Cryptographie

Chiffrement symétrique

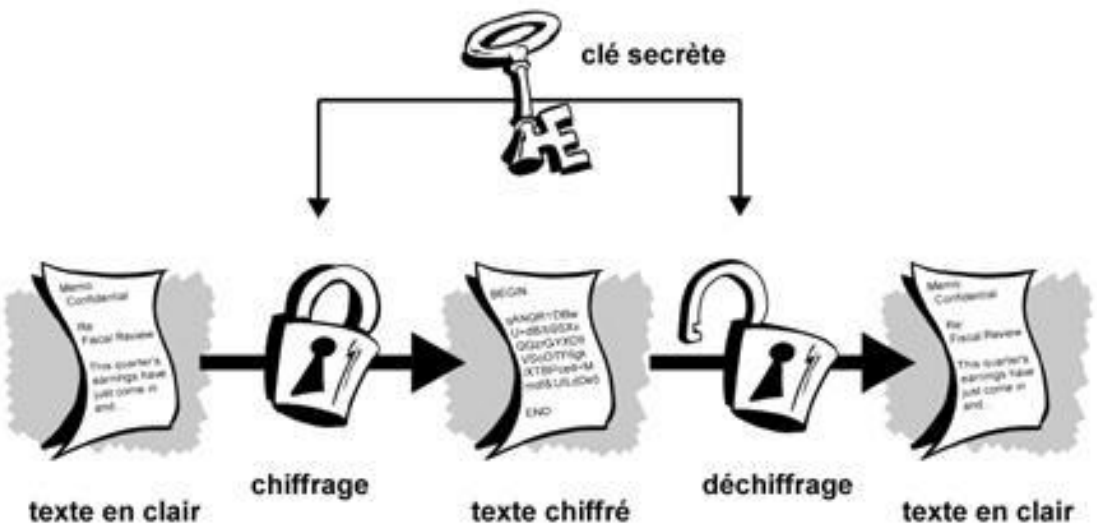
Alexander Schaub[1]

schaub.alexander@free.fr

19/11/2024

[1]DGA-MI, Bruz

Principe



- Fonction cryptographique **bijective**
- La **même** clé utilisée pour le chiffrement et le déchiffrement

Chiffrement incassable

Le Graal : un adversaire interceptant un message chiffé **ne peut pas retrouver la moindre information** concernant le message clair

⟶ Tous les messages clairs sont équiprobables

⟶ La sécurité est **inconditionnelle** (ne dépend pas de la puissance de calcul de l'attaquant)

Le chiffre de Vernam

Un chiffre de Vigenère avec une clé :

- **aussi longue** que le texte à chiffrer
- **parfaitement aléatoire**
- utilisée pour chiffrer **un seul message** (et jamais réutilisée ensuite)

Preuve

Preuve

Notons le chiffre de Vigenère $c = m \oplus k$ où k est la clé, c le chiffré, m le clair. $\oplus$ représente l'addition modulo 26 lettre par lettre (après avoir converti chaque lettre de k en nombre entre 0 et 25). Le déchiffrement est noté $m = c \ominus k$.

Si je connais c mais pas k, quelle est la probabilité que le message est un certain message m_0 donné ? C'est celle que k soit égale à $k_0 = c \ominus m_0$. Or, toutes les clés sont équiprobables, donc c'est aussi le cas de tous les messages.

Mauvaise utilisation

Deux mots de 7 lettres de la langue française ont été chiffrés avec la même clef. Les chiffrés sont **WIBXBCY** et **PIBKMAJ**. Quels couples de mots sont possibles ?

