Cours de Cryptographie

A. B. Dragut

Univ. Aix-Marseille, IUT Aix en Provence

Plan

- Algorithmes et propriétés – suite
  - Chiffrement par blocs
  - Blowfish
  - ...

Plan

- Algorithmes et propriétés – suite
  - Chiffrement par blocs
  - Blowfish
  - ...

Plan

- Algorithmes et propriétés – suite
  - Chiffrement par blocs
  - Blowfish
  - . . .

Plan

- Algorithmes et propriétés – suite
  - Chiffrement par blocs
  - Blowfish
  - ...

Chiffrement par blocs

- Chiffrement par blocs
  - bloc de n bits en bloc de n bits
- Un message de longueur arbitraire
  - $\Longrightarrow$ le couper en blocs de n bits $\in \Sigma^n$

Chiffrement par blocs

- Chiffrement par blocs
  - bloc de n bits en bloc de n bits
- Un message de longueur arbitraire
  - $\Longrightarrow$ le couper en blocs de n bits $\in \Sigma^n$

Chiffrement par blocs

- Chiffrement par blocs
  - bloc de n bits en bloc de n bits
- Un message de longueur arbitraire
  - $\Longrightarrow$ le couper en blocs de n bits $\in \Sigma^n$

Chiffrement par blocs

- Chiffrement par blocs
  - bloc de n bits en bloc de n bits
- Un message de longueur arbitraire
  - $\Longrightarrow$ le couper en blocs de n bits $\in \Sigma^n$

Elements de base

- Chiffrement par blocs à n bits
  - une fonction $e : \Sigma^n \times K \rightarrow \Sigma^n$
- Fonction de codage inversible :

$$\begin{aligned} \text{chiffré : } y &= e_k(x) \\ x &= d_k(y) \end{aligned}$$

Elements de base

- Chiffrement par blocs à n bits
  - une fonction $e : \Sigma^n \times K \rightarrow \Sigma^n$
- Fonction de codage inversible :

$$\begin{aligned} \text{chiffré} : y &= e_k(x) \\ x &= d_k(y) \end{aligned}$$

Elements de base

- Chiffrement par blocs à n bits
  - une fonction $e : \Sigma^n \times K \rightarrow \Sigma^n$
- Fonction de codage inversible :

$$\begin{aligned} \text{chiffré} : y &= e_k(x) \\ x &= d_k(y) \end{aligned}$$

Elements de base

- Chiffrement par blocs à n bits
  - une fonction $e : \Sigma^n \times K \rightarrow \Sigma^n$
- Fonction de codage inversible :

$$\begin{aligned} \text{chiffré} : y &= e_k(x) \\ x &= d_k(y) \end{aligned}$$

Elements de base

- Chiffrement par blocs à n bits
  - une fonction $e : \Sigma^n \times K \rightarrow \Sigma^n$
- Fonction de codage inversible :

$$\begin{aligned} \text{chiffré} : y &= e_k(x) \\ x &= d_k(y) \end{aligned}$$

Propriétés recherchées

- $\Longrightarrow$ Efficacité :
  - génération de la clé
  - transmission
  - stockage
- Sécurité : résistance à la cryptanalyse
  - $\Longrightarrow$ divers scénarios de modification de données

Propriétés recherchées

- $\Longrightarrow$ Efficacité :
  - génération de la clé
  - transmission
  - stockage
- Sécurité : résistance à la cryptanalyse
  - $\Longrightarrow$ divers scénarios de modification de données

Propriétés recherchées

- $\Longrightarrow$ Efficacité :
  - génération de la clé
  - transmission
  - stockage
- Sécurité : résistance à la cryptanalyse
  - $\Longrightarrow$ divers scénarios de modification de données

Propriétés recherchées

- $\Longrightarrow$ Efficacité :
  - génération de la clé
  - transmission
  - stockage
- Sécurité : résistance à la cryptanalyse
  - $\Longrightarrow$ divers scénarios de modification de données

Propriétés recherchées

- $\Longrightarrow$ Efficacité :
  - génération de la clé
  - transmission
  - stockage
- Sécurité : résistance à la cryptanalyse
  - $\Longrightarrow$ divers scénarios de modification de données

Propriétés recherchées

- $\Longrightarrow$ Efficacité :
  - génération de la clé
  - transmission
  - stockage
- Sécurité : résistance à la cryptanalyse
  - $\Longrightarrow$ divers scénarios de modification de données

Modes de chiffrement

- Modes de chiffrement
  - ECB (Electronic CodeBook)
  - CBC (Cipher-Block Chaining)
  - CFB (Cipher Feedback)
  - OFB (Output Feedback)
- $\rightsquigarrow$ différentes propriétés de propagation d'erreurs
- $\rightsquigarrow$ d'autres modes assurent le chiffrement et l'authentification

Modes de chiffrement

- Modes de chiffrement
  - ECB (Electronic CodeBook)
  - CBC (Cipher-Block Chaining)
  - CFB (Cipher Feedback)
  - OFB (Output Feedback)
- $\rightsquigarrow$ différentes propriétés de propagation d'erreurs
- $\rightsquigarrow$ d'autres modes assurent le chiffrement et l'authentification

Modes de chiffrement

- Modes de chiffrement
  - ECB (Electronic CodeBook)
  - CBC (Cipher-Block Chaining)
  - CFB (Cipher Feedback)
  - OFB (Output Feedback)
- $\rightsquigarrow$ différentes propriétés de propagation d'erreurs
- $\rightsquigarrow$ d'autres modes assurent le chiffrement et l'authentification

Modes de chiffrement

- Modes de chiffrement
  - ECB (Electronic CodeBook)
  - CBC (Cipher-Block Chaining)
  - CFB (Cipher Feedback)
  - OFB (Output Feedback)
- $\rightsquigarrow$ différentes propriétés de propagation d'erreurs
- $\rightsquigarrow$ d'autres modes assurent le chiffrement et l'authentification

Modes de chiffrement

- Modes de chiffrement
  - ECB (Electronic CodeBook)
  - CBC (Cipher-Block Chaining)
  - CFB (Cipher Feedback)
  - OFB (Output Feedback)
- $\rightsquigarrow$ différentes propriétés de propagation d'erreurs
- $\rightsquigarrow$ d'autres modes assurent le chiffrement et l'authentification

Modes de chiffrement

- Modes de chiffrement
  - ECB (Electronic CodeBook)
  - CBC (Cipher-Block Chaining)
  - CFB (Cipher Feedback)
  - OFB (Output Feedback)
- $\rightsquigarrow$ différentes propriétés de propagation d'erreurs
- $\rightsquigarrow$ d'autres modes assurent le chiffrement et l'authentification

Modes de chiffrement

- Modes de chiffrement
  - ECB (Electronic CodeBook)
  - CBC (Cipher-Block Chaining)
  - CFB (Cipher Feedback)
  - OFB (Output Feedback)
- $\rightsquigarrow$ différentes propriétés de propagation d'erreurs
- $\rightsquigarrow$ d'autres modes assurent le chiffrement et l'authentification

ECB

- le message à chiffrer est divisé en blocs à n bits
- chaque bloc est chiffré séparément
  - $\Longrightarrow$ deux blocs identiques $\Longrightarrow$ le même code
  - $\Longrightarrow$ permutation dans les blocs chiffrés $\Longrightarrow$ pareil pour les blocs en clair.
  - $\Longrightarrow$ padding aléatoire obligatoire
- Cryptanalyse : on cherche les séquences identiques $\Longrightarrow$ "dictionnaire"
- Pas de propagation d'erreur.

ECB

- le message à chiffrer est divisé en blocs à n bits
- chaque bloc est chiffré séparément
  - $\Longrightarrow$ deux blocs identiques $\Longrightarrow$ le même code
  - $\Longrightarrow$ permutation dans les blocs chiffrés $\Longrightarrow$ pareil pour les blocs en clair.
  - $\Longrightarrow$ padding aléatoire obligatoire
- Cryptanalyse : on cherche les séquences identiques $\Longrightarrow$ "dictionnaire"
- Pas de propagation d'erreur.

ECB

- le message à chiffrer est divisé en blocs à n bits
- chaque bloc est chiffré séparément
  - $\Longrightarrow$ deux blocs identiques $\Longrightarrow$ le même code
  - $\Longrightarrow$ permutation dans les blocs chiffrés $\Longrightarrow$ pareil pour les blocs en clair.
  - $\Longrightarrow$ padding aléatoire obligatoire
- Cryptanalyse : on cherche les séquences identiques $\Longrightarrow$ "dictionnaire"
- Pas de propagation d'erreur.

ECB

- le message à chiffrer est divisé en blocs à n bits
- chaque bloc est chiffré séparément
  - $\Longrightarrow$ deux blocs identiques $\Longrightarrow$ le même code
  - $\Longrightarrow$ permutation dans les blocs chiffrés $\Longrightarrow$ pareil pour les blocs en clair.
  - $\Longrightarrow$ padding aléatoire obligatoire
- Cryptanalyse : on cherche les séquences identiques $\Longrightarrow$ "dictionnaire"
- Pas de propagation d'erreur.

ECB

- le message à chiffrer est divisé en blocs à n bits
- chaque bloc est chiffré séparément
  - $\Longrightarrow$ deux blocs identiques $\Longrightarrow$ le même code
  - $\Longrightarrow$ permutation dans les blocs chiffrés $\Longrightarrow$ pareil pour les blocs en clair.
  - $\Longrightarrow$ padding aléatoire obligatoire
- Cryptanalyse : on cherche les séquences identiques $\Longrightarrow$ "dictionnaire"
- Pas de propagation d'erreur.

ECB

- le message à chiffrer est divisé en blocs à n bits
- chaque bloc est chiffré séparément
  - $\Longrightarrow$ deux blocs identiques $\Longrightarrow$ le même code
  - $\Longrightarrow$ permutation dans les blocs chiffrés $\Longrightarrow$ pareil pour les blocs en clair.
  - $\Longrightarrow$ padding aléatoire obligatoire
- Cryptanalyse : on cherche les séquences identiques $\Longrightarrow$ "dictionnaire"
- Pas de propagation d'erreur.

ECB

- le message à chiffrer est divisé en blocs à n bits
- chaque bloc est chiffré séparément
  - $\Longrightarrow$ deux blocs identiques $\Longrightarrow$ le même code
  - $\Longrightarrow$ permutation dans les blocs chiffrés $\Longrightarrow$ pareil pour les blocs en clair.
  - $\Longrightarrow$ padding aléatoire obligatoire
- Cryptanalyse : on cherche les séquences identiques $\Longrightarrow$ "dictionnaire"
- Pas de propagation d'erreur.

ECB

- le message à chiffrer est divisé en blocs à n bits
- chaque bloc est chiffré séparément
  - $\Longrightarrow$ deux blocs identiques $\Longrightarrow$ le même code
  - $\Longrightarrow$ permutation dans les blocs chiffrés $\Longrightarrow$ pareil pour les blocs en clair.
  - $\Longrightarrow$ padding aléatoire obligatoire
- Cryptanalyse : on cherche les séquences identiques $\Longrightarrow$ "dictionnaire"
- Pas de propagation d'erreur.

Exemple padding

- $n = 8$
- `Je_viens_lundi_soir`
- $\underbrace{12345678}_{\texttt{bloc 1}}\underbrace{12345678}_{\texttt{bloc 2}}\underbrace{123?????}_{\texttt{bloc 3}}$
- On peut utiliser
  - 0x80 quatre fois, et après 0x00
  - 0x00 et après 0x05
  - 0x00 cinq fois
- Recommandé par les normes PKCS(Public Key Cryptography Standards)

(bloc)	1	2	3	?	?	?	?	?
(texte)	o	i	r	5 qui manquent				
(en hexa)	6F	69	72	05	05	05	05	05

Exemple padding

- $n = 8$
- `Je_viens_lundi_soir`
- $\underbrace{12345678}_{\text{bloc 1}}\underbrace{12345678}_{\text{bloc 2}}\underbrace{123?????}_{\text{bloc 3}}$
- On peut utiliser
  - 0x80 quatre fois, et après 0x00
  - 0x00 et après 0x05
  - 0x00 cinq fois
- Recommandé par les normes PKCS(Public Key Cryptography Standards)

(bloc)	1	2	3	?	?	?	?	?
(texte)	o	i	r	5 qui manquent				
(en hexa)	6F	69	72	05	05	05	05	05

Exemple padding

- $n = 8$
- `Je_viens_lundi_soir`
- `12345678123456781233333`

  `bloc 1` `bloc 2` `bloc 3`
- On peut utiliser
  - 0x80 quatre fois, et après 0x00
  - 0x00 et après 0x05
  - 0x00 cinq fois
- Recommandé par les normes PKCS(Public Key Cryptography Standards)

(bloc)	1	2	3	?	?	?	?	?
(texte)	o	i	r	5 qui manquent				
(en hexa)	6F	69	72	05	05	05	05	05

Exemple padding

- $n = 8$
- `Je_viens_lundi_soir`
- $\underbrace{\texttt{12345678}}_{\text{bloc 1}}\underbrace{\texttt{12345678}}_{\text{bloc 2}}\underbrace{\texttt{123?????}}_{\text{bloc 3}}$
- On peut utiliser
  - 0x80 quatre fois, et après 0x00
  - 0x00 et après 0x05
  - 0x00 cinq fois
- Recommandé par les normes PKCS(Public Key Cryptography Standards)

(bloc)	1	2	3	?	?	?	?	?
(texte)	o	i	r	5 qui manquent				
(en hexa)	6F	69	72	05	05	05	05	05

Exemple padding

- $n = 8$
- `Je_viens_lundi_soir`
- $\underbrace{\texttt{12345678}}_{\text{bloc 1}}\underbrace{\texttt{12345678}}_{\text{bloc 2}}\underbrace{\texttt{123?????}}_{\text{bloc 3}}$
- On peut utiliser
  - 0x80 quatre fois, et après 0x00
  - 0x00 et après 0x05
  - 0x00 cinq fois
- Recommandé par les normes PKCS(Public Key Cryptography Standards)

(bloc)	1	2	3	?	?	?	?	?
(texte)	o	i	r	5 qui manquent				
(en hexa)	6F	69	72	05	05	05	05	05

Exemple padding

- $n = 8$
- `Je_viens_lundi_soir`
- $\underbrace{\texttt{12345678}}_{\text{bloc 1}}\underbrace{\texttt{12345678}}_{\text{bloc 2}}\underbrace{\texttt{123?????}}_{\text{bloc 3}}$
- On peut utiliser
  - 0x80 quatre fois, et après 0x00
  - 0x00 et après 0x05
  - 0x00 cinq fois
- Recommandé par les normes PKCS(Public Key Cryptography Standards)

(bloc)	1	2	3	?	?	?	?	?
(texte)	o	i	r	5 qui manquent				
(en hexa)	6F	69	72	05	05	05	05	05

Exemple padding

- $n = 8$
- `Je_viens_lundi_soir`
- `12345678123456781233333`

  `bloc 1` `bloc 2` `bloc 3`
- On peut utiliser
  - 0x80 quatre fois, et après 0x00
  - 0x00 et après 0x05
  - 0x00 cinq fois
- Recommandé par les normes PKCS(Public Key Cryptography Standards)

(bloc)	1	2	3	?	?	?	?	?
(texte)	o	i	r	5 qui manquent				
(en hexa)	6F	69	72	05	05	05	05	05

Exemple padding

- $n = 8$
- `Je_viens_lundi_soir`
- $\underbrace{\texttt{12345678}}_{\text{bloc 1}}\underbrace{\texttt{12345678}}_{\text{bloc 2}}\underbrace{\texttt{123?????}}_{\text{bloc 3}}$
- On peut utiliser
  - 0x80 quatre fois, et après 0x00
  - 0x00 et après 0x05
  - 0x00 cinq fois
- Recommandé par les normes PKCS(Public Key Cryptography Standards)

(bloc)	1	2	3	?	?	?	?	?
(texte)	o	i	r	5 qui manquent				
(en hexa)	6F	69	72	05	05	05	05	05

Remplissage (padding) aléatoire

ASCII : Offre $90000.00
(en hexa) : 4F666665722024393030302E3030
INPUT : 4F666665722024393030302E303001
OUTPUT : 33BEF550BADE4798DDA5C960E2C70EB9

ASCII : Offre $1000000.00
(en hexa) : 4F 66 66 65 72 2024313030303030302E3030
INPUT : 4F666665722024313030303030302E3030070707070707070707
OUTPUT : A4B8D1BF3020DB24CDD459BAB6A7BA7B02AC39EE7C1BF090
Si EVE connaît le but de la communication, elle peut déduire le nombre utilisant la longueur du chiffré
$\Longrightarrow$ On rajoute un nombre aléatoire d'octets comme "padding" et on l'indique dans le dernier octet rajouté $\Longrightarrow$ au plus 255 octets peuvent être rajoutés.

