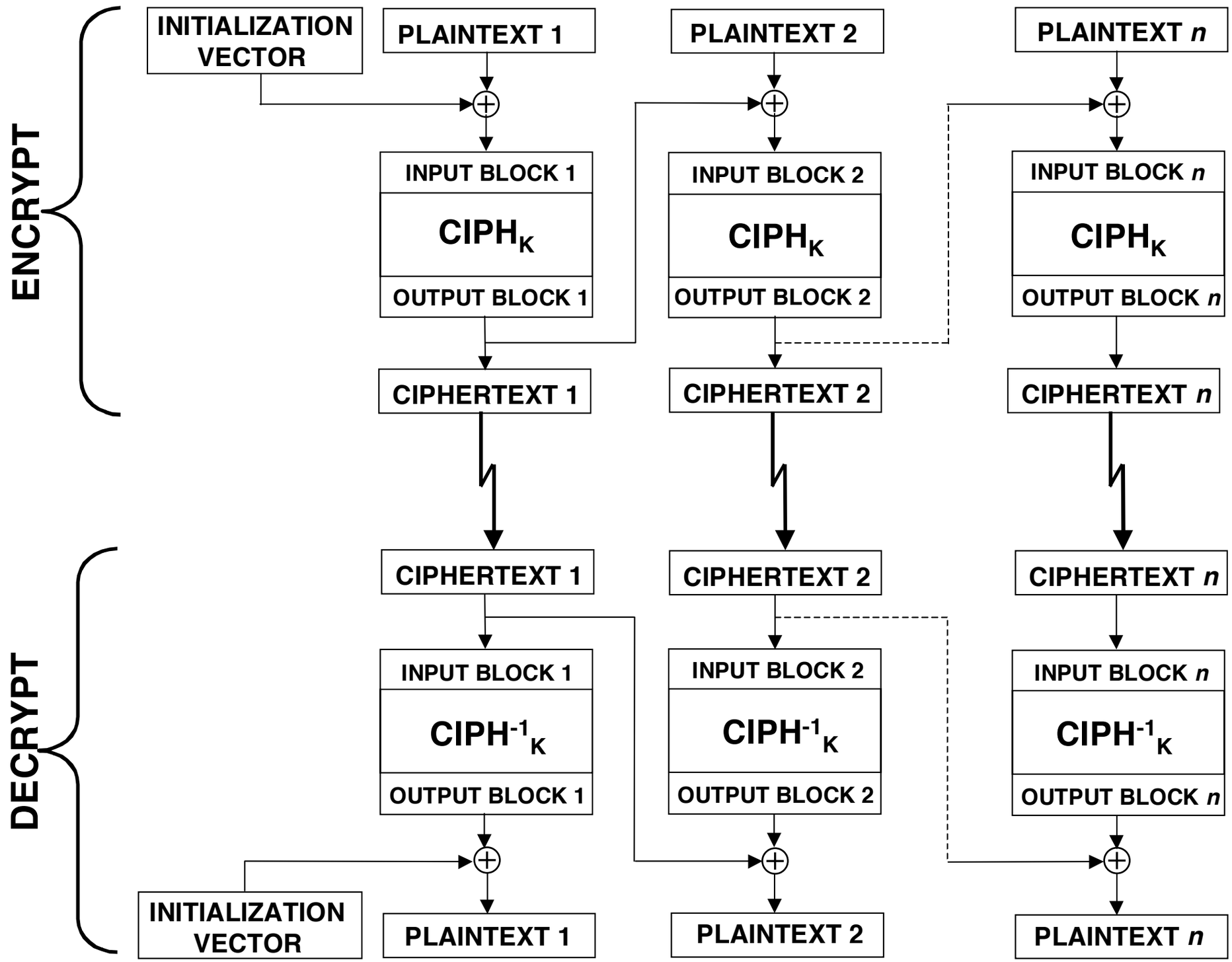
Exemples :

- train de bits 10000...000 ;
- séquence de k octets de valeur k (PKCS).


THE CAKE IS A LIE

THE CAKE IS A LIE

CBC (cipher block chaining)



Vecteur d'initialisation (IV)

Pour éviter que deux premiers blocs identiques soient chiffrés à l'identique, on ajoute un vecteur d'initialisation.

L'IV n'a pas besoin d'être secret, on peut le transmettre en clair avec le message chiffré.

Cependant l'IV ne doit pas être prédictible :

- on peut **chiffrer** une suite prédictible,
- on peut aussi utiliser un générateur pseudo-aléatoire de qualité cryptographique.