Chiffrement symétriques en pratique

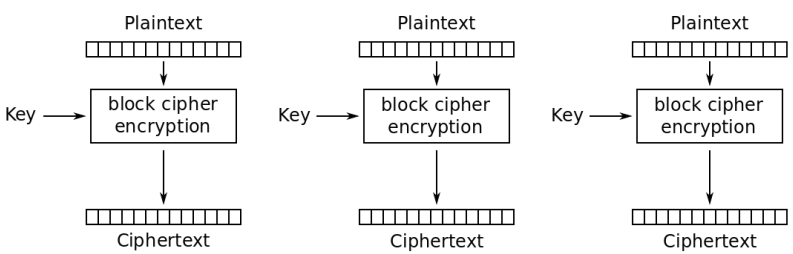
Deux grandes familles :

- Chiffrement **par flot**
  - Chiffre de Vernam (XOR au lieu de Vigenère en général) avec clé générée par un générateur aléatoire particulier
- Chiffrement par **bloc**
  - Messagé découpé en blocs de taille égale, chaque bloc traité un par un, avec un mode opératoire particulier

Chiffrement par bloc

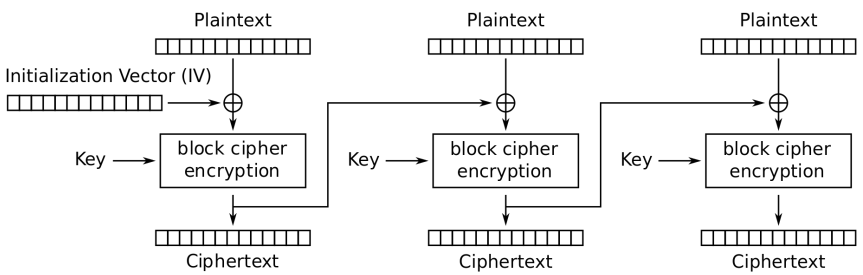
Différents modes opératoires :

ECB (Electronic CodeBook) :



Electronic Codebook (ECB) mode encryption

CBC (CypherBlock Chaining) :



Cipher Block Chaining (CBC) mode encryption

Pourquoi il faut oublier ECB tout de suite

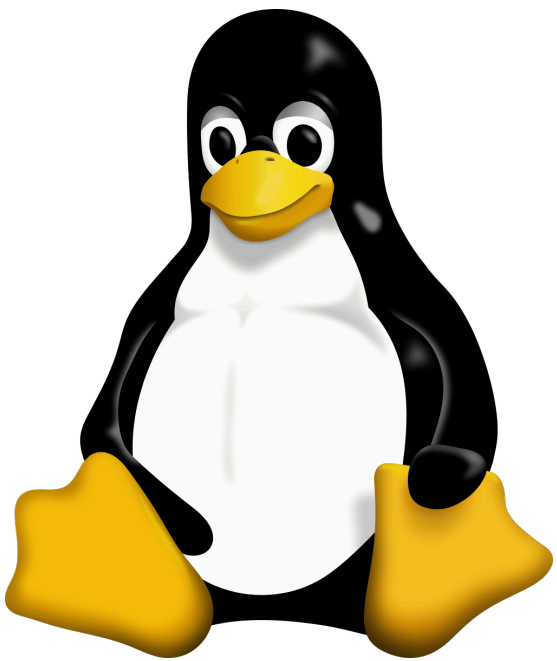


Figure 1: Tux

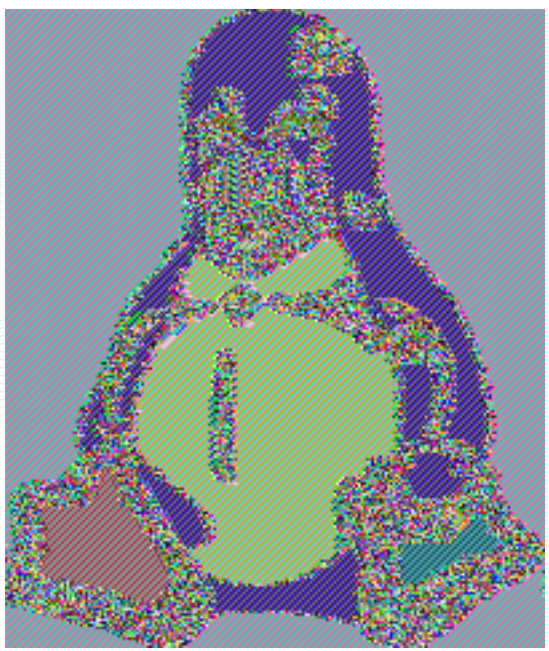


Figure 2: Tux chiffré en ECB

Figure 3: Toi après avoir chiffré en ECB

Chiffrement symétrique et bourrage (*padding*)