Remplissage (padding) aléatoire

ASCII : Offre $90000.00
(en hexa) : 4F666665722024393030302E3030
INPUT : 4F666665722024393030302E303001
OUTPUT : 33BEF550BADE4798DDA5C960E2C70EB9

ASCII : Offre $1000000.00
(en hexa) : 4F 66 66 65 72 202431303030302E3030
INPUT : 4F666665722024313030303030302E30300707070707070707
OUTPUT : A4B8D1BF3020DB24CDD459BAB6A7BA7B02AC39EE7C1BF090

Si EVE connaît le but de la communication, elle peut déduire le nombre utilisant la longueur du chiffré
$\Longrightarrow$ On rajoute un nombre aléatoire d'octets comme "padding" et on l'indique dans le dernier octet rajouté $\Longrightarrow$ au plus 255 octets peuvent être rajoutés.

Remplissage (padding) aléatoire

ASCII : Offre $90000.00
(en hexa) : 4F666665722024393030302E3030
INPUT : 4F666665722024393030302E303001
OUTPUT : 33BEF550BADE4798DDA5C960E2C70EB9

ASCII : Offre $1000000.00
(en hexa) : 4F 66 66 65 72 202431303030302E3030
INPUT : 4F666665722024313030303030302E30300707070707070707
OUTPUT : A4B8D1BF3020DB24CDD459BAB6A7BA7B02AC39EE7C1BF090
Si EVE connaît le but de la communication, elle peut déduire le nombre utilisant la longueur du chiffré
$\Longrightarrow$ On rajoute un nombre aléatoire d'octets comme "padding" et on l'indique dans le dernier octet rajouté $\Longrightarrow$ au plus 255 octets peuvent être rajoutés.

Remplissage (padding) aléatoire

ASCII : Offre $90000.00
(en hexa) : 4F666665722024393030302E3030
INPUT : 4F666665722024393030302E303001
OUTPUT : 33BEF550BADE4798DDA5C960E2C70EB9

ASCII : Offre $1000000.00
(en hexa) : 4F 66 66 65 72 2024313030303030302E3030
INPUT : 4F666665722024313030303030302E30300707070707070707
OUTPUT : A4B8D1BF3020DB24CDD459BAB6A7BA7B02AC39EE7C1BF090
Si EVE connaît le but de la communication, elle peut déduire le nombre utilisant la longueur du chiffré
$\Longrightarrow$ On rajoute un nombre aléatoire d'octets comme "padding" et on l'indique dans le dernier octet rajouté $\Longrightarrow$ au plus 255 octets peuvent être rajoutés.

Padding aléatoire

ASCII PT : Offer $90000.00
(en hexa) : 4F666665722024393030302E3030
INPUT :
4F666665722024393030302E303012441C0D5E2C60147DF54910B6A6445311
OUTPUT :
33BEF550BADE4798B164164E571A5266B0D488FAD934D6386494FAF528C8ED82

ASCII : Offre $1000000.00
(en hexa) : 4F666665722024313030303030302E3030
INPUT : 4F666665722024313030303030302E3030CEF8302A84BA07
OUTPUT :
A4B8D1BF3020DB24CDD459BAB6A7BA7BC01DF3FCC3B7DC1B

Cette convention particulière consistant à utiliser seulement le dernier octet nous limite à 255 octets aléatoires de remplissage

Padding aléatoire

ASCII PT : Offer $90000.00
(en hexa) : 4F666665722024393030302E3030
INPUT :
4F666665722024393030302E303012441C0D5E2C60147DF54910B6A6445311
OUTPUT :
33BEF550BADE4798B164164E571A5266B0D488FAD934D6386494FAF528C8ED82

ASCII : Offre $1000000.00
(en hexa) : 4F666665722024313030303030302E3030
INPUT : 4F666665722024313030303030302E3030CEF8302A84BA07
OUTPUT :
A4B8D1BF3020DB24CDD459BAB6A7BA7BC01DF3FCC3B7DC1B

Cette convention particulière consistant à utiliser seulement le dernier octet nous limite à 255 octets aléatoires de remplissage

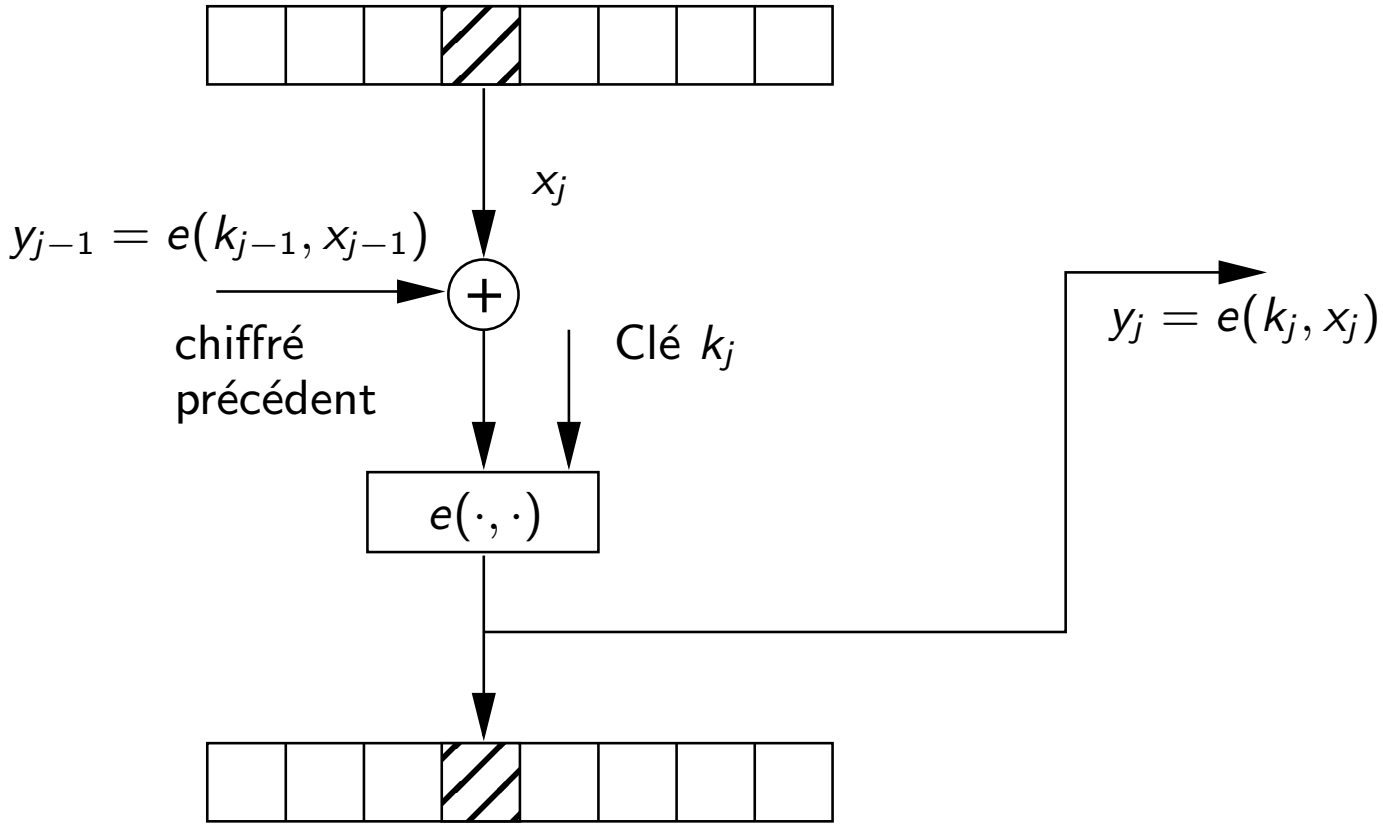
Padding aléatoire

ASCII PT : Offer $90000.00
(en hexa) : 4F666665722024393030302E3030
INPUT :
4F666665722024393030302E303012441C0D5E2C60147DF54910B6A6445311
OUTPUT :
33BEF550BADE4798B164164E571A5266B0D488FAD934D6386494FAF528C8ED82

ASCII : Offre $1000000.00
(en hexa) : 4F6666657220243130303030302E3030
INPUT : 4F6666657220243130303030302E3030CEF8302A84BA07
OUTPUT :
A4B8D1BF3020DB24CDD459BAB6A7BA7BC01DF3FCC3B7DC1B

Cette convention particulière consistant à utiliser seulement le dernier octet nous limite à 255 octets aléatoires de remplissage

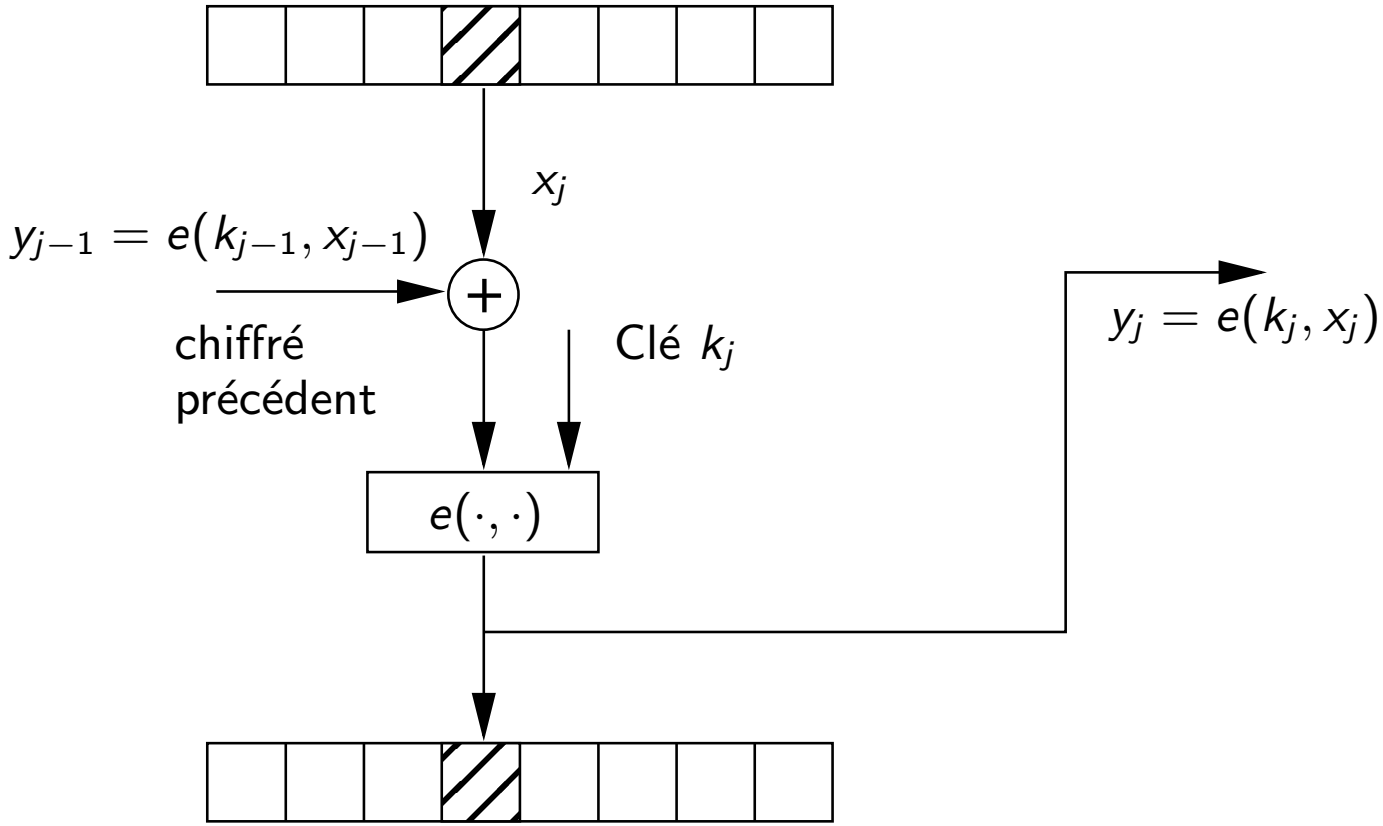
CBC



- $\rightarrow$ avant de chiffrer le bloc courant : un OU exclusif (XOR) avec le chiffrement du bloc précédent
- $\rightarrow$ pour x_0 on utilise un vecteur prédéfini

$$\begin{aligned} y_0 &= x_0 \oplus \mathrm{Vect} \\ y_j &= e(k_j, x_j \oplus y_{j-1}) \end{aligned}$$

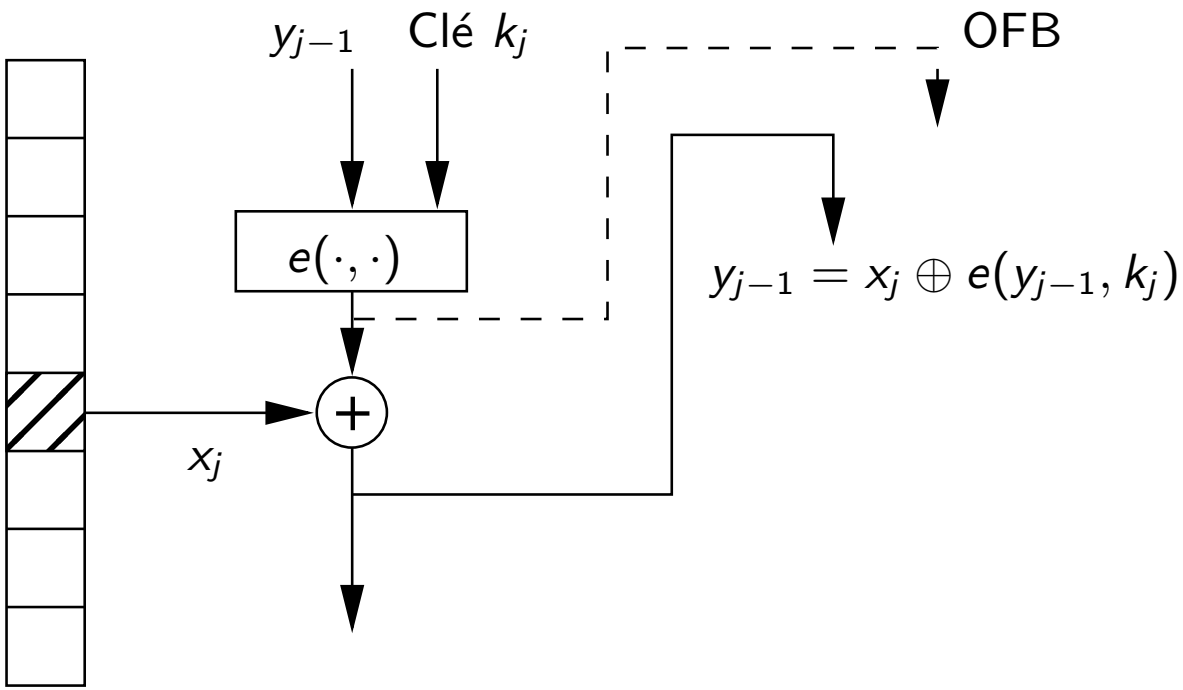
CBC



- $\rightarrow$ avant de chiffrer le bloc courant : un OU exclusif (XOR) avec le chiffrement du bloc précédent
- $\rightarrow$ pour x_0 on utilise un vecteur prédéfini