Pause exercice

Dans le cas de CBC, quelles attaques sont possibles si :

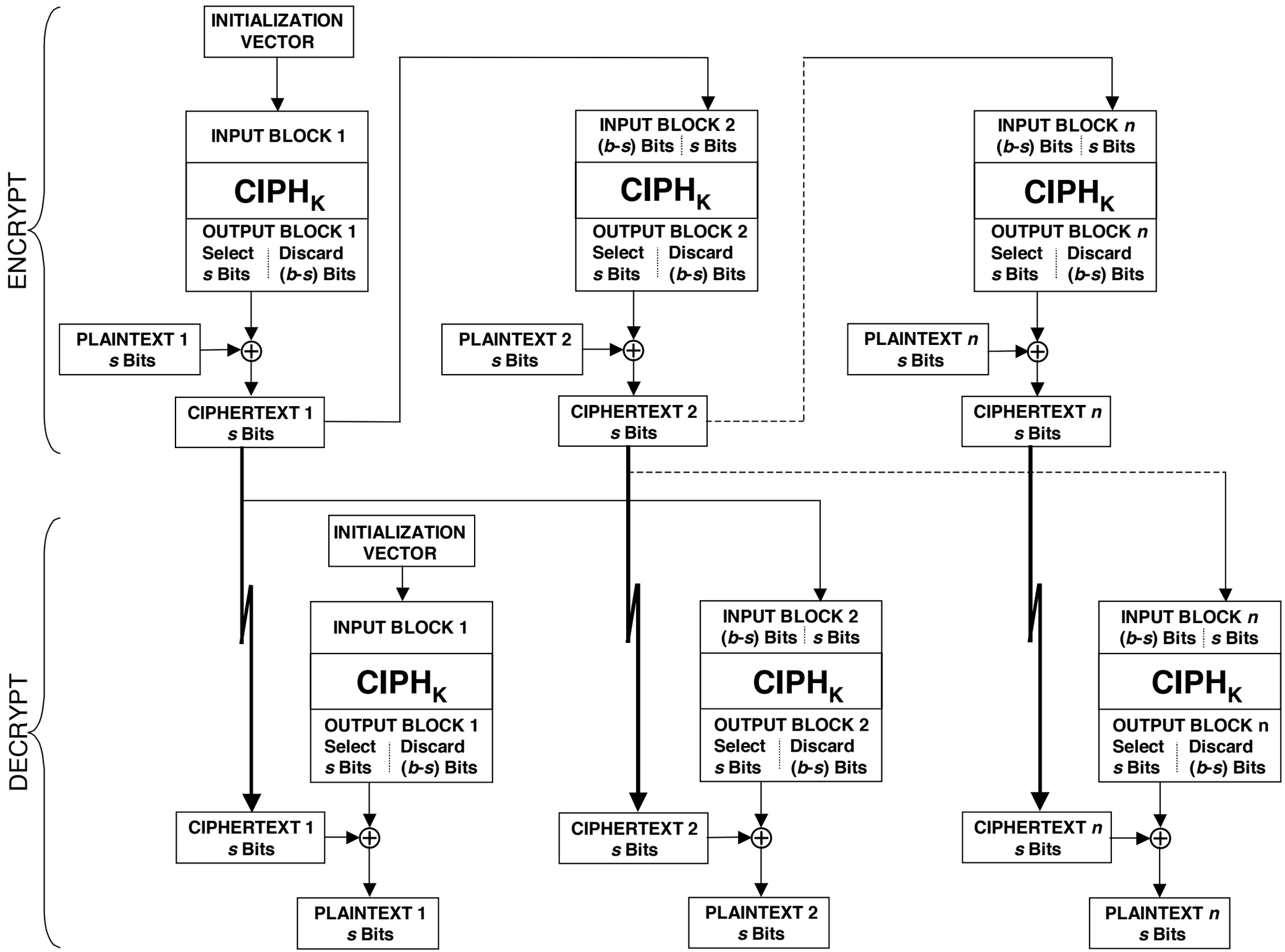
- on n’utilise pas d’IV (IV=0) ?
- on utilise une séquence d’IV facile à prédire (0, 1, 2, 3, 4, 5, …) ?