Que faire si la longueur du texte clair n'est pas multiple de la taille de bloc ?

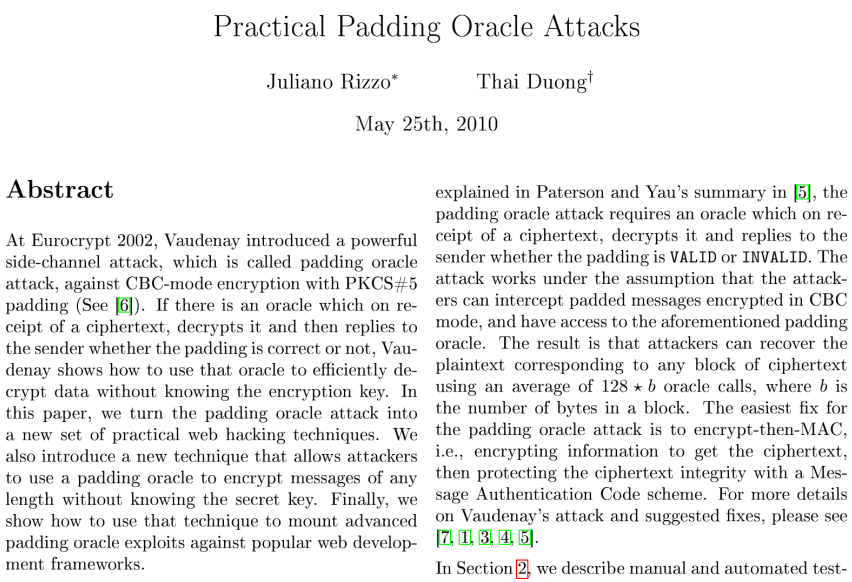
1. On rajoute des zéros
2. On rajoute de l'aléa
3. On peut pas chiffrer :(
4. Autre chose ?

Chiffrement symétrique et bourrage (*padding*) - en vrai

Différents modes existent, on rajoute soit :

- une valeur particulière d'octet (`0x80`) puis que des octets à 0 (ISO/IEC 7816-4)
- $n - 1$ octets aléatoires puis un octet valant n (ISO 10126)
- n octets égaux à n (avec n entre 1 et la longueur de bloc + 1) (PKCS#7)

Pourquoi c'est aussi compliqué ?



Practical Padding Oracle Attacks

Juliano Rizzo* Thai Duong†

May 25th, 2010

Abstract

At Eurocrypt 2002, Vaudenay introduced a powerful side-channel attack, which is called padding oracle attack, against CBC-mode encryption with PKCS#5 padding (See [6]). If there is an oracle which on receipt of a ciphertext, decrypts it and then replies to the sender whether the padding is correct or not, Vaudenay shows how to use that oracle to efficiently decrypt data without knowing the encryption key. In this paper, we turn the padding oracle attack into a new set of practical web hacking techniques. We also introduce a new technique that allows attackers to use a padding oracle to encrypt messages of any length without knowing the secret key. Finally, we show how to use that technique to mount advanced padding oracle exploits against popular web development frameworks.

explained in Paterson and Yau's summary in [5], the padding oracle attack requires an oracle which on receipt of a ciphertext, decrypts it and replies to the sender whether the padding is `VALID` or `INVALID`. The attack works under the assumption that the attackers can intercept padded messages encrypted in CBC mode, and have access to the aforementioned padding oracle. The result is that attackers can recover the plaintext corresponding to any block of ciphertext using an average of $128 \star b$ oracle calls, where b is the number of bytes in a block. The easiest fix for the padding oracle attack is to encrypt-then-MAC, i.e., encrypting information to get the ciphertext, then protecting the ciphertext integrity with a Message Authentication Code scheme. For more details on Vaudenay's attack and suggested fixes, please see [7, 1, 3, 4, 5].