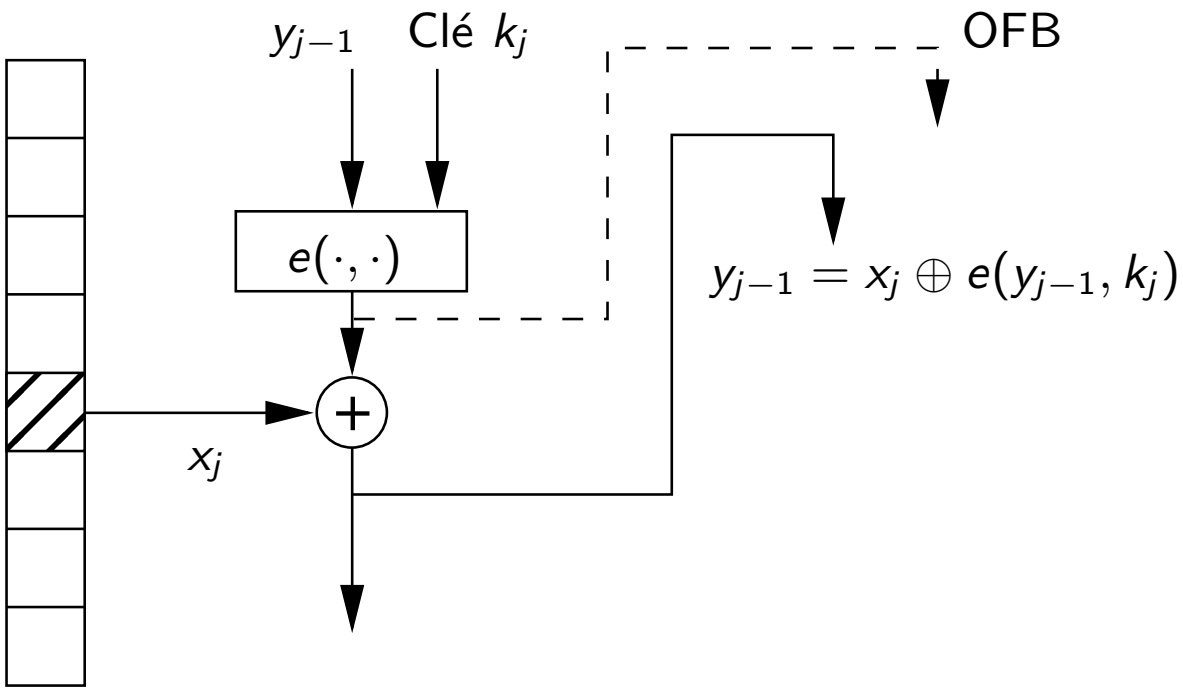
$$\begin{aligned} y_0 &= x_0 \oplus \mathrm{Vect} \\ y_j &= e(k_j, x_j \oplus y_{j-1}) \end{aligned}$$

CFB-OFB



- produit un flot de clés en chiffrant le précédent bloc chiffré
- OFB – une variante avec des problèmes de sécurité

CFB-OFB



- produit un flot de clés en chiffrant le précédent bloc chiffré
- OFB – une variante avec des problèmes de sécurité

Design d'un chiffrement par blocs

une fonction inversible obtenu à l'aide de plusieurs fonctions de non-inversibles : **réseau de Feistel**

- subdivisé en plusieurs rondes/tours/étages
- version équilibrée : les données sont divisées en deux parties de taille identique : DES, BlowFish
- version non-éuilibré : les données sont divisées en deux parties de taille différentes : MacGuffin, Skipjack
- chaque tour utilise une clé intermédiaire obtenu d'une clé principale : algorithme de génération de clés(key schedule)

Design d'un chiffrement par blocs

une fonction inversible obtenu à l'aide de plusieurs fonctions de non-inversibles : **réseau de Feistel**

- subdivisé en plusieurs rondes/tours/étages
- version équilibrée : les données sont divisées en deux parties de taille identique : DES, BlowFish
- version non-éuilibré : les données sont divisées en deux parties de taille différentes : MacGuffin, Skipjack
- chaque tour utilise une clé intermédiaire obtenu d'une clé principale : algorithme de génération de clés(key schedule)

Design d'un chiffrement par blocs

une fonction inversible obtenu à l'aide de plusieurs fonctions de non-inversibles : **réseau de Feistel**

- subdivisé en plusieurs rondes/tours/étages
- version équilibrée : les données sont divisées en deux parties de taille identique : DES, BlowFish
- version non-éuilibré : les données sont divisées en deux parties de taille différentes : MacGuffin, Skipjack
- chaque tour utilise une clé intermédiaire obtenu d'une clé principale : algorithme de génération de clés(key schedule)

Design d'un chiffrement par blocs

une fonction inversible obtenu à l'aide de plusieurs fonctions de non-inversibles : **réseau de Feistel**

- subdivisé en plusieurs rondes/tours/étages
- version équilibrée : les données sont divisées en deux parties de taille identique : DES, BlowFish
- version non-éuilibré : les données sont divisées en deux parties de taille différentes : MacGuffin, Skipjack
- chaque tour utilise une clé intermédiaire obtenu d'une clé principale : algorithme de génération de clés(key schedule)

Autres éléments de design d'un chiffrement par blocs

- **le bourrage ou padding** pour les chiffrement par blocs implémentés en ECB, CBC :
  - le texte en clair doit être un multiple de n=taille du bloc
  - compléter le dernier bloc avec des bits complémentaires
- **blanchiment de clé(key whitening)** : avant le premier tour et après le dernier tour on combine les données avec des parties de la clé (ex : avec XOR)
  - empêche le cryptanalyste d'obtenir une paire texte en clair, texte chiffré
  - combiner par OU exclusif une partie de la clé avec la sortie/entrée de l'algorithme
  - Variante : un chiffrement par blocs tweakable (tweak=pincer/changer par petits ajustements) a trois entrées : contrairement à la clé, le tweak n'est pas secret, mais il est aléatoire et dans certaines applications, il devrait changer de bloc à bloc

Autres éléments de design d'un chiffrement par blocs

- **le bourrage ou padding** pour les chiffrement par blocs implémentés en ECB, CBC :
  - le texte en clair doit être un multiple de n=taille du bloc
  - compléter le dernier bloc avec des bits complémentaires
- **blanchiment de clé(key whitening)** : avant le premier tour et après le dernier tour on combine les données avec des parties de la clé (ex : avec XOR)
  - empêche le cryptanalyste d'obtenir une paire texte en clair, texte chiffré
  - combiner par OU exclusif une partie de la clé avec la sortie/entrée de l'algorithme
  - Variante : un chiffrement par blocs tweakable (tweak=pincer/changer par petits ajustements) a trois entrées : contrairement à la clé, le tweak n'est pas secret, mais il est aléatoire et dans certaines applications, il devrait changer de bloc à bloc

Autres éléments de design d'un chiffrement par blocs

- **le bourrage ou padding** pour les chiffrement par blocs implémentés en ECB, CBC :
  - le texte en clair doit être un multiple de n=taille du bloc
  - compléter le dernier bloc avec des bits complémentaires
- **blanchiment de clé(key whitening)** : avant le premier tour et après le dernier tour on combine les données avec des parties de la clé (ex : avec XOR)
  - empêche le cryptanalyste d'obtenir une paire texte en clair, texte chiffré
  - combiner par OU exclusif une partie de la clé avec la sortie/entrée de l'algorithme
  - Variante : un chiffrement par blocs tweakable (tweak=pincer/changer par petits ajustements) a trois entrées : contrairement à la clé, le tweak n'est pas secret, mais il est aléatoire et dans certaines applications, il devrait changer de bloc à bloc

Autres éléments de design d'un chiffrement par blocs

- **le bourrage ou padding** pour les chiffrement par blocs implémentés en ECB, CBC :
  - le texte en clair doit être un multiple de n=taille du bloc
  - compléter le dernier bloc avec des bits complémentaires
- **blanchiment de clé(key whitening)** : avant le premier tour et après le dernier tour on combine les données avec des parties de la clé (ex : avec XOR)
  - empêche le cryptanalyste d'obtenir une paire texte en clair, texte chiffré
  - combiner par OU exclusif une partie de la clé avec la sortie/entrée de l'algorithme
  - Variante : un chiffrement par blocs tweakable (tweak=pincer/changer par petits ajustements) a trois entrées : contrairement à la clé, le tweak n'est pas secret, mais il est aléatoire et dans certaines applications, il devrait changer de bloc à bloc

Autres éléments de design d'un chiffrement par blocs

- **le bourrage ou padding** pour les chiffrement par blocs implémentés en ECB, CBC :
  - le texte en clair doit être un multiple de n=taille du bloc
  - compléter le dernier bloc avec des bits complémentaires
- **blanchiment de clé(key whitening)** : avant le premier tour et après le dernier tour on combine les données avec des parties de la clé (ex : avec XOR)
  - empêche le cryptanalyste d'obtenir une paire texte en clair, texte chiffré
  - combiner par OU exclusif une partie de la clé avec la sortie/entrée de l'algorithme
  - Variante : un chiffrement par blocs tweakable (tweak=pincer/changer par petits ajustements) a trois entrées : contrairement à la clé, le tweak n'est pas secret, mais il est aléatoire et dans certaines applications, il devrait changer de bloc à bloc

Autres éléments de design d'un chiffrement par blocs

- **le bourrage ou padding** pour les chiffrement par blocs implémentés en ECB, CBC :
  - le texte en clair doit être un multiple de n=taille du bloc
  - compléter le dernier bloc avec des bits complémentaires
- **blanchiment de clé(key whitening)** : avant le premier tour et après le dernier tour on combine les données avec des parties de la clé (ex : avec XOR)
  - empêche le cryptanalyste d'obtenir une paire texte en clair, texte chiffré
  - combiner par OU exclusif une partie de la clé avec la sortie/entrée de l'algorithme
  - Variante : un chiffrement par blocs tweakable (tweak=pincer/changer par petits ajustements) a trois entrées : contrairement à la clé, le tweak n'est pas secret, mais il est aléatoire et dans certaines applications, il devrait changer de bloc à bloc

Autres éléments de design d'un chiffrement par blocs

- **le bourrage ou padding** pour les chiffrement par blocs implémentés en ECB, CBC :
  - le texte en clair doit être un multiple de n=taille du bloc
  - compléter le dernier bloc avec des bits complémentaires
- **blanchiment de clé(key whitening)** : avant le premier tour et après le dernier tour on combine les données avec des parties de la clé (ex : avec XOR)
  - empêche le cryptanalyste d'obtenir une paire texte en clair, texte chiffré
  - combiner par OU exclusif une partie de la clé avec la sortie/entrée de l'algorithme
  - Variante : un chiffrement par blocs tweakable (tweak=pincer/changer par petits ajustements) a trois entrées : contrairement à la clé, le tweak n'est pas secret, mais il est aléatoire et dans certaines applications, il devrait changer de bloc à bloc

XOR masque jetable(one-time pad)

- table de vérité

A	B	$AXORB$
1	1	0
1	0	1
0	1	1
0	0	0

- On dit que le XOR masque le A avec B.
- Le symbole $\oplus$ représente l'application de l'opération XOR bit par bit.

XOR masque jetable(one-time pad)

- table de vérité

A	B	$AXORB$
1	1	0
1	0	1
0	1	1
0	0	0

- On dit que le XOR masque le A avec B.
- Le symbole $\oplus$ représente l'application de l'opération XOR bit par bit.

XOR masque jetable(one-time pad)

- table de vérité

A	B	$AXORB$
1	1	0
1	0	1
0	1	1
0	0	0

- On dit que le XOR masque le A avec B.
- Le symbole $\oplus$ représente l'application de l'opération XOR bit par bit.

Propriètés mathématiques de XOR

- $A \oplus A = 0$
- $A \oplus 0 = A$
- Associativité $A \oplus (B \oplus C) = (A \oplus B) \oplus C$
- $(A \oplus B) \oplus B = A$ (Conséquence des deux premières propriétés et de l'associativité).

Propriètés mathématiques de XOR

- $A \oplus A = 0$
- $A \oplus 0 = A$
- Associativité $A \oplus (B \oplus C) = (A \oplus B) \oplus C$
- $(A \oplus B) \oplus B = A$ (Conséquence des deux premières propriétés et de l'associativité).

Propriètés mathématiques de XOR

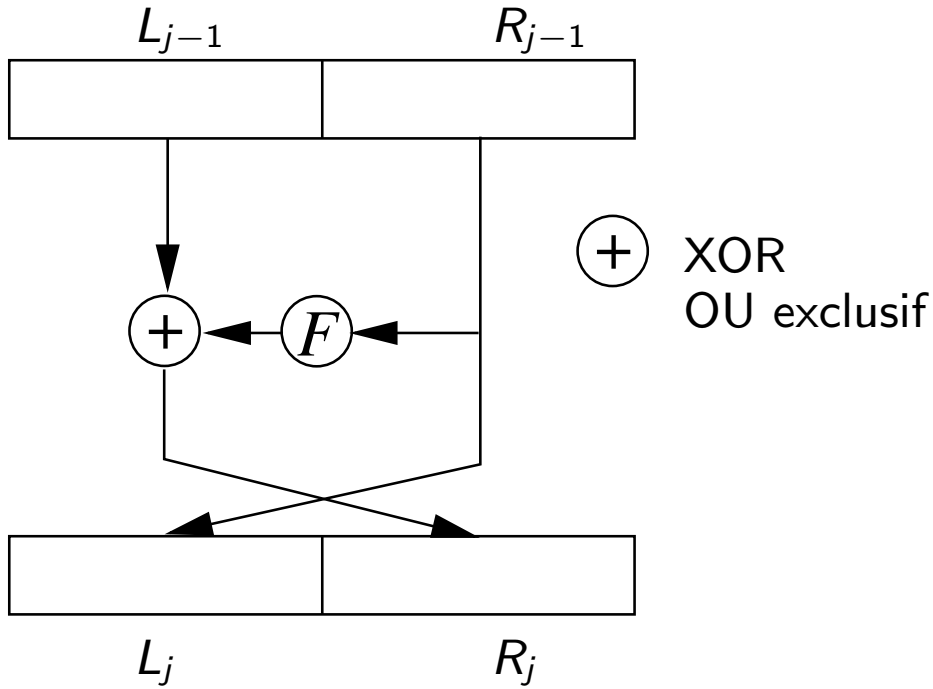
- $A \oplus A = 0$
- $A \oplus 0 = A$
- Associativité $A \oplus (B \oplus C) = (A \oplus B) \oplus C$
- $(A \oplus B) \oplus B = A$ (Conséquence des deux premières propriétés et de l'associativité).

Propriètés mathématiques de XOR

- $A \oplus A = 0$
- $A \oplus 0 = A$
- Associativité $A \oplus (B \oplus C) = (A \oplus B) \oplus C$
- $(A \oplus B) \oplus B = A$ (Conséquence des deux premières propriétés et de l'associativité).