Chiffrement par flots

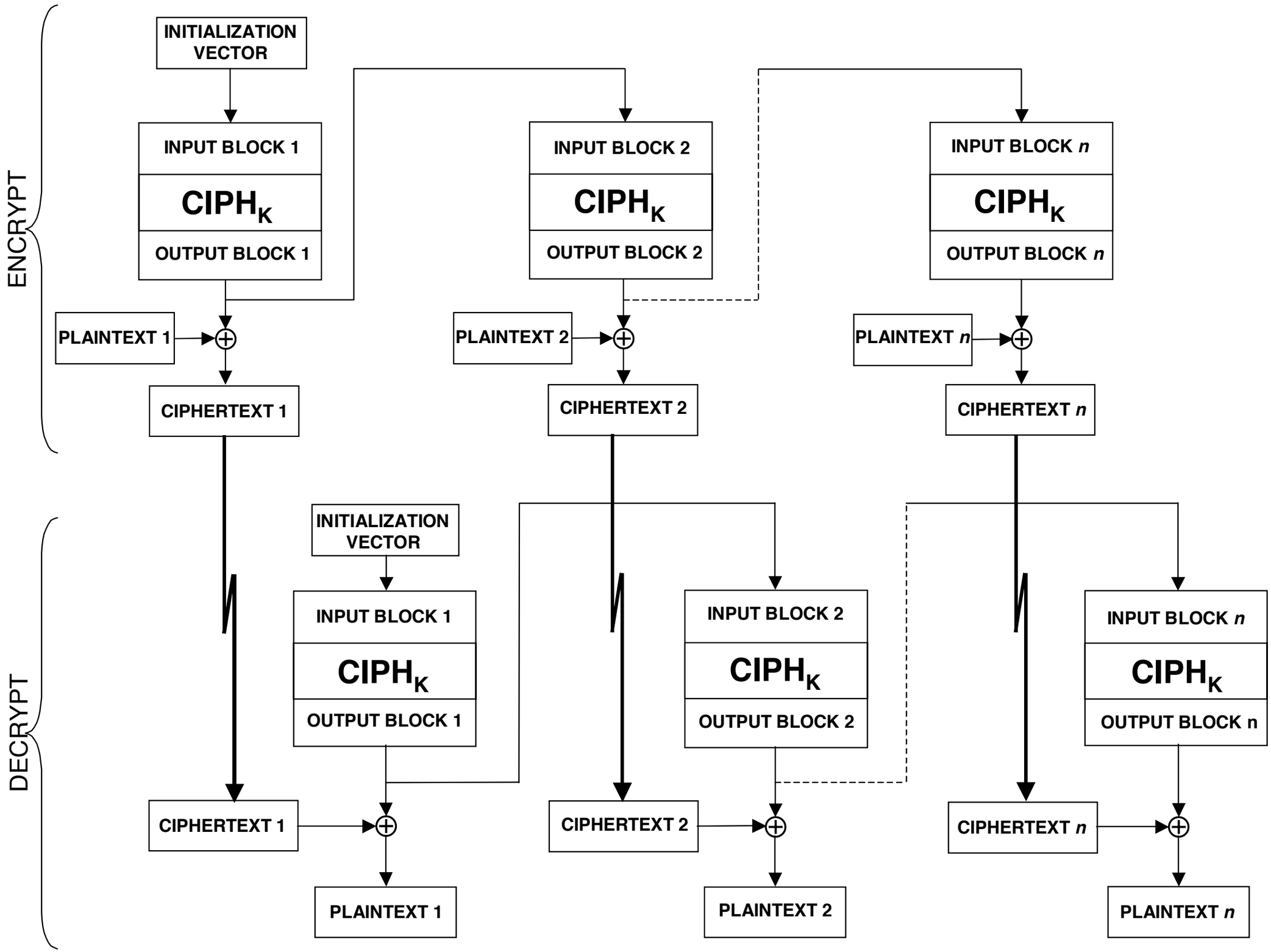
Pour éviter le padding, on peut utiliser un algorithme de chiffrement par blocs comme générateur pour du chiffrement par flots.

Idée : chiffrer uniquement à partir de la clé et du vecteur d'initialisation, indépendamment du message en clair (ou dépendant seulement des blocs précédents du message en clair) qui est combiné ensuite par XOR.