In Section 2, we describe manual and automated testing techniques to find padding oracles in real life sys-

Il faut faire **très** attention au moment du déchiffrement !

Chiffrement par flot

Rappel : il s'agit de

1. Générer une séquence d'octets aléatoires
2. Effectuer un XOR entre la séquence aléatoire et le texte clair

Et voilà !

De la différence entre le bon et le mauvais aléa

Tous les générateurs d'aléa ne se valent pas

- Générateurs d'aléa "vrai" : en général basés sur un ou plusieurs phénomène physiques
  - **Avantages** : on ne peut jamais générer deux fois le même aléa, peut être complètement imprédictible
  - **Incovénients** : on ne peut jamais générer deux fois le même aléa, lent
  - **Exemple** : `/dev/random` sous linux, `https://www.random.org/`, des lancers de pièces, de dés

De la différence entre le bon et le mauvais aléa

Tous les générateurs d'aléa ne se valent pas

- Générateurs d'aléa "basique" : génère de l'"aléa" en utilisant une graine (ou **seed**)
  - **Avantage** : on peut générer plusieurs fois le même aléa (reproductibilité !), rapide en général
  - **Inconvénient** : pas réellement aléatoire (il en a juste l'air)
  - **Exemple** : `rand` (initialié par `srand`) en C, `random.random()` en Python

De la différence entre le bon et le mauvais aléa

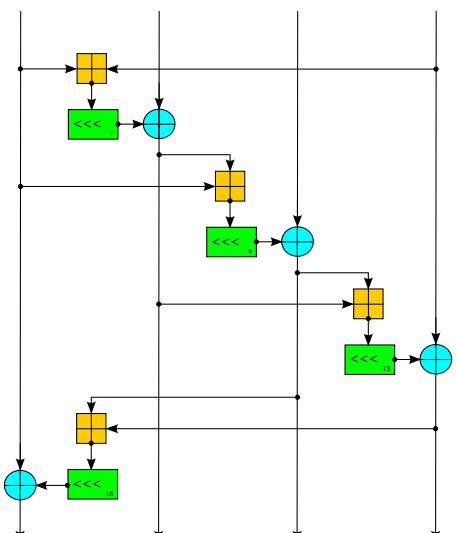
Tous les générateurs d'aléa ne se valent pas

- Générateurs d'aléa "cryptographique" : génère de l'"aléa" en utilisant une graine (ou **seed**), mais en mieux…
  - **Avantage** : on peut générer plusieurs fois le même aléa (reproductibilité !), rapide en général, peut être utilisé en crypto
  - **Inconvénient** : plus lent qu'un générateur basique
  - **Exemple** : `/dev/random` sour Linux, RC4, Chacha20, …

Mais au fond quelle différence y a-t-il entre le bon et le mauvais générateur d'aléa ?

Aléa vrai	Aléa basique	Aléa cryptographique
TRNG	PRNG	CSPRNG
Complètement imprédictible	Rapide et uniforme	Imprédictible en pratique
Génération de clé, IV, …	Simulation (pas en crypto !!)	Chiffrement par flot

Un exemple de CSPRNG : Salsa20



```
b ^= (a + d) <<< 7;
c ^= (b + a) <<< 9;
d ^= (c + b) <<< 13;
a ^= (d + c) <<< 18;
```

Salsa20 en pratique

Pour l'utiliser, il faut :

- un **nonce** (ou IV) : valeur **unique**, potentiellement publique (128 bits)
- une **clé** (256 bits)

Et on obtient une séquence pseudo-aléatoire imprédictible en pratique

Ne **jamais** chiffrer deux messages avec la **même clé** et le **même nonce** !