Éléments de base

Fonction de Feistel

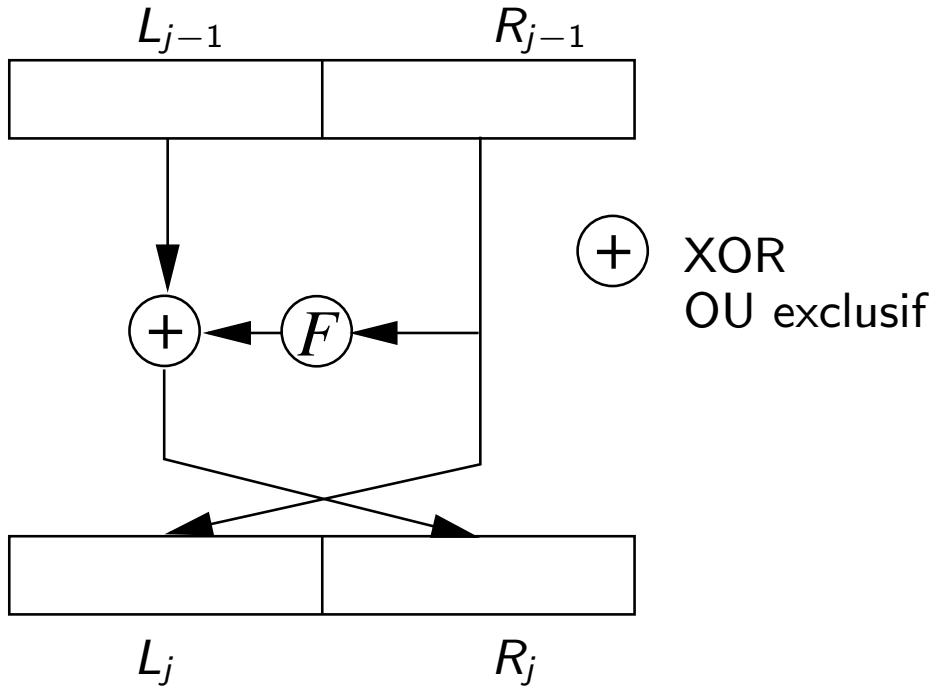


- une permutation. Pourqui ?
- Il y a une fonction inverse pour un tour. $G_j(L_j||R_j) = R_j \oplus F(L_j)||L_j$

$$\begin{aligned} G_j(F_j(L_{j-1}||R_{j-1})) &= G_j(R_{j-1}||L_{j-1} \oplus F(R_{j-1})) \\ &= (L_{j-1} \oplus F(R_{j-1})) \oplus F(R_{j-1})||R_{j-1} \\ &= L_{j-1}||R_{j-1} \end{aligned}$$

Éléments de base

Fonction de Feistel

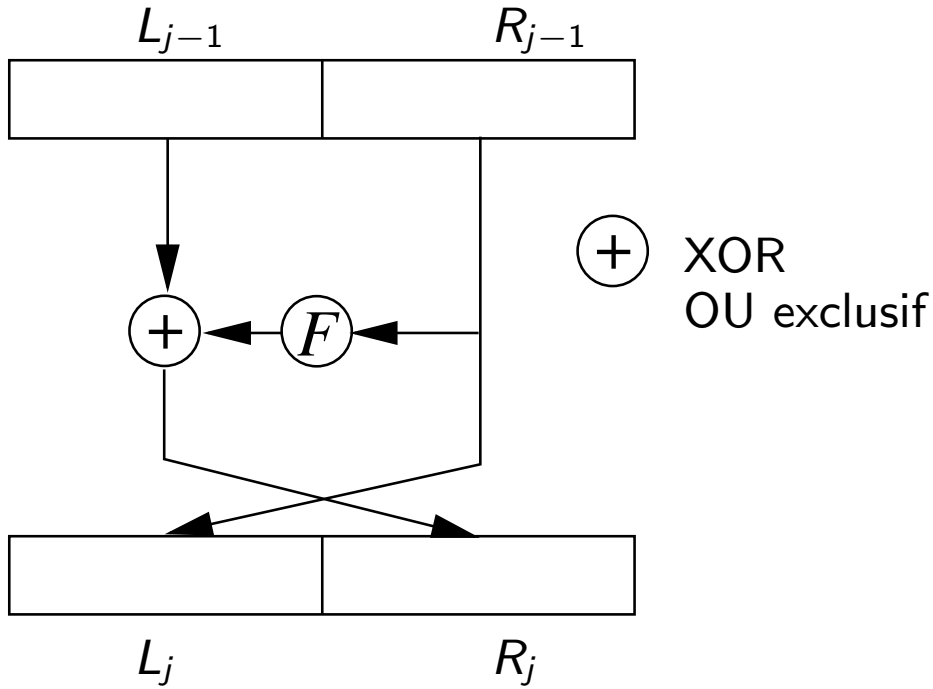


- une permutation. Pourqui ?
- Il y a une fonction inverse pour un tour. $G_j(L_j||R_j) = R_j \oplus F(L_j)||L_j$

$$\begin{aligned} G_j(F_j(L_{j-1}||R_{j-1})) &= G_j(R_{j-1}||L_{j-1} \oplus F(R_{j-1})) \\ &= (L_{j-1} \oplus F(R_{j-1})) \oplus F(R_{j-1})||R_{j-1} \\ &= L_{j-1}||R_{j-1} \end{aligned}$$

Éléments de base

Fonction de Feistel



- une permutation. Pourqui ?
- Il y a une fonction inverse pour un tour. $G_j(L_j||R_j) = R_j \oplus F(L_j)||L_j$

$$\begin{aligned} G_j(F_j(L_{j-1}||R_{j-1})) &= G_j(R_{j-1}||L_{j-1} \oplus F(R_{j-1})) \\ &= (L_{j-1} \oplus F(R_{j-1})) \oplus F(R_{j-1})||R_{j-1} \\ &= L_{j-1}||R_{j-1} \end{aligned}$$

Fonctions

- **Même en prenant F pas inversible** il y a une fonction inverse après d tours

$$G_1 \circ \cdots \circ G_d$$

- $\Longrightarrow$ **Après d tours, on peut utiliser le même circuit pour cryptage et décryptage en inversant seulement l'ordre des opérations**

Fonctions

- **Même en prenant F pas inversible** il y a une fonction inverse après d tours

$$G_1 \circ \cdots \circ G_d$$

- $\Longrightarrow$ **Après d tours, on peut utiliser le même circuit pour cryptage et décryptage en inversant seulement l'ordre des opérations**

Design de F. Réseau de permutation-substitution (SPN)

- permutation des bits via des tables P (P-Boxes)
- fonction non-linéaire (substitution) avec des tables S (S-Boxes)
- mixage linéaire en utilisant la fonction XOR
- application de la clé du tour (intégrée dans une fonction ou via un XOR)

Design de F. Réseau de permutation-substitution (SPN)

- permutation des bits via des tables P (P-Boxes)
- fonction non-linéaire (substitution) avec des tables S (S-Boxes)
- mixage linéaire en utilisant la fonction XOR
- application de la clé du tour (intégrée dans une fonction ou via un XOR)

Design de F. Réseau de permutation-substitution (SPN)

- permutation des bits via des tables P (P-Boxes)
- fonction non-linéaire (substitution) avec des tables S (S-Boxes)
- mixage linéaire en utilisant la fonction XOR
- application de la clé du tour (intégrée dans une fonction ou via un XOR)

Design de F. Réseau de permutation-substitution (SPN)

- permutation des bits via des tables P (P-Boxes)
- fonction non-linéaire (substitution) avec des tables S (S-Boxes)
- mixage linéaire en utilisant la fonction XOR
- application de la clé du tour (intégrée dans une fonction ou via un XOR)

Design de F. Le critère d'avalanche stricte

- Shannon : concept de diffusion ; Horst Feistel : le concept d'avalanche
- Le critère d'avalanche stricte (Strict Avalanche Criterion) Webster et Tavares en 1985 :
  - pour toute inversion d'un seul bit en entrée, alors chaque bit en sortie a une chance sur deux d'être modifié.
- l'uniformisation des sorties a pour but
  - d'empêcher les prédictions dues à une ligne stratégique
  - de rendre l'inversion de la fonction F plus difficile
- Le critère d'indépendance des bits
  - lorsque un seul bit i d'entrée est inversé, les bits de sortie j et k doivent changer de manière indépendante, pour tout j, k et i.

Design de F. Le critère d'avalanche stricte

- Shannon : concept de diffusion ; Horst Feistel : le concept d'avalanche
- Le critère d'avalanche stricte (Strict Avalanche Criterion) Webster et Tavares en 1985 :
  - pour toute inversion d'un seul bit en entrée alors chaque bit en sortie a une chance sur deux d'être modifié
- l'uniformisation des sorties a pour but
  - d'empêcher les prédictions dues à une ligne stratégique
  - de rendre l'inversion de la fonction F plus difficile
- Le critère d'indépendance des bits
  - lorsque un seul bit d'entrée est inversé, les bits de sortie doivent changer de manière indépendante pour toute fonction f

Design de F. Le critère d'avalanche stricte

- Shannon : concept de diffusion ; Horst Feistel : le concept d'avalanche
- Le critère d'avalanche stricte (Strict Avalanche Criterion) Webster et Tavares en 1985 :
  - pour toute inversion d'un seul bit en entrée alors chaque bit en sortie a une chance sur deux d'être modifié
- l'uniformisation des sorties a pour but
  - d'empêcher les prédictions dues à une ligne stratégique
  - de rendre l'inversion de la fonction F plus difficile
- Le critère d'indépendance des bits
  - lorsque un seul bit d'entrée est inversé, les bits de sortie doivent changer de manière indépendante pour toute fonction f

Design de F. Le critère d'avalanche stricte

- Shannon : concept de diffusion ; Horst Feistel : le concept d'avalanche
- Le critère d'avalanche stricte (Strict Avalanche Criterion) Webster et Tavares en 1985 :
  - pour toute inversion d'un seul bit en entrée alors chaque bit en sortie a une chance sur deux d'être modifié
- l'uniformisation des sorties a pour but
  - d'empêcher les prédictions dues à une trop grande régularité
  - de rendre l'inversion de la fonction F plus difficile
- Le critère d'indépendance des bits
  - lorsque un seul bit d'entrée est inversé, les bits de sortie doivent changer de manière indépendante pour toutes les clés K

Design de F. Le critère d'avalanche stricte

- Shannon : concept de diffusion ; Horst Feistel : le concept d'avalanche
- Le critère d'avalanche stricte (Strict Avalanche Criterion) Webster et Tavares en 1985 :
  - pour toute inversion d'un seul bit en entrée alors chaque bit en sortie a une chance sur deux d'être modifié
- l'uniformisation des sorties a pour but
  - d'empêcher les prédictions dues à une trop grande régularité
  - de rendre l'inversion de la fonction F plus difficile
- Le critère d'indépendance des bits
  - lorsque un seul bit d'entrée est inversé, les bits de sortie doivent changer de manière indépendante pour toutes les clés K

Design de F. Le critère d'avalanche stricte

- Shannon : concept de diffusion ; Horst Feistel : le concept d'avalanche
- Le critère d'avalanche stricte (Strict Avalanche Criterion) Webster et Tavares en 1985 :
  - pour toute inversion d'un seul bit en entrée alors chaque bit en sortie a une chance sur deux d'être modifié
- l'uniformisation des sorties a pour but
  - d'empêcher les prédictions dues à un biais statistique.
  - de rendre l'inversion de la fonction F plus difficile
- Le critère d'indépendance des bits
  - lorsque un seul bit d'entrée est inversé, les bits de sortie ne doivent pas changer de manière prévisible pour tous les cas.

Design de F. Le critère d'avalanche stricte

- Shannon : concept de diffusion ; Horst Feistel : le concept d'avalanche
- Le critère d'avalanche stricte (Strict Avalanche Criterion) Webster et Tavares en 1985 :
  - pour toute inversion d'un seul bit en entrée alors chaque bit en sortie a une chance sur deux d'être modifié
- l'uniformisation des sorties a pour but
  - d'empêcher les prédictions dues à un biais statistique.
  - de rendre l'inversion de la fonction F plus difficile
- Le critère d'indépendance des bits
  - lorsque un seul bit d'entrée est inversé, les bits de sortie ne doivent pas changer de manière prévisible pour tous les cas.

Design de F. Le critère d'avalanche stricte

- Shannon : concept de diffusion ; Horst Feistel : le concept d'avalanche
- Le critère d'avalanche stricte (Strict Avalanche Criterion) Webster et Tavares en 1985 :
  - pour toute inversion d'un seul bit en entrée alors chaque bit en sortie a une chance sur deux d'être modifié
- l'uniformisation des sorties a pour but
  - d'empêcher les prédictions dues à un biais statistique.
  - de rendre l'inversion de la fonction F plus difficile
- Le critère d'indépendance des bits
  - lorsque un seul bit d'entrée est inversé, les bits de sortie ne doivent changer de manière indépendante pour tout $f_i \in F$.

Design de F. Le critère d'avalanche stricte

- Shannon : concept de diffusion ; Horst Feistel : le concept d'avalanche
- Le critère d'avalanche stricte (Strict Avalanche Criterion) Webster et Tavares en 1985 :
  - pour toute inversion d'un seul bit en entrée alors chaque bit en sortie a une chance sur deux d'être modifié
- l'uniformisation des sorties a pour but
  - d'empêcher les prédictions dues à un biais statistique.
  - de rendre l'inversion de la fonction F plus difficile
- Le critère d'indépendance des bits
  - lorsque un seul bit d'entrée est inversé, les bits de sortie j et k doivent changer de manière indépendante pour tout j et k.

Design de F. Le critère d'avalanche stricte

- Shannon : concept de diffusion ; Horst Feistel : le concept d'avalanche
- Le critère d'avalanche stricte (Strict Avalanche Criterion) Webster et Tavares en 1985 :
  - pour toute inversion d'un seul bit en entrée alors chaque bit en sortie a une chance sur deux d'être modifié
- l'uniformisation des sorties a pour but
  - d'empêcher les prédictions dues à un biais statistique.
  - de rendre l'inversion de la fonction F plus difficile
- Le critère d'indépendance des bits
  - lorsque un seul un bit i d'entrée est inversé, les bits de sortie j et k devrait changer de manière autonome, pour tout i, j et k.

Design de F. Le critère d'avalanche stricte

- Shannon : concept de diffusion ; Horst Feistel : le concept d'avalanche
- Le critère d'avalanche stricte (Strict Avalanche Criterion) Webster et Tavares en 1985 :
  - pour toute inversion d'un seul bit en entrée alors chaque bit en sortie a une chance sur deux d'être modifié
- l'uniformisation des sorties a pour but
  - d'empêcher les prédictions dues à un biais statistique.
  - de rendre l'inversion de la fonction F plus difficile
- Le critère d'indépendance des bits
  - lorsque un seul un bit i d'entrée est inversé, les bits de sortie j et k devrait changer de manière autonome, pour tout i, j et k.

Design de F. Le critère d'avalanche stricte

- Shannon : concept de diffusion ; Horst Feistel : le concept d'avalanche
- Le critère d'avalanche stricte (Strict Avalanche Criterion) Webster et Tavares en 1985 :
  - pour toute inversion d'un seul bit en entrée alors chaque bit en sortie a une chance sur deux d'être modifié
- l'uniformisation des sorties a pour but
  - d'empêcher les prédictions dues à un biais statistique.
  - de rendre l'inversion de la fonction F plus difficile
- Le critère d'indépendance des bits
  - lorsque un seul un bit i d'entrée est inversé, les bits de sortie j et k devrait changer de manière autonome, pour tout i, j et k.

Design de F. Opérations non linéaires.

- Un algorithme de chiffrement avec toutes les opérations linéaires : le chiffre est réduit à un système d'équations linéaires $\Rightarrow$ **une attaque algébrique**
  - le texte comme un vecteur de bits et on utilise l'algèbre de Boole
  - le texte comme un vecteur d'octets et on utilise l'arithmétique modulo 2^8
- Pas suffisament de non-linéarité dans algorithme de chiffrement $\Rightarrow$ une approximation "assez" linéaire pour la fonction de tour
- Cryptanalyse linéaire : avec suffisamment de paires connues clair-chiffréon peut casser le chiffrement.

Design de F. Opérations non linéaires.

- Un algorithme de chiffrement avec toutes les opérations linéaires : le chiffre est réduit à un système d'équations linéaires $\Rightarrow$ **une attaque algébrique**
  - le texte comme un vecteur de bits et on utilise l'algèbre de Boole
  - le texte comme un vecteur d'octets et on utilise l'arithmétique modulo 2^8
- Pas suffisament de non-linéarité dans algorithme de chiffrement $\Rightarrow$ une approximation "assez" linéaire pour la fonction de tour
- Cryptanalyse linéaire : avec suffisamment de paires connues clair-chiffréon peut casser le chiffrement.

Design de F. Opérations non linéaires.

- **Utilisation des tables S (S-box) $m \times n$ de substitution**
  - prend m bits et rend n bits, $m < n$
  - complique la relation entre la clé et le chiffré
  - $\Longrightarrow$ propriété de confusion (Shannon) avalanche
  - $\Longrightarrow$ protection contre la cryptanalyse différentielle

Design de F. Opérations non linéaires.

- **Utilisation des tables S (S-box) $m \times n$ de substitution**
  - prend m bits et rend n bits, $m < n$
  - complique la relation entre la clé et le chiffré
  - $\Longrightarrow$ propriété de confusion (Shannon) avalanche
  - $\Longrightarrow$ protection contre la cryptanalyse différentielle

Design de F. Opérations non linéaires.

- **Utilisation des tables S (S-box) $m \times n$ de substitution**
  - prend m bits et rend n bits, $m < n$
  - complique la relation entre la clé et le chiffré
  - $\Longrightarrow$ propriété de confusion (Shannon) avalanche
  - $\Longrightarrow$ protection contre la cryptanalyse différentielle

Design de F. Opérations non linéaires.

- **Utilisation des tables S (S-box) $m \times n$ de substitution**
  - prend m bits et rend n bits, $m < n$
  - complique la relation entre la clé et le chiffré
  - $\Longrightarrow$ propriété de confusion (Shannon) avalanche
  - $\Longrightarrow$ protection contre la cryptanalyse différentielle

Design de F. Opérations non linéaires.