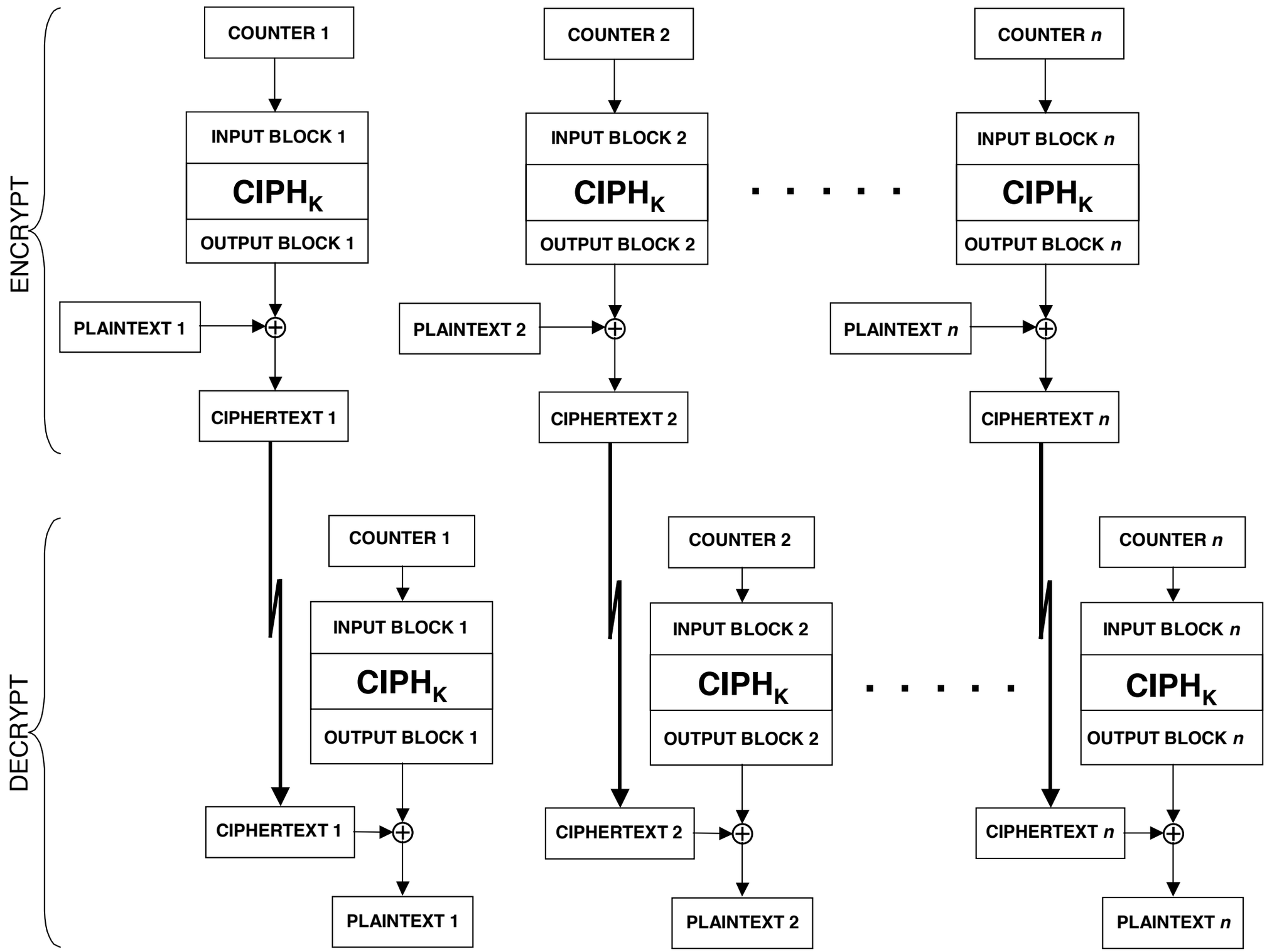
CFB (Cipher FeedBack)



OFB (Output FeedBack)



CTR (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*

Génération de compteurs

CTR nécessite une suite de blocs compteurs à usage unique. On combine deux idées :

1. incrémenter le bloc compteur pour obtenir une suite de blocs distincts ;
2. choisir un bloc initial à usage unique :
   - par exemple en continuant à compter là où on s'est arrêté ;
   - ou encore en choisissant un préfixe aléatoire unique pour le bloc.

Comparaison des modes

Comment choisir un mode opératoire ? Quelles sont les propriétés qu'on peut étudier :

- Auto-synchronisation
- Accès randomisé
- Chiffrement et/ou déchiffrement parallélisable
- Capacité à supporter des erreurs

Quels algorithmes choisir ?

Actuellement, on ne vire personne pour avoir choisit l'une des combinaisons suivantes pour garantir la **confidentialité** :

- AES-256-CBC avec IV aléatoires
- AES-256-CTR avec Nonce distincts

Attention, souvent la **confidentialité** ne suffit pas à atteindre le but recherché !