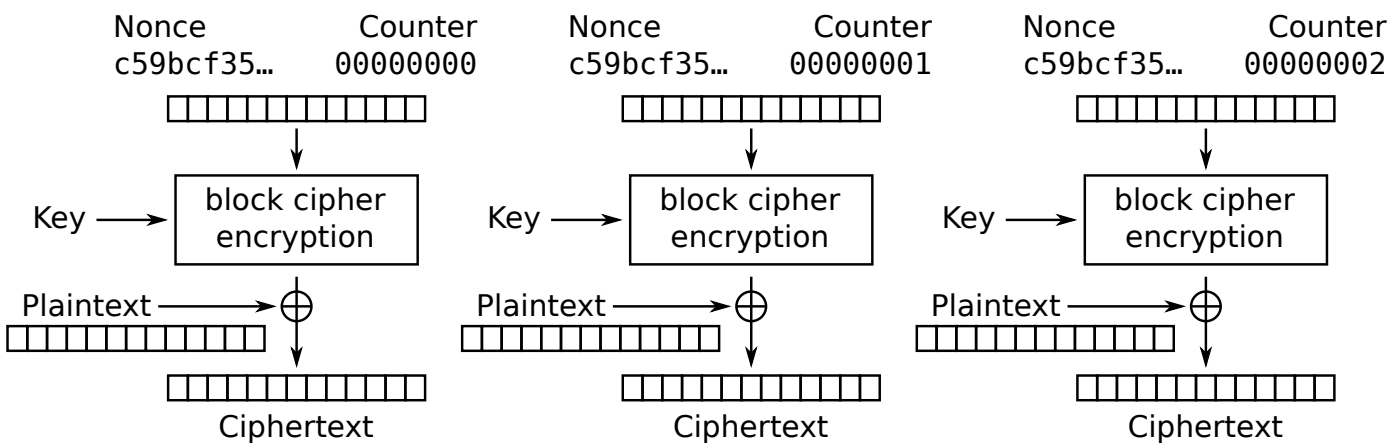
Variantes

Il existe aussi

- Salsa20/8 et Salsa20/12 (plus rapides, moins sûres ?)
- XSalsa20 (nonce plus grand)
- Chacha20 et XChacha20 (variantes)