- **Utilisation des tables S (S-box) $m \times n$ de substitution**
  - prend m bits et rend n bits, $m < n$
  - complique la relation entre la clé et le chiffré
  - $\Longrightarrow$ propriété de confusion (Shannon) avalanche
  - $\Longrightarrow$ protection contre la cryptanalyse différentielle

Élements de base

Bits d'entrée :

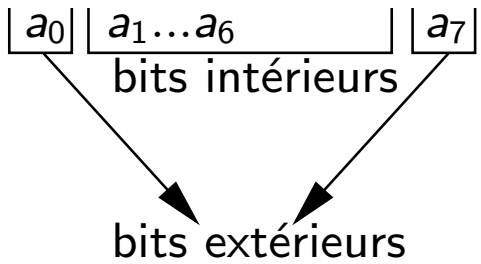


Table S

x_n		$a_1 \ldots a_6$
$a_0 a_7$		$c_1 c_2 \ldots c_n$

Élements de base

Bits d'entrée :

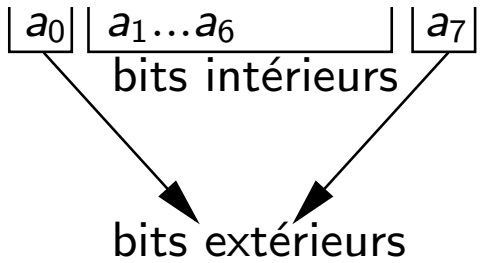


Table S

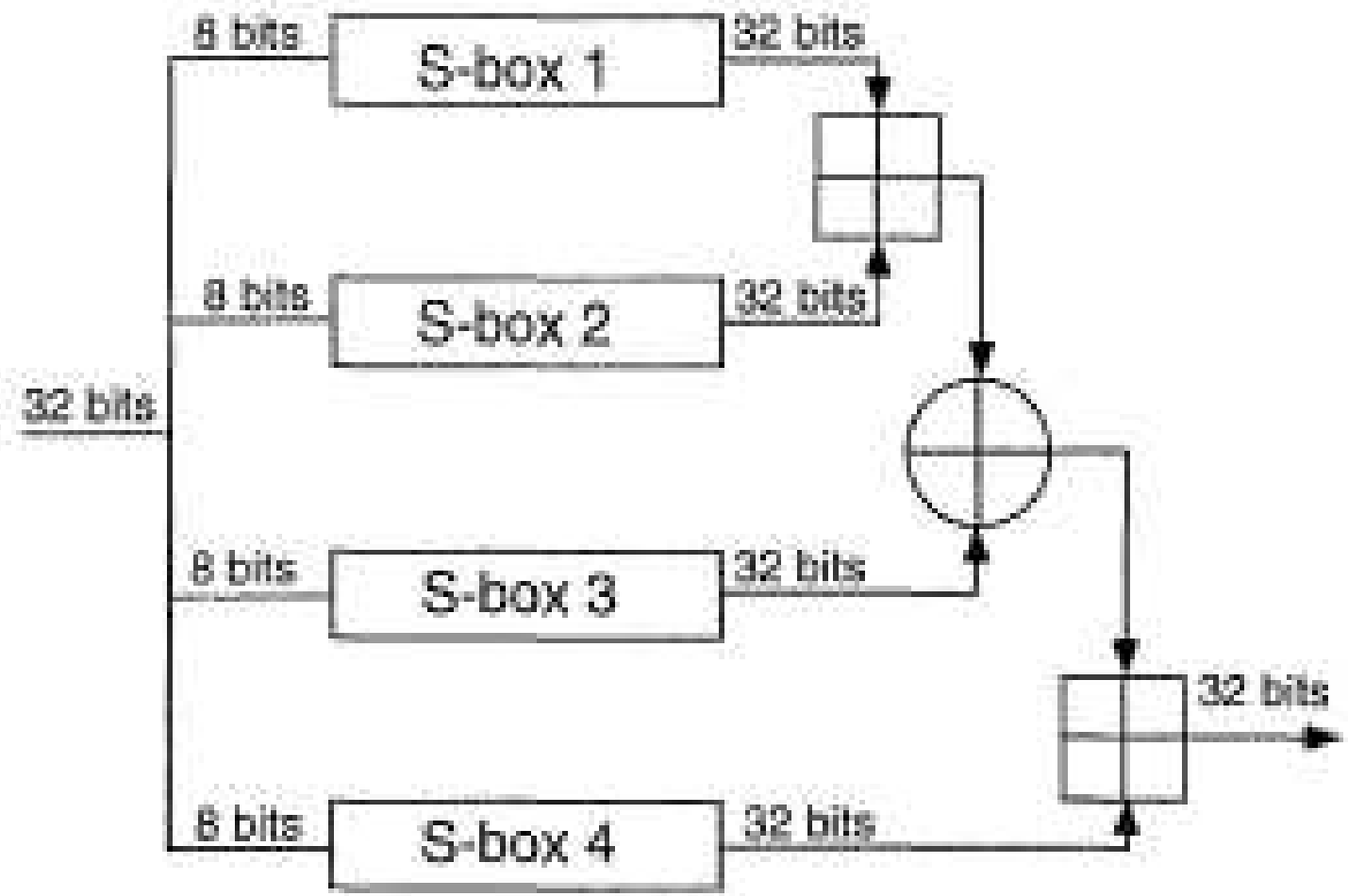
x_n		$a_1 \ldots a_6$
$a_0 a_7$		$c_1 c_2 \ldots c_n$

Design de F. Plus d'opérations non linéaires.

Une méthode consiste à mélanger des opérations de différentes algèbres. Dans la fonction F de Blowfish on combine les quatre mots de 32 bits en un seul.

- ni $x = a + b + c + d$,
- ni les opérations booléennes $x = a \oplus b \oplus c \oplus d$
- le mélange, $x = ((a + b) \oplus c) + d$

La function F de BlowFish



Encodage Blowfish

- Utilisation de deux opérations primitives :
  1. addition modulo 2^{32}
  2. XOR bit-à-bit
- $\Longrightarrow$ Ces deux opérations ne commutent pas
- $\Longrightarrow$ Rendent la cryptanalyse difficile

Encodage Blowfish

- Utilisation de deux opérations primitives :
  1. addition modulo 2^{32}
  2. XOR bit-à-bit
- $\Longrightarrow$ Ces deux opérations ne commutent pas
- $\Longrightarrow$ Rendent la cryptanalyse difficile

Encodage Blowfish

- Utilisation de deux opérations primitives :
  1. addition modulo 2^{32}
  2. XOR bit-à-bit
- $\Longrightarrow$ Ces deux opérations ne commutent pas
- $\Longrightarrow$ Rendent la cryptanalyse difficile

Encodage Blowfish

- Utilisation de deux opérations primitives :
  1. addition modulo 2^{32}
  2. XOR bit-à-bit
- $\Longrightarrow$ Ces deux opérations ne commutent pas
- $\Longrightarrow$ Rendent la cryptanalyse difficile

Encodage Blowfish

- Utilisation de deux opérations primitives :
  1. addition modulo 2^{32}
  2. XOR bit-à-bit
- $\Longrightarrow$ Ces deux opérations ne commutent pas
- $\Longrightarrow$ Rendent la cryptanalyse difficile

Design de F. Autres méthodes.

- IDEA : utilise une multiplication modulo 2^{16}
- AES : multiplications avec des polynômes dans un corps de Galois.
- les rotations sur les mots ou registres ne sont pas linéaires dans l'algèbre normale
  - CAST-128 et CAST-256 utilisent une rotation dépendante de la clé de la fonction F
  - RC5, RC6 et MARS utilisent des rotation dépendantes des données

Design de F. Autres méthodes.

- IDEA : utilise une multiplication modulo 2^{16}
- AES : multiplications avec des polynômes dans un corps de Galois.
- les rotations sur les mots ou registres ne sont pas linéaires dans l'algèbre normale
  - CAST-128 et CAST-256 utilisent une rotation dépendante de la clé de la fonction F
  - RC5, RC6 et MARS utilisent des rotation dépendantes des données

Design de F. Autres méthodes.

- IDEA : utilise une multiplication modulo 2^{16}
- AES : multiplications avec des polynômes dans un corps de Galois.
- les rotations sur les mots ou registres ne sont pas linéaires dans l'algèbre normale
  - CAST-128 et CAST-256 utilisent une rotation dépendante de la clé de la fonction F
  - RC5, RC6 et MARS utilisent des rotation dépendantes des données

Design de F. Autres méthodes.

- IDEA : utilise une multiplication modulo 2^{16}
- AES : multiplications avec des polynômes dans un corps de Galois.
- les rotations sur les mots ou registres ne sont pas linéaires dans l'algèbre normale
  - CAST-128 et CAST-256 utilisent une rotation dépendante de la clé de la fonction F
  - RC5, RC6 et MARS utilisent des rotation dépendantes des données

Design de F. Autres méthodes.

- IDEA : utilise une multiplication modulo 2^{16}
- AES : multiplications avec des polynômes dans un corps de Galois.
- les rotations sur les mots ou registres ne sont pas linéaires dans l'algèbre normale
  - CAST-128 et CAST-256 utilisent une rotation dépendante de la clé de la fonction F
  - RC5, RC6 et MARS utilisent des rotation dépendantes des données

Design de F. Autres méthodes.

- IDEA : utilise une multiplication modulo 2^{16}
- AES : multiplications avec des polynômes dans un corps de Galois.
- les rotations sur les mots ou registres ne sont pas linéaires dans l'algèbre normale
  - CAST-128 et CAST-256 utilisent une rotation dépendante de la clé de la fonction F
  - RC5, RC6 et MARS utilisent des rotation dépendantes des données

Blowfish

- Utilisé dans `ssh`
- Simple
  - un réseau de Feistel
  - opérations simples
    - addition
    - ou exclusif
    - recherche dans des tableaux avec des paramètres de 32 bits
- Rapide, compact si les clés ne changent pas
- $\Longrightarrow$ pas adapté aux applications avec beaucoup de changement de clé

Blowfish

- Utilisé dans `ssh`
- Simple
  - un réseau de Feistel
  - opérations simples
    - addition
    - ou exclusif
    - recherche dans des tableaux avec des paramètres de 32 bits
- Rapide, compact si les clés ne changent pas
- $\Longrightarrow$ pas adapté aux applications avec beaucoup de changement de clé

Blowfish

- Utilisé dans `ssh`
- Simple
  - un réseau de Feistel
  - opérations simples
    - addition
    - ou exclusif
    - recherche dans des tableaux avec des paramètres de 32 bits
- Rapide, compact si les clés ne changent pas
- $\Longrightarrow$ pas adapté aux applications avec beaucoup de changement de clé

Blowfish

- Utilisé dans `ssh`
- Simple
  - un réseau de Feistel
  - opérations simples
    - addition
    - ou exclusif
    - recherche dans des tableaux avec des paramètres de 32 bits
- Rapide, compact si les clés ne changent pas
- $\Longrightarrow$ pas adapté aux applications avec beaucoup de changement de clé

Blowfish

- Utilisé dans `ssh`
- Simple
  - un réseau de Feistel
  - opérations simples
    - addition
    - ou exclusif
    - recherche dans des tableaux avec des paramètres de 32 bits
- Rapide, compact si les clés ne changent pas
- $\Longrightarrow$ pas adapté aux applications avec beaucoup de changement de clé

Blowfish

- Utilisé dans `ssh`
- Simple
  - un réseau de Feistel
  - opérations simples
    - addition
    - ou exclusif
    - recherche dans des tableaux avec des paramètres de 32 bits
- Rapide, compact si les clés ne changent pas
- $\Longrightarrow$ pas adapté aux applications avec beaucoup de changement de clé

Blowfish

- Utilisé dans `ssh`
- Simple
  - un réseau de Feistel
  - opérations simples
    - addition
    - ou exclusif
    - recherche dans des tableaux avec des paramètres de 32 bits
- Rapide, compact si les clés ne changent pas
- $\Longrightarrow$ pas adapté aux applications avec beaucoup de changement de clé

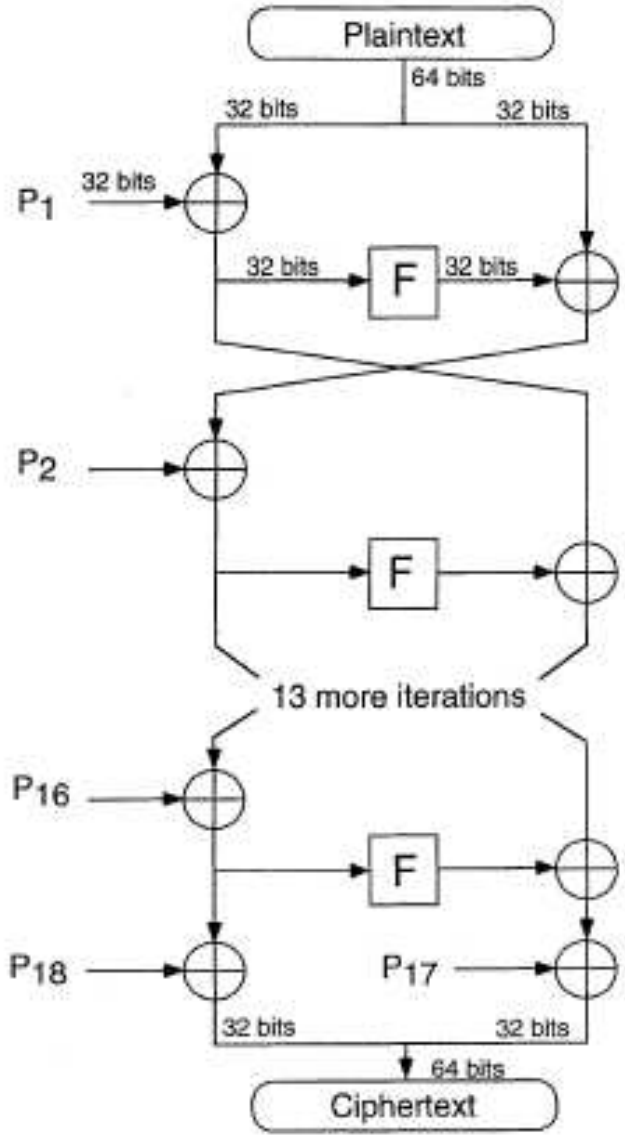
Blowfish

- Utilisé dans `ssh`
- Simple
  - un réseau de Feistel
  - opérations simples
    - addition
    - ou exclusif
    - recherche dans des tableaux avec des paramètres de 32 bits
- Rapide, compact si les clés ne changent pas
- $\Longrightarrow$ pas adapté aux applications avec beaucoup de changement de clé

Blowfish

- Utilisé dans `ssh`
- Simple
  - un réseau de Feistel
  - opérations simples
    - addition
    - ou exclusif
    - recherche dans des tableaux avec des paramètres de 32 bits
- Rapide, compact si les clés ne changent pas
- $\Longrightarrow$ pas adapté aux applications avec beaucoup de changement de clé

Schéma


Plaintext
64 bits
32 bits
32 bits
P1
32 bits
32 bits
F
32 bits
P2
F
13 more iterations
P16
F
P18
P17
32 bits
32 bits
64 bits
Ciphertext

Cryptage Blowfish

- Entrées :
  - F, $M_1,\ldots,M_{18}$ sous-clés
  - texte en clair sur 64 bits
- séparer x en deux moitiés x_L, x_R
- pour chaque i de 1 à 16 faire
  - $x_L = x_L \oplus M_i$
  - $x_R = F(x_L) \oplus x_R$
  - échanger x_L et x_R
- échanger x_L et x_R (i.e. annuler le dernier échange)
- $x_R = x_R \oplus M_{17}$
- $x_L = x_L \oplus M_{18}$

Cryptage Blowfish

- Entrées :
  - F, $M_1,\ldots,M_{18}$ sous-clés
  - texte en clair sur 64 bits
- séparer x en deux moitiés x_L, x_R
- pour chaque i de 1 à 16 faire
  - $x_L = x_L \oplus M_i$
  - $x_R = F(x_L) \oplus x_R$
  - échanger x_L et x_R
- échanger x_L et x_R (i.e. annuler le dernier échange)
- $x_R = x_R \oplus M_{17}$
- $x_L = x_L \oplus M_{18}$

Cryptage Blowfish

- Entrées :
  - F, $M_1, \ldots, M_{18}$ sous-clés
  - texte en clair sur 64 bits
- séparer x en deux moitiés x_L, x_R
- pour chaque i de 1 à 16 faire
  - $x_L = x_L \oplus M_i$
  - $x_R = F(x_L) \oplus x_R$
  - échanger x_L et x_R
- échanger x_L et x_R (i.e. annuler le dernier échange)
- $x_R = x_R \oplus M_{17}$
- $x_L = x_L \oplus M_{18}$

Cryptage Blowfish

- Entrées :
  - F, $M_1,\ldots,M_{18}$ sous-clés
  - texte en clair sur 64 bits
- séparer x en deux moitiés x_L, x_R
- pour chaque i de 1 à 16 faire
  - $x_L = x_L \oplus M_i$
  - $x_R = F(x_L) \oplus x_R$
  - échanger x_L et x_R
- échanger x_L et x_R (i.e. annuler le dernier échange)
- $x_R = x_R \oplus M_{17}$
- $x_L = x_L \oplus M_{18}$

Cryptage Blowfish

- Entrées :
  - F, $M_1,\ldots,M_{18}$ sous-clés
  - texte en clair sur 64 bits
- séparer x en deux moitiés x_L, x_R
- pour chaque i de 1 à 16 faire
  - $x_L = x_L \oplus M_i$
  - $x_R = F(x_L) \oplus x_R$
  - échanger x_L et x_R
- échanger x_L et x_R (i.e. annuler le dernier échange)
- $x_R = x_R \oplus M_{17}$
- $x_L = x_L \oplus M_{18}$

Cryptage Blowfish

- Entrées :
  - F, $M_1,\ldots,M_{18}$ sous-clés
  - texte en clair sur 64 bits
- séparer x en deux moitiés x_L, x_R
- pour chaque i de 1 à 16 faire
  - $x_L = x_L \oplus M_i$
  - $x_R = F(x_L) \oplus x_R$
  - échanger x_L et x_R
- échanger x_L et x_R (i.e. annuler le dernier échange)
- $x_R = x_R \oplus M_{17}$
- $x_L = x_L \oplus M_{18}$

Cryptage Blowfish

- Entrées :
  - F, $M_1,\ldots,M_{18}$ sous-clés
  - texte en clair sur 64 bits
- séparer x en deux moitiés x_L, x_R
- pour chaque i de 1 à 16 faire
  - $x_L = x_L \oplus M_i$
  - $x_R = F(x_L) \oplus x_R$
  - échanger x_L et x_R
- échanger x_L et x_R (i.e. annuler le dernier échange)
- $x_R = x_R \oplus M_{17}$
- $x_L = x_L \oplus M_{18}$

Cryptage Blowfish

- Entrées :
  - F, $M_1,\ldots,M_{18}$ sous-clés
  - texte en clair sur 64 bits
- séparer x en deux moitiés x_L, x_R
- pour chaque i de 1 à 16 faire
  - $x_L = x_L \oplus M_i$
  - $x_R = F(x_L) \oplus x_R$
  - échanger x_L et x_R
- échanger x_L et x_R (i.e. annuler le dernier échange)
- $x_R = x_R \oplus M_{17}$
- $x_L = x_L \oplus M_{18}$

Cryptage Blowfish

- Entrées :
  - F, $M_1,\ldots,M_{18}$ sous-clés
  - texte en clair sur 64 bits
- séparer x en deux moitiés x_L, x_R
- pour chaque i de 1 à 16 faire
  - $x_L = x_L \oplus M_i$
  - $x_R = F(x_L) \oplus x_R$
  - échanger x_L et x_R
- échanger x_L et x_R (i.e. annuler le dernier échange)
- $x_R = x_R \oplus M_{17}$
- $x_L = x_L \oplus M_{18}$

Cryptage Blowfish

- Entrées :
  - F, $M_1,\ldots,M_{18}$ sous-clés
  - texte en clair sur 64 bits
- séparer x en deux moitiés x_L, x_R
- pour chaque i de 1 à 16 faire
  - $x_L = x_L \oplus M_i$
  - $x_R = F(x_L) \oplus x_R$
  - échanger x_L et x_R
- échanger x_L et x_R (i.e. annuler le dernier échange)
- $x_R = x_R \oplus M_{17}$
- $x_L = x_L \oplus M_{18}$

Cryptage Blowfish

- Entrées :
  - F, $M_1,\ldots,M_{18}$ sous-clés
  - texte en clair sur 64 bits
- séparer x en deux moitiés x_L, x_R
- pour chaque i de 1 à 16 faire
  - $x_L = x_L \oplus M_i$
  - $x_R = F(x_L) \oplus x_R$
  - échanger x_L et x_R
- échanger x_L et x_R (i.e. annuler le dernier échange)
- $x_R = x_R \oplus M_{17}$
- $x_L = x_L \oplus M_{18}$

Cryptage Blowfish

- Fonction F (de mélange et hachage)
  - Diviser x_L en quatre parts de huit bits : a, b, c, d ; générer les tables S

$$F(x_L) = (S_{1,a} + S_{2,b} \bmod 2^{32}) \oplus S_{3,c} + S_{4,d} \bmod 2^{32}$$

Cryptage Blowfish

- Fonction F (de mélange et hachage)
  - Diviser x_L en quatre parts de huit bits : a, b, c, d ; générer les tables S

$$F(x_L) = (S_{1,a} + S_{2,b} \bmod 2^{32}) \oplus S_{3,c} + \\ + S_{4,d} \bmod 2^{32}$$

Détails Blowfish

- Les tables S : $S_{1,a}$, $S_{2,b}$, $S_{3,c}$, $S_{4,d}$
- Les clés $P_1,\ldots,P_{18}$ initialisés avec une chaîne de caractères fixée : les décimales de π

Détails Blowfish

- Les tables S : $S_{1,a}$, $S_{2,b}$, $S_{3,c}$, $S_{4,d}$
- Les clés $P_1,\ldots,P_{18}$ initialisés avec une chaîne de caractères fixée : les décimales de π

Détails Blowfish

- Les tables S : $S_{1,a}$, $S_{2,b}$, $S_{3,c}$, $S_{4,d}$
- Les clés $P_1,\ldots,P_{18}$ initialisés avec une chaîne de caractères fixée : les décimales de π

Attaques – Cryptographie différentielle (Vaudenay)

- avec $2^{8 \cdot \text{nbre rondes}+1}$ textes en clair choisis
- clé faible dans une table S :
  - il y a une collision : $S(a) = S(b)$ pour $a \neq b$
  - $\Longrightarrow 2^{4 \cdot \text{nbre rondes}+1}$ textes en clair choisis
- Il n'y a pas moyen de vérifier si une clé est faible avant l'expansion

Attaques – Cryptographie différentielle (Vaudenay)

- avec $2^{8 \cdot \text{nbre rondes}+1}$ textes en clair choisis
- clé faible dans une table S :
  - il y a une collision : $S(a) = S(b)$ pour $a \neq b$
  - $\Longrightarrow 2^{4 \cdot \text{nbre rondes}+1}$ textes en clair choisis
- Il n'y a pas moyen de vérifier si une clé est faible avant l'expansion

Attaques – Cryptographie différentielle (Vaudenay)

- avec $2^{8 \cdot \text{nbre rondes}+1}$ textes en clair choisis
- clé faible dans une table S :
  - il y a une collision : $S(a) = S(b)$ pour $a \neq b$
  - $\Longrightarrow 2^{4 \cdot \text{nbre rondes}+1}$ textes en clair choisis
- Il n'y a pas moyen de vérifier si une clé est faible avant l'expansion

Attaques – Cryptographie différentielle (Vaudenay)

- avec $2^{8 \cdot \text{nbre rondes}+1}$ textes en clair choisis
- clé faible dans une table S :
  - il y a une collision : $S(a) = S(b)$ pour $a \neq b$
  - $\Longrightarrow 2^{4 \cdot \text{nbre rondes}+1}$ textes en clair choisis
- Il n'y a pas moyen de vérifier si une clé est faible avant l'expansion

Attaques – Cryptographie différentielle (Vaudenay)

- avec $2^{8 \cdot \text{nbre rondes}+1}$ textes en clair choisis
- clé faible dans une table S :
  - il y a une collision : $S(a) = S(b)$ pour $a \neq b$
  - $\Longrightarrow 2^{4 \cdot \text{nbre rondes}+1}$ textes en clair choisis
- Il n'y a pas moyen de vérifier si une clé est faible avant l'expansion

Attaques – continuation

- Cryptanalyse :
  - Vaudenay : les permutations dans Blowfish s'écartaient sensiblement des permutations complètement aléatoires sur quatorze rondes
  - L'attaque des clés dites faibles, détectables et cassables en seulement $2^{4\text{nombre rondes}} + 1$ textes clairs choisis ne peut être étendue au Blowfish complet avec ses 16 rondes.
  - Rijmen : une attaque sur quatre rondes basée sur une cryptanalyse différentielle de second degré.
  - La recherche exhaustive reste la seule solution pour vaincre un Blowfish complet à ce jour.

Attaques – continuation

- Cryptanalyse :
  - Vaudenay : les permutations dans Blowfish s'écartaient sensiblement des permutations complètement aléatoires sur quatorze rondes
  - L'attaque des clés dites faibles, détectables et cassables en seulement $2^{4\text{nombre rondes}} + 1$ textes clairs choisis ne peut être étendue au Blowfish complet avec ses 16 rondes.
  - Rijmen : une attaque sur quatre rondes basée sur une cryptanalyse différentielle de second degré.
  - La recherche exhaustive reste la seule solution pour vaincre un Blowfish complet à ce jour.

Attaques – continuation

- Cryptanalyse :
  - Vaudenay : les permutations dans Blowfish s'écartaient sensiblement des permutations complètement aléatoires sur quatorze rondes
  - L'attaque des clés dites faibles, détectables et cassables en seulement $2^{4\text{nombre rondes}} + 1$ textes clairs choisis ne peut être étendue au Blowfish complet avec ses 16 rondes.
  - Rijmen : une attaque sur quatre rondes basée sur une cryptanalyse différentielle de second degré.
  - La recherche exhaustive reste la seule solution pour vaincre un Blowfish complet à ce jour.

Attaques – continuation

- Cryptanalyse :
  - Vaudenay : les permutations dans Blowfish s'écartaient sensiblement des permutations complètement aléatoires sur quatorze rondes
  - L'attaque des clés dites faibles, détectables et cassables en seulement $2^{4\text{nombre rondes}} + 1$ textes clairs choisis ne peut être étendue au Blowfish complet avec ses 16 rondes.
  - Rijmen : une attaque sur quatre rondes basée sur une cryptanalyse différentielle de second degré.
  - La recherche exhaustive reste la seule solution pour vaincre un Blowfish complet à ce jour.

Attaques – continuation

- Cryptanalyse :
  - Vaudenay : les permutations dans Blowfish s'écartaient sensiblement des permutations complètement aléatoires sur quatorze rondes
  - L'attaque des clés dites faibles, détectables et cassables en seulement $2^{4\text{nombre rondes}} + 1$ textes clairs choisis ne peut être étendue au Blowfish complet avec ses 16 rondes.
  - Rijmen : une attaque sur quatre rondes basée sur une cryptanalyse différentielle de second degré.
  - La recherche exhaustive reste la seule solution pour vaincre un Blowfish complet à ce jour.

Attaques par surchiffrement (par collisions)

- $l_k(x) \stackrel{?}{=} e_{k_1}(e_{k_2}(x))$
- Possible si l'ensemble de clés a une structure de groupe

Attaques par surchiffrement (par collisions)

- $I_k(x) \stackrel{?}{=} e_{k_1}(e_{k_2}(x))$
- Possible si l'ensemble de clés a une structure de groupe

Génération de clé pour Blowfish

- La taille du bloc est de 64
- Le nombre de rondes est de 16
- La clé utilisée est de taille variable, de 32 à 448 bits
- La clé est utilisée pour générer
  - 18 sous-clés de 32 bits, stockées dans des P-tableaux
  - 4 S-boîtes de 8×32, stockées dans des S-tableaux
- Besoin de 521 encodages, donc le changement de clé est lent
- Initialisation : 18 sous-clés et quatre S-tables :
  - $18 * 32 + 4 * 256 * 32 = 576 + 32768 = 33344$, $33344/4 = 8366$ chiffres hexa de PI.
- Générer 18 sous-clé et quatre S-tables :
  - appliquer l'algorithme Blowfish $(18 + 4 * 256)/2 = 1042/2 = 521$ fois
  - mettre le texte initial en clair : $T = (0x000000, 0x000000)$,
  - ensuite $T = (P1, P2), \ldots$

Génération de clé pour Blowfish

- La taille du bloc est de 64
- Le nombre de rondes est de 16
- La clé utilisée est de taille variable, de 32 à 448 bits
- La clé est utilisée pour générer
  - 18 sous-clés de 32 bits, stockées dans des P-tableaux
  - 4 S-boîtes de 8×32, stockées dans des S-tableaux
- Besoin de 521 encodages, donc le changement de clé est lent
- Initialisation : 18 sous-clés et quatre S-tables :
  - $18 * 32 + 4 * 256 * 32 = 576 + 32768 = 33344$, $33344/4 = 8366$ chiffres hexa de PI.
- Générer 18 sous-clé et quatre S-tables :
  - appliquer l'algorithme Blowfish $(18 + 4 * 256)/2 = 1042/2 = 521$ fois
  - mettre le texte initial en clair : $T = (0x000000, 0x000000)$,
  - ensuite $T = (P1, P2), \ldots$

Génération de clé pour Blowfish

- La taille du bloc est de 64
- Le nombre de rondes est de 16
- La clé utilisée est de taille variable, de 32 à 448 bits
- La clé est utilisée pour générer
  - 18 sous-clés de 32 bits, stockées dans des P-tableaux
  - 4 S-boîtes de 8×32, stockées dans des S-tableaux
- Besoin de 521 encodages, donc le changement de clé est lent
- Initialisation : 18 sous-clés et quatre S-tables :
  - $18 * 32 + 4 * 256 * 32 = 576 + 32768 = 33344$, $33344/4 = 8366$ chiffres hexa de PI.
- Générer 18 sous-clé et quatre S-tables :
  - appliquer l'algorithme Blowfish $(18 + 4 * 256)/2 = 1042/2 = 521$ fois
  - mettre le texte initial en clair : $T = (0x00000, 0x00000)$,
  - ensuite $T = (P1, P2), \ldots$

Génération de clé pour Blowfish

- La taille du bloc est de 64
- Le nombre de rondes est de 16
- La clé utilisée est de taille variable, de 32 à 448 bits
- La clé est utilisée pour générer
  - 18 sous-clés de 32 bits, stockées dans des P-tableaux
  - 4 S-boîtes de 8×32, stockées dans des S-tableaux
- Besoin de 521 encodages, donc le changement de clé est lent
- Initialisation : 18 sous-clés et quatre S-tables :
  - $18 * 32 + 4 * 256 * 32 = 576 + 32768 = 33344$, $33344/4 = 8366$ chiffres hexa de PI.
- Générer 18 sous-clé et quatre S-tables :
  - appliquer l'algorithme Blowfish $(18 + 4 * 256)/2 = 1042/2 = 521$ fois
  - mettre le texte initial en clair : $T = (0x00000, 0x00000)$,
  - ensuite $T = (P1, P2), \ldots$

Génération de clé pour Blowfish

- La taille du bloc est de 64
- Le nombre de rondes est de 16
- La clé utilisée est de taille variable, de 32 à 448 bits
- La clé est utilisée pour générer
  - 18 sous-clés de 32 bits, stockées dans des P-tableaux
  - 4 S-boîtes de 8×32, stockées dans des S-tableaux
- Besoin de 521 encodages, donc le changement de clé est lent
- Initialisation : 18 sous-clés et quatre S-tables :
  - $18 * 32 + 4 * 256 * 32 = 576 + 32768 = 33344$, $33344/4 = 8366$ chiffres hexa de PI.
- Générer 18 sous-clé et quatre S-tables :
  - appliquer l'algorithme Blowfish $(18 + 4 * 256)/2 = 1042/2 = 521$ fois
  - mettre le texte initial en clair : $T = (0x00000, 0x00000)$,
  - ensuite $T = (P1, P2), \ldots$

Génération de clé pour Blowfish

- La taille du bloc est de 64
- Le nombre de rondes est de 16
- La clé utilisée est de taille variable, de 32 à 448 bits
- La clé est utilisée pour générer
  - 18 sous-clés de 32 bits, stockées dans des P-tableaux
  - 4 S-boîtes de 8×32, stockées dans des S-tableaux
- Besoin de 521 encodages, donc le changement de clé est lent
- Initialisation : 18 sous-clés et quatre S-tables :
  - $18 * 32 + 4 * 256 * 32 = 576 + 32768 = 33344$, $33344/4 = 8366$ chiffres hexa de PI.
- Générer 18 sous-clé et quatre S-tables :
  - appliquer l'algorithme Blowfish $(18 + 4 * 256)/2 = 1042/2 = 521$ fois
  - mettre le texte initial en clair : $T = (0x00000, 0x00000)$,
  - ensuite $T = (P1, P2), \ldots$

Génération de clé pour Blowfish

- La taille du bloc est de 64
- Le nombre de rondes est de 16
- La clé utilisée est de taille variable, de 32 à 448 bits
- La clé est utilisée pour générer
  - 18 sous-clés de 32 bits, stockées dans des P-tableaux
  - 4 S-boîtes de 8×32, stockées dans des S-tableaux
- Besoin de 521 encodages, donc le changement de clé est lent
- Initialisation : 18 sous-clés et quatre S-tables :
  - $18 * 32 + 4 * 256 * 32 = 576 + 32768 = 33344$, $33344/4 = 8366$ chiffres hexa de PI.
- Générer 18 sous-clé et quatre S-tables :
  - appliquer l'algorithme Blowfish $(18 + 4 * 256)/2 = 1042/2 = 521$ fois
  - mettre le texte initial en clair : $T = (0x00000, 0x00000)$,
  - ensuite $T = (P1, P2), \ldots$

Génération de clé pour Blowfish

- La taille du bloc est de 64
- Le nombre de rondes est de 16
- La clé utilisée est de taille variable, de 32 à 448 bits
- La clé est utilisée pour générer
  - 18 sous-clés de 32 bits, stockées dans des P-tableaux
  - 4 S-boîtes de 8×32, stockées dans des S-tableaux
- Besoin de 521 encodages, donc le changement de clé est lent
- Initialisation : 18 sous-clés et quatre S-tables :
  - $18 * 32 + 4 * 256 * 32 = 576 + 32768 = 33344$, $33344/4 = 8366$ chiffres hexa de PI.
- Générer 18 sous-clé et quatre S-tables :
  - appliquer l'algorithme Blowfish $(18 + 4 * 256)/2 = 1042/2 = 521$ fois
  - mettre le texte initial en clair : $T = (0x00000, 0x00000)$,
  - ensuite $T = (P1, P2), \ldots$

Génération de clé pour Blowfish

- La taille du bloc est de 64
- Le nombre de rondes est de 16
- La clé utilisée est de taille variable, de 32 à 448 bits
- La clé est utilisée pour générer
  - 18 sous-clés de 32 bits, stockées dans des P-tableaux
  - 4 S-boîtes de 8×32, stockées dans des S-tableaux
- Besoin de 521 encodages, donc le changement de clé est lent
- Initialisation : 18 sous-clés et quatre S-tables :
  - $18 * 32 + 4 * 256 * 32 = 576 + 32768 = 33344$, $33344/4 = 8366$ chiffres hexa de PI.
- Générer 18 sous-clé et quatre S-tables :
  - appliquer l'algorithme Blowfish $(18 + 4 * 256)/2 = 1042/2 = 521$ fois
  - mettre le texte initial en clair : $T = (0x00000, 0x00000)$,
  - ensuite $T = (P1, P2), \ldots$

Génération de clé pour Blowfish

- La taille du bloc est de 64
- Le nombre de rondes est de 16
- La clé utilisée est de taille variable, de 32 à 448 bits
- La clé est utilisée pour générer
  - 18 sous-clés de 32 bits, stockées dans des P-tableaux
  - 4 S-boîtes de 8×32, stockées dans des S-tableaux
- Besoin de 521 encodages, donc le changement de clé est lent
- Initialisation : 18 sous-clés et quatre S-tables :
  - $18 * 32 + 4 * 256 * 32 = 576 + 32768 = 33344$, $33344/4 = 8366$ chiffres hexa de PI.
- Générer 18 sous-clé et quatre S-tables :
  - appliquer l'algorithme Blowfish $(18 + 4 * 256)/2 = 1042/2 = 521$ fois
  - mettre le texte initial en clair : $T = (0\text{x}000000, 0\text{x}000000)$,
  - ensuite $T = (P1, P2), \ldots$

Génération de clé pour Blowfish

- La taille du bloc est de 64
- Le nombre de rondes est de 16
- La clé utilisée est de taille variable, de 32 à 448 bits
- La clé est utilisée pour générer
  - 18 sous-clés de 32 bits, stockées dans des P-tableaux
  - 4 S-boîtes de 8×32, stockées dans des S-tableaux
- Besoin de 521 encodages, donc le changement de clé est lent
- Initialisation : 18 sous-clés et quatre S-tables :
  - $18 * 32 + 4 * 256 * 32 = 576 + 32768 = 33344$, $33344/4 = 8366$ chiffres hexa de PI.
- Générer 18 sous-clé et quatre S-tables :
  - appliquer l'algorithme Blowfish $(18 + 4 * 256)/2 = 1042/2 = 521$ fois
  - mettre le texte initial en clair : $T = (0\text{x}000000, 0\text{x}000000)$,
  - ensuite $T = (P1, P2), \ldots$

Génération de clé pour Blowfish

- La taille du bloc est de 64
- Le nombre de rondes est de 16
- La clé utilisée est de taille variable, de 32 à 448 bits
- La clé est utilisée pour générer
  - 18 sous-clés de 32 bits, stockées dans des P-tableaux
  - 4 S-boîtes de 8×32, stockées dans des S-tableaux
- Besoin de 521 encodages, donc le changement de clé est lent
- Initialisation : 18 sous-clés et quatre S-tables :
  - $18 * 32 + 4 * 256 * 32 = 576 + 32768 = 33344$, $33344/4 = 8366$ chiffres hexa de PI.
- Générer 18 sous-clé et quatre S-tables :
  - appliquer l'algorithme Blowfish $(18 + 4 * 256)/2 = 1042/2 = 521$ fois
  - mettre le texte initial en clair : $T = (0\text{x}000000, 0\text{x}000000)$,
  - ensuite $T = (P1, P2), \ldots$

Génération de clé pour Blowfish

- La taille du bloc est de 64
- Le nombre de rondes est de 16
- La clé utilisée est de taille variable, de 32 à 448 bits
- La clé est utilisée pour générer
  - 18 sous-clés de 32 bits, stockées dans des P-tableaux
  - 4 S-boîtes de 8×32, stockées dans des S-tableaux
- Besoin de 521 encodages, donc le changement de clé est lent
- Initialisation : 18 sous-clés et quatre S-tables :
  - $18 * 32 + 4 * 256 * 32 = 576 + 32768 = 33344$, $33344/4 = 8366$ chiffres hexa de PI.
- Générer 18 sous-clé et quatre S-tables :
  - appliquer l'algorithme Blowfish $(18 + 4 * 256)/2 = 1042/2 = 521$ fois
  - mettre le texte initial en clair : $T = (0\text{x}000000, 0\text{x}000000)$,
  - ensuite $T = (P1, P2), \ldots$

L'algorithme d'ordonnancement des clés

- Entrées :
  - K – la clé – 32 bits ou plus
  - PI – la représentation binaire de la partie fractionelle de π :
    - $PI = 3.1415927\ldots - 3.0$
    - $PI = 2/16 + 4 * /16 * *2 + 3/16 * *3 + 15/16 * *4 + \ldots$
    - $PI = 0x243f6a8885a308d313198a2e03707344\ldots$
- Sorties :
  - $P1, P2, \ldots, P18$: 18 sous-clés de 32 bits
  - $S1[], S2[], S3[], S4[]$: quatre S-boîtes – des tableaux de 256 éléments de 32-bits

L'algorithme d'ordonnancement des clés

- Entrées :
  - K – la clé – 32 bits ou plus
  - PI – la représentation binaire de la partie fractionelle de π :
    - $PI = 3.1415927\ldots - 3.0$
    - $PI = 2/16 + 4 * /16 * *2 + 3/16 * *3 + 15/16 * *4 + \ldots$
    - $PI = 0x243f6a8885a308d313198a2e03707344\ldots$
- Sorties :
  - $P1, P2, \ldots, P18$: 18 sous-clés de 32 bits
  - $S1[], S2[], S3[], S4[]$: quatre S-boîtes – des tableaux de 256 éléments de 32-bits

L'algorithme d'ordonnancement des clés

- Entrées :
  - K – la clé – 32 bits ou plus
  - PI – la représentation binaire de la partie fractionelle de π :
    - $PI = 3.1415927 \cdots - 3.0$
    - $PI = 2/16 + 4 * /16 * *2 + 3/16 * *3 + 15/16 * *4 + \ldots$
    - $PI = 0x243f6a8885a308d313198a2e03707344 \ldots$
- Sorties :
  - $P1, P2, \ldots, P18$: 18 sous-clés de 32 bits
  - $S1[], S2[], S3[], S4[]$: quatre S-boîtes – des tableaux de 256 éléments de 32-bits

L'algorithme d'ordonnancement des clés

- Entrées :
  - K – la clé – 32 bits ou plus
  - PI – la représentation binaire de la partie fractionelle de π :
    - $PI = 3.1415927 \cdots - 3.0$
    - $PI = 2/16 + 4 * /16 * *2 + 3/16 * *3 + 15/16 * *4 + \ldots$
    - $PI = 0x243f6a8885a308d313198a2e03707344 \ldots$
- Sorties :
  - $P1, P2, \ldots, P18$: 18 sous-clés de 32 bits
  - $S1[], S2[], S3[], S4[]$: quatre S-boîtes – des tableaux de 256 éléments de 32-bits

L'algorithme d'ordonnancement des clés

- Entrées :
  - K – la clé – 32 bits ou plus
  - PI – la représentation binaire de la partie fractionelle de π :
    - $PI = 3.1415927 \cdots - 3.0$
    - $PI = 2/16 + 4 * /16 * *2 + 3/16 * *3 + 15/16 * *4 + \ldots$
    - $PI = 0x243f6a8885a308d313198a2e03707344 \ldots$
- Sorties :
  - $P1, P2, \ldots, P18$: 18 sous-clés de 32 bits
  - $S1[], S2[], S3[], S4[]$: quatre S-boîtes – des tableaux de 256 éléments de 32-bits

L'algorithme d'ordonnancement des clés

- Entrées :
  - K – la clé – 32 bits ou plus
  - PI – la représentation binaire de la partie fractionelle de π :
    - $PI = 3.1415927 \cdots - 3.0$
    - $PI = 2/16 + 4 * /16 * *2 + 3/16 * *3 + 15/16 * *4 + \ldots$
    - $PI = 0x243f6a8885a308d313198a2e03707344 \ldots$
- Sorties :
  - $P1, P2, \ldots, P18$: 18 sous-clés de 32 bits
  - $S1[], S2[], S3[], S4[]$: quatre S-boîtes – des tableaux de 256 éléments de 32-bits

L'algorithme d'ordonnancement des clés

- Entrées :
  - K – la clé – 32 bits ou plus
  - PI – la représentation binaire de la partie fractionelle de π :
    - $PI = 3.1415927 \cdots - 3.0$
    - $PI = 2/16 + 4 * /16 * *2 + 3/16 * *3 + 15/16 * *4 + \ldots$
    - $PI = 0x243f6a8885a308d313198a2e03707344 \ldots$
- Sorties :
  - $P1$, $P2$, ..., $P18$: 18 sous-clés de 32 bits
  - $S1[]$, $S2[]$, $S3[]$, $S4[]$: quatre S-boîtes – des tableaux de 256 éléments de 32-bits

L'algorithme d'ordonnancement des clés

- Entrées :
  - K – la clé – 32 bits ou plus
  - PI – la représentation binaire de la partie fractionelle de π :
    - $PI = 3.1415927 \cdots - 3.0$
    - $PI = 2/16 + 4 * /16 * *2 + 3/16 * *3 + 15/16 * *4 + \ldots$
    - $PI = 0x243f6a8885a308d313198a2e03707344 \ldots$
- Sorties :
  - $P1$, $P2$, ..., $P18$: 18 sous-clés de 32 bits
  - $S1[]$, $S2[]$, $S3[]$, $S4[]$: quatre S-boîtes – des tableaux de 256 éléments de 32-bits

L'algorithme d'ordonnancement des clés

- Entrées :
  - K – la clé – 32 bits ou plus
  - PI – la représentation binaire de la partie fractionelle de π :
    - $PI = 3.1415927 \cdots - 3.0$
    - $PI = 2/16 + 4 * /16 * *2 + 3/16 * *3 + 15/16 * *4 + \ldots$
    - $PI = 0x243f6a8885a308d313198a2e03707344 \ldots$
- Sorties :
  - $P1$, $P2$, ..., $P18$: 18 sous-clés de 32 bits
  - $S1[]$, $S2[]$, $S3[]$, $S4[]$: quatre S-boîtes – des tableaux de 256 éléments de 32-bits

L'algorithme d'ordonnancement des clés

- Algorithme
  - $(P1, P2, \ldots, P18, S1[], S2[], S3[], S4[]) = PI$
  - $K' = (K, K, K, \ldots)$, répéter la clé jusqu'à une longueur de $18 * 32$ bits
  - $(P1, P2, \ldots, P18) = (P1, P2, \ldots, P18)\mathrm{XOR}K'$
  - $T = (0x000000, 0x000000)$, texte en clair initial mis en place
  - $T = Blowfish(T)$, application de l'algorithme Blowfish
  - $(P1, P2) = T$, mise-à-jour des deux sous-clés
  - $T = Blowfish(T)$, autre application de Blowfish
  - $(P3, P4) = T$
  - $\ldots$
  - $T = Blowfish(T)$
  - $(P17, P18) = T$
  - $T = Blowfish(T)$
  - $(S1[0], S1[1]) = T$
  - $T = Blowfish(T)$
  - $(S1[2], S1[3]) = T$
  - $\ldots$
  - $T = Blowfish(T)$
  - $(S1[254], S1[255]) = T$
  - $T = Blowfish(T)$
  - $(S2[0], S2[1]) = T$
  - $\ldots$
  - $T = Blowfish(T)$
  - $(S4[254], S4[255]) = T$

L'algorithme d'ordonnancement des clés

- Algorithme
  - $(P1, P2, \ldots, P18, S1[], S2[], S3[], S4[]) = PI$
  - $K' = (K, K, K, \ldots)$, répéter la clé jusqu'à une longueur de $18 * 32$ bits
  - $(P1, P2, \ldots, P18) = (P1, P2, \ldots, P18) \mathrm{XOR} K'$
  - $T = (0x000000, 0x000000)$, texte en clair initial mis en place
  - $T = Blowfish(T)$, application de l'algorithme Blowfish
  - $(P1, P2) = T$, mise-à-jour des deux sous-clés
  - $T = Blowfish(T)$, autre application de Blowfish
  - $(P3, P4) = T$
  - ...
  - $T = Blowfish(T)$
  - $(P17, P18) = T$
  - $T = Blowfish(T)$
  - $(S1[0], S1[1]) = T$
  - $T = Blowfish(T)$
  - $(S1[2], S1[3]) = T$
  - ...
  - $T = Blowfish(T)$
  - $(S1[254], S1[255]) = T$
  - $T = Blowfish(T)$
  - $(S2[0], S2[1]) = T$
  - ...
  - $T = Blowfish(T)$
  - $(S4[254], S4[255]) = T$

L'algorithme d'ordonnancement des clés

- Algorithme
  - $(P1, P2, \ldots, P18, S1[], S2[], S3[], S4[]) = PI$
  - $K' = (K, K, K, \ldots)$, répéter la clé jusqu'à une longueur de $18 * 32$ bits
  - $(P1, P2, \ldots, P18) = (P1, P2, \ldots, P18)\mathrm{XOR}K'$
  - $T = (0x000000, 0x000000)$, texte en clair initial mis en place
  - $T = Blowfish(T)$, application de l'algorithme Blowfish
  - $(P1, P2) = T$, mise-à-jour des deux sous-clés
  - $T = Blowfish(T)$, autre application de Blowfish
  - $(P3, P4) = T$
  - $\ldots$
  - $T = Blowfish(T)$
  - $(P17, P18) = T$
  - $T = Blowfish(T)$
  - $(S1[0], S1[1]) = T$
  - $T = Blowfish(T)$
  - $(S1[2], S1[3]) = T$
  - $\ldots$
  - $T = Blowfish(T)$
  - $(S1[254], S1[255]) = T$
  - $T = Blowfish(T)$
  - $(S2[0], S2[1]) = T$
  - $\ldots$
  - $T = Blowfish(T)$
  - $(S4[254], S4[255]) = T$

L'algorithme d'ordonnancement des clés

- Algorithme
  - $(P1, P2, \ldots, P18, S1[], S2[], S3[], S4[]) = PI$
  - $K' = (K, K, K, \ldots)$, répéter la clé jusqu'à une longueur de $18 * 32$ bits
  - $(P1, P2, \ldots, P18) = (P1, P2, \ldots, P18)\mathrm{XOR}K'$
  - $T = (0x000000, 0x000000)$, texte en clair initial mis en place
  - $T = Blowfish(T)$, application de l'algorithme Blowfish
  - $(P1, P2) = T$, mise-à-jour des deux sous-clés
  - $T = Blowfish(T)$, autre application de Blowfish
  - $(P3, P4) = T$
  - $\ldots$
  - $T = Blowfish(T)$
  - $(P17, P18) = T$
  - $T = Blowfish(T)$
  - $(S1[0], S1[1]) = T$
  - $T = Blowfish(T)$
  - $(S1[2], S1[3]) = T$
  - $\ldots$
  - $T = Blowfish(T)$
  - $(S1[254], S1[255]) = T$
  - $T = Blowfish(T)$
  - $(S2[0], S2[1]) = T$
  - $\ldots$
  - $T = Blowfish(T)$
  - $(S4[254], S4[255]) = T$

L'algorithme d'ordonnancement des clés

- Algorithme
  - $(P1, P2, \ldots, P18, S1[], S2[], S3[], S4[]) = PI$
  - $K' = (K, K, K, \ldots)$, répéter la clé jusqu'à une longueur de $18 * 32$ bits
  - $(P1, P2, \ldots, P18) = (P1, P2, \ldots, P18)\mathrm{XOR}K'$
  - $T = (0x000000, 0x000000)$, texte en clair initial mis en place
  - $T = Blowfish(T)$, application de l'algorithme Blowfish
  - $(P1, P2) = T$, mise-à-jour des deux sous-clés
  - $T = Blowfish(T)$, autre application de Blowfish
  - $(P3, P4) = T$
  - ...
  - $T = Blowfish(T)$
  - $(P17, P18) = T$
  - $T = Blowfish(T)$
  - $(S1[0], S1[1]) = T$
  - $T = Blowfish(T)$
  - $(S1[2], S1[3]) = T$
  - ...
  - $T = Blowfish(T)$
  - $(S1[254], S1[255]) = T$
  - $T = Blowfish(T)$
  - $(S2[0], S2[1]) = T$
  - ...
  - $T = Blowfish(T)$
  - $(S4[254], S4[255]) = T$

L'algorithme d'ordonnancement des clés

- Algorithme
  - $(P1, P2, \ldots, P18, S1[], S2[], S3[], S4[]) = PI$
  - $K' = (K, K, K, \ldots)$, répéter la clé jusqu'à une longueur de $18 * 32$ bits
  - $(P1, P2, \ldots, P18) = (P1, P2, \ldots, P18)\mathrm{XOR}K'$
  - $T = (0x000000, 0x000000)$, texte en clair initial mis en place
  - $T = Blowfish(T)$, application de l'algorithme Blowfish
  - $(P1, P2) = T$, mise-à-jour des deux sous-clés
  - $T = Blowfish(T)$, autre application de Blowfish
  - $(P3, P4) = T$
  - ...
  - $T = Blowfish(T)$
  - $(P17, P18) = T$
  - $T = Blowfish(T)$
  - $(S1[0], S1[1]) = T$
  - $T = Blowfish(T)$
  - $(S1[2], S1[3]) = T$
  - ...
  - $T = Blowfish(T)$
  - $(S1[254], S1[255]) = T$
  - $T = Blowfish(T)$
  - $(S2[0], S2[1]) = T$
  - ...
  - $T = Blowfish(T)$
  - $(S4[254], S4[255]) = T$

L'algorithme d'ordonnancement des clés

- Algorithme
  - $(P1, P2, \ldots, P18, S1[], S2[], S3[], S4[]) = PI$
  - $K' = (K, K, K, \ldots)$, répéter la clé jusqu'à une longueur de $18 * 32$ bits
  - $(P1, P2, \ldots, P18) = (P1, P2, \ldots, P18)\mathrm{XOR}K'$
  - $T = (0x000000, 0x000000)$, texte en clair initial mis en place
  - $T = Blowfish(T)$, application de l'algorithme Blowfish
  - $(P1, P2) = T$, mise-à-jour des deux sous-clés
  - $T = Blowfish(T)$, autre application de Blowfish
  - $(P3, P4) = T$
  - $\ldots$
  - $T = Blowfish(T)$
  - $(P17, P18) = T$
  - $T = Blowfish(T)$
  - $(S1[0], S1[1]) = T$
  - $T = Blowfish(T)$
  - $(S1[2], S1[3]) = T$
  - $\ldots$
  - $T = Blowfish(T)$
  - $(S1[254], S1[255]) = T$
  - $T = Blowfish(T)$
  - $(S2[0], S2[1]) = T$
  - $\ldots$
  - $T = Blowfish(T)$
  - $(S4[254], S4[255]) = T$

L'algorithme d'ordonnancement des clés

- Algorithme
  - $(P1, P2, \ldots, P18, S1[], S2[], S3[], S4[]) = PI$
  - $K' = (K, K, K, \ldots)$, répéter la clé jusqu'à une longueur de $18 * 32$ bits
  - $(P1, P2, \ldots, P18) = (P1, P2, \ldots, P18)\mathrm{XOR}K'$
  - $T = (0x000000, 0x000000)$, texte en clair initial mis en place
  - $T = Blowfish(T)$, application de l'algorithme Blowfish
  - $(P1, P2) = T$, mise-à-jour des deux sous-clés
  - $T = Blowfish(T)$, autre application de Blowfish
  - $(P3, P4) = T$
  - $\ldots$
  - $T = Blowfish(T)$
  - $(P17, P18) = T$
  - $T = Blowfish(T)$
  - $(S1[0], S1[1]) = T$
  - $T = Blowfish(T)$
  - $(S1[2], S1[3]) = T$
  - $\ldots$
  - $T = Blowfish(T)$
  - $(S1[254], S1[255]) = T$
  - $T = Blowfish(T)$
  - $(S2[0], S2[1]) = T$
  - $\ldots$
  - $T = Blowfish(T)$
  - $(S4[254], S4[255]) = T$

L'algorithme d'ordonnancement des clés

- Algorithme
  - $(P1, P2, \ldots, P18, S1[], S2[], S3[], S4[]) = PI$
  - $K' = (K, K, K, \ldots)$, répéter la clé jusqu'à une longueur de $18 * 32$ bits
  - $(P1, P2, \ldots, P18) = (P1, P2, \ldots, P18)\mathrm{XOR}K'$
  - $T = (0x000000, 0x000000)$, texte en clair initial mis en place
  - $T = Blowfish(T)$, application de l'algorithme Blowfish
  - $(P1, P2) = T$, mise-à-jour des deux sous-clés
  - $T = Blowfish(T)$, autre application de Blowfish
  - $(P3, P4) = T$
  - $\ldots$
  - $T = Blowfish(T)$
  - $(P17, P18) = T$
  - $T = Blowfish(T)$
  - $(S1[0], S1[1]) = T$
  - $T = Blowfish(T)$
  - $(S1[2], S1[3]) = T$
  - $\ldots$
  - $T = Blowfish(T)$
  - $(S1[254], S1[255]) = T$
  - $T = Blowfish(T)$
  - $(S2[0], S2[1]) = T$
  - $\ldots$
  - $T = Blowfish(T)$
  - $(S4[254], S4[255]) = T$

L'algorithme d'ordonnancement des clés

- Algorithme
  - $(P1, P2, \ldots, P18, S1[], S2[], S3[], S4[]) = PI$
  - $K' = (K, K, K, \ldots)$, répéter la clé jusqu'à une longueur de $18 * 32$ bits
  - $(P1, P2, \ldots, P18) = (P1, P2, \ldots, P18)\mathrm{XOR}K'$
  - $T = (0x000000, 0x000000)$, texte en clair initial mis en place
  - $T = Blowfish(T)$, application de l'algorithme Blowfish
  - $(P1, P2) = T$, mise-à-jour des deux sous-clés
  - $T = Blowfish(T)$, autre application de Blowfish
  - $(P3, P4) = T$
  - ...
  - $T = Blowfish(T)$
  - $(P17, P18) = T$
  - $T = Blowfish(T)$
  - $(S1[0], S1[1]) = T$
  - $T = Blowfish(T)$
  - $(S1[2], S1[3]) = T$
  - ...
  - $T = Blowfish(T)$
  - $(S1[254], S1[255]) = T$
  - $T = Blowfish(T)$
  - $(S2[0], S2[1]) = T$
  - ...
  - $T = Blowfish(T)$
  - $(S4[254], S4[255]) = T$

L'algorithme d'ordonnancement des clés

- Algorithme
  - $(P1, P2, \ldots, P18, S1[], S2[], S3[], S4[]) = PI$
  - $K' = (K, K, K, \ldots)$, répéter la clé jusqu'à une longueur de $18 * 32$ bits
  - $(P1, P2, \ldots, P18) = (P1, P2, \ldots, P18)\mathrm{XOR}K'$
  - $T = (0x000000, 0x000000)$, texte en clair initial mis en place
  - $T = Blowfish(T)$, application de l'algorithme Blowfish
  - $(P1, P2) = T$, mise-à-jour des deux sous-clés
  - $T = Blowfish(T)$, autre application de Blowfish
  - $(P3, P4) = T$
  - ...
  - $T = Blowfish(T)$
  - $(P17, P18) = T$
  - $T = Blowfish(T)$
  - $(S1[0], S1[1]) = T$
  - $T = Blowfish(T)$
  - $(S1[2], S1[3]) = T$
  - ...
  - $T = Blowfish(T)$
  - $(S1[254], S1[255]) = T$
  - $T = Blowfish(T)$
  - $(S2[0], S2[1]) = T$
  - ...
  - $T = Blowfish(T)$
  - $(S4[254], S4[255]) = T$

L'algorithme d'ordonnancement des clés

- Algorithme
  - $(P1, P2, \ldots, P18, S1[], S2[], S3[], S4[]) = PI$
  - $K' = (K, K, K, \ldots)$, répéter la clé jusqu'à une longueur de $18 * 32$ bits
  - $(P1, P2, \ldots, P18) = (P1, P2, \ldots, P18)\mathrm{XOR}K'$
  - $T = (0x000000, 0x000000)$, texte en clair initial mis en place
  - $T = Blowfish(T)$, application de l'algorithme Blowfish
  - $(P1, P2) = T$, mise-à-jour des deux sous-clés
  - $T = Blowfish(T)$, autre application de Blowfish
  - $(P3, P4) = T$
  - ...
  - $T = Blowfish(T)$
  - $(P17, P18) = T$
  - $T = Blowfish(T)$
  - $(S1[0], S1[1]) = T$
  - $T = Blowfish(T)$
  - $(S1[2], S1[3]) = T$
  - ...
  - $T = Blowfish(T)$
  - $(S1[254], S1[255]) = T$
  - $T = Blowfish(T)$
  - $(S2[0], S2[1]) = T$
  - ...
  - $T = Blowfish(T)$
  - $(S4[254], S4[255]) = T$

L'algorithme d'ordonnancement des clés

- Algorithme
  - $(P1, P2, \ldots, P18, S1[], S2[], S3[], S4[]) = PI$
  - $K' = (K, K, K, \ldots)$, répéter la clé jusqu'à une longueur de $18 * 32$ bits
  - $(P1, P2, \ldots, P18) = (P1, P2, \ldots, P18)\mathrm{XOR}K'$
  - $T = (0x000000, 0x000000)$, texte en clair initial mis en place
  - $T = Blowfish(T)$, application de l'algorithme Blowfish
  - $(P1, P2) = T$, mise-à-jour des deux sous-clés
  - $T = Blowfish(T)$, autre application de Blowfish
  - $(P3, P4) = T$
  - ...
  - $T = Blowfish(T)$
  - $(P17, P18) = T$
  - $T = Blowfish(T)$
  - $(S1[0], S1[1]) = T$
  - $T = Blowfish(T)$
  - $(S1[2], S1[3]) = T$
  - ...
  - $T = Blowfish(T)$
  - $(S1[254], S1[255]) = T$
  - $T = Blowfish(T)$
  - $(S2[0], S2[1]) = T$
  - ...
  - $T = Blowfish(T)$
  - $(S4[254], S4[255]) = T$

L'algorithme d'ordonnancement des clés

- Algorithme
  - $(P1, P2, \ldots, P18, S1[], S2[], S3[], S4[]) = PI$
  - $K' = (K, K, K, \ldots)$, répéter la clé jusqu'à une longueur de $18 * 32$ bits
  - $(P1, P2, \ldots, P18) = (P1, P2, \ldots, P18)\mathrm{XOR}K'$
  - $T = (0x000000, 0x000000)$, texte en clair initial mis en place
  - $T = Blowfish(T)$, application de l'algorithme Blowfish
  - $(P1, P2) = T$, mise-à-jour des deux sous-clés
  - $T = Blowfish(T)$, autre application de Blowfish
  - $(P3, P4) = T$
  - ...
  - $T = Blowfish(T)$
  - $(P17, P18) = T$
  - $T = Blowfish(T)$
  - $(S1[0], S1[1]) = T$
  - $T = Blowfish(T)$
  - $(S1[2], S1[3]) = T$
  - ...
  - $T = Blowfish(T)$
  - $(S1[254], S1[255]) = T$
  - $T = Blowfish(T)$
  - $(S2[0], S2[1]) = T$
  - ...
  - $T = Blowfish(T)$
  - $(S4[254], S4[255]) = T$

L'algorithme d'ordonnancement des clés

- Algorithme
  - $(P1, P2, \ldots, P18, S1[], S2[], S3[], S4[]) = PI$
  - $K' = (K, K, K, \ldots)$, répéter la clé jusqu'à une longueur de $18 * 32$ bits
  - $(P1, P2, \ldots, P18) = (P1, P2, \ldots, P18)\mathrm{XOR}K'$
  - $T = (0x000000, 0x000000)$, texte en clair initial mis en place
  - $T = Blowfish(T)$, application de l'algorithme Blowfish
  - $(P1, P2) = T$, mise-à-jour des deux sous-clés
  - $T = Blowfish(T)$, autre application de Blowfish
  - $(P3, P4) = T$
  - ...
  - $T = Blowfish(T)$
  - $(P17, P18) = T$
  - $T = Blowfish(T)$
  - $(S1[0], S1[1]) = T$
  - $T = Blowfish(T)$
  - $(S1[2], S1[3]) = T$
  - ...
  - $T = Blowfish(T)$
  - $(S1[254], S1[255]) = T$
  - $T = Blowfish(T)$
  - $(S2[0], S2[1]) = T$
  - ...
  - $T = Blowfish(T)$
  - $(S4[254], S4[255]) = T$

L'algorithme d'ordonnancement des clés

- Algorithme
  - $(P1, P2, \ldots, P18, S1[], S2[], S3[], S4[]) = PI$
  - $K' = (K, K, K, \ldots)$, répéter la clé jusqu'à une longueur de $18 * 32$ bits
  - $(P1, P2, \ldots, P18) = (P1, P2, \ldots, P18)\mathrm{XOR}K'$
  - $T = (0x000000, 0x000000)$, texte en clair initial mis en place
  - $T = Blowfish(T)$, application de l'algorithme Blowfish
  - $(P1, P2) = T$, mise-à-jour des deux sous-clés
  - $T = Blowfish(T)$, autre application de Blowfish
  - $(P3, P4) = T$
  - ...
  - $T = Blowfish(T)$
  - $(P17, P18) = T$
  - $T = Blowfish(T)$
  - $(S1[0], S1[1]) = T$
  - $T = Blowfish(T)$
  - $(S1[2], S1[3]) = T$
  - ...
  - $T = Blowfish(T)$
  - $(S1[254], S1[255]) = T$
  - $T = Blowfish(T)$
  - $(S2[0], S2[1]) = T$
  - ...
  - $T = Blowfish(T)$
  - $(S4[254], S4[255]) = T$

L'algorithme