En pratique

- Bibliothèques crypto

```
from Crypto.Cipher import DES
import base64

def pad(s): return s+(8-len(s)%8)*chr(8-len(s)%8)

msg="The cake is a lie!\n"
key="1337DEADBEEF1664".decode('hex')
iv="0102030405060708".decode('hex')
cipher=DES.new(key,DES.MODE_CBC,iv)
print base64.encodestring(cipher.encrypt(pad(msg)))
```

- Couteau suisse cryptographique

```
$ echo yfUy4reTDKkwOf4Nkb5FXQZlGACALaOU \
 | openssl des-cbc -K 1337DEADBEEF1664 \
           -iv 0102030405060708 -a -d
The cake is a lie!
```

```java
import javax.crypto.*;
import javax.crypto.spec.*;
import java.util.Base64;
import javax.xml.bind.DatatypeConverter;

public class secret {
    public static void main(String args[]) {
        try {
            String message = "The cake is a lie!\n";
            byte[] iv = DatatypeConverter.parseHexBinary("0102030405060708");
            byte[] key = DatatypeConverter.parseHexBinary("1337DEADBEEF1664");
            Cipher DES = Cipher.getInstance("DES/CBC/PKCS5Padding");
            IvParameterSpec I = new IvParameterSpec(iv);
            SecretKey K = new SecretKeySpec(key,"DES");
            DES.init(Cipher.ENCRYPT_MODE, K, I);
            byte[] res = DES.doFinal(message.getBytes("utf-8"));
            System.out.println(Base64.getEncoder().encodeToString(res));
        } catch(Exception e) {
            System.out.println("Oups...");
        }
    }
}
```

Pause exercice

Un protocole naïf

Alice et Bob ont échangé une clé secrète de 256 bits et élu l'algorithme AES-256-CBC. Ils souhaitent établir un canal de communication sécurisé entre eux, sur une socket TCP, comme suit. Eve écoute les échanges « sécurisés ».

Pour transmettre un message M, on applique d'abord le padding, puis on choisit un IV aléatoire, on chiffre M avec AES-CBC et on transmet le résultat C préfixé par l'IV. À la réception, les messages sont acquittés par ACK ou NACK.

Identifier les **nombreux** problèmes de ce protocole. Expliquer comment Eve peut déchiffrer facilement les messages transmis !

Security Flaws Induced by CBC Padding Applications to SSL, IPSEC, WTLS...

Serge Vaudenay

Swiss Federal Institute of Technology (EPFL)
Serge.Vaudenay@epfl.ch

Abstract. In many standards, e.g. SSL/TLS, IPSEC, WTLS, messages are first pre-formatted, then encrypted in CBC mode with a block cipher. Decryption needs to check if the format is valid. Validity of the format is easily leaked from communication protocols in a chosen ciphertext attack since the receiver usually sends an acknowledgment or an error message. This is a side channel.
In this paper we show various ways to perform an efficient side channel attack. We discuss potential applications, extensions to other padding schemes and various ways to fix the problem.

Pause exercice (suite)

Un protocole naïf

Alice et Bob ont échangé une clé secrète de 256 bits et élu l'algorithme **AES-256-CTR**. Ils souhaitent établir un canal de communication sécurisé entre eux, sur une socket TCP, comme suit. Eve écoute les échanges « sécurisés ».

(…)

On remplace AES-256-CBC par AES-256-CTR, est-ce que c'est mieux… ou pire ?

Protocole cryptographique

Un protocole cryptographique est construit à partir de briques, les primitives cryptographiques, dans le but d’assurer un certain nombre de propriétés, typiquement :

- confidentialité ;
- intégrité ;
- authenticité ;
- non répudiabilité.

Confidentialité

- Assurer que seules les deux parties ont accès aux données échangées.
- Empêche l'**écoute** des données en transit.
- Selon le contexte cette confidentialité peut être **persistante** dans le temps.

Intégrité

- Assurer la correction et la consistance des données transmises.
- Empêche de **modifier** les données en transit.
- Empêche de **forger** des nouvelles données.
- Selon le contexte ce contrôle d'intégrité peut se faire avec ou sans **répudiabilité**.

Authenticité

- Permettre aux deux parties en présence de **valider l'identité** de l'autre partie.
- Empêche les accès non autorisés mais aussi...
- Empêche les attaques de type **homme du milieu**.
- Affaibli parfois dans le cadre client/serveur par une authentification du serveur uniquement.

Attention à la notion d'identité (IP, DNS, ... ?)

Quelques primitives cryptographiques

- **Cryptographie symétrique** (à clé secrète) ;
- Cryptographie asymétrique (à clé publique) ;
- Générateurs de nombres pseudo-aléatoires de qualité cryptographique ;
- Fonctions de hachage de qualité cryptographique ;
- *Codes d'authentification de messages (MAC)* ;
- Algorithmes de signature numérique.