Le tout par l'éternel **Daniel J. Bernstein**

Transformer un chiffrement par bloc en chiffrement par flot



Counter (CTR) mode encryption

Exemples de chiffrements par bloc

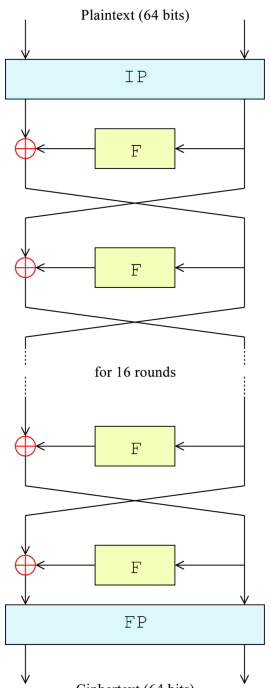


Figure 4: Schéma de Feistel

Chiffrement

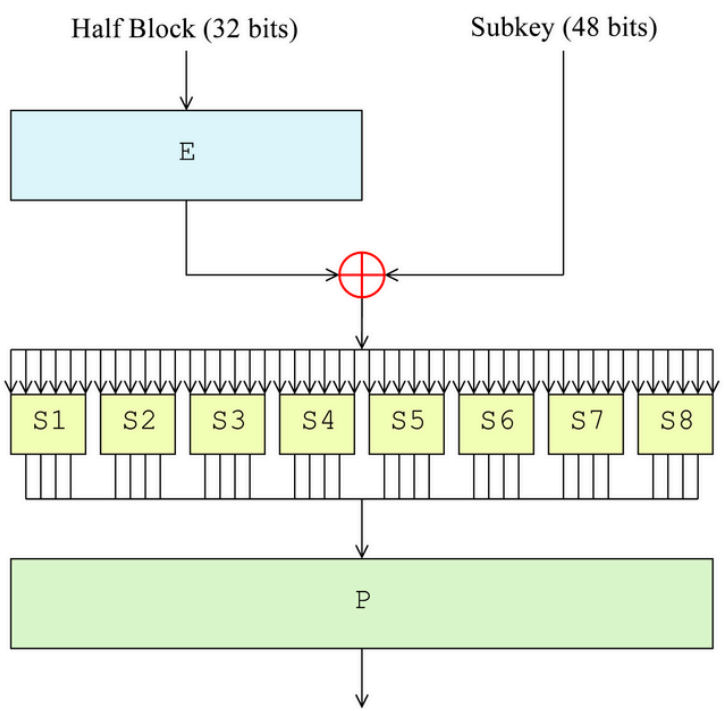
- $L_i = R_{i-1}$
- $R_i = L_{i-1} + F(R_{i-1})$

Déchiffrement

- $R_i = L_{i+1}$
- $L_i = R_{i+1} + F(L_{i+1})$

Note : F n'a **pas** besoin d'être inversible !

Data Encryption Standard (DES)



E : extension de 32 bits vers 48 bits (en dupliquant des bits)

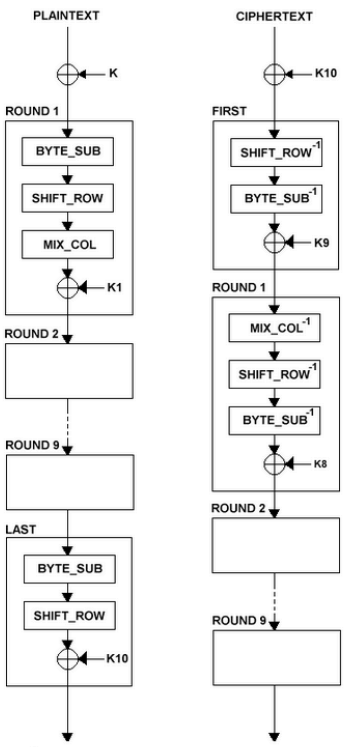
S_i : Boîte de substituion non-linéaire de 6 vers 4 bits

P : permutation des bits pour une meilleure **diffusion**

DES : les problèmes

- Longueur des blocs : 64 bits 👎
- Longueur des clés : 64 bits 👎 dont 56 utiles 👎 👎
- Valeurs des S-Box : paraissent arbitraires (mais en fait : la NSA a bien fait les choses ! 👍 pour une fois…)
- Se retrouve parfois utilisé en mode Triple DES (triple chiffrement avec plusieurs clés différentes) pour une sécurité de 112 bits

Le chiffrement par blocs moderne : AES



- Etat interne : matrice de 4×4 octets
- BYTE_SUB : substitution simple octet par octet (SBOX)
- SHIFT_ROW : permutation simple, on décale la i-ème ligne de $i - 1$ positions vers la droite
- MIX_COL : opération linéaire colonne par colonne qui améliore la diffusion

Principales caractéristiques d'AES

- Taille de clés : 128 bits, 192 bits ou 256 bits selon variante
- Taille de blocs : 128 bits
- SBOX : choisie avec soin…

⟶ LE chiffrement par blocs le plus utilisé aujourd'hui

Pour conclure

On a vu comment **cacher le contenu** des messages mais…

- Comment peut-on **échanger les clés** entre participants légitimes ?
- Comment garantir **l'origine** des messages ?
- Comment garantir que les messages n'ont pas été **modifiés** par un attaquant ?

Les réponses dans la suite du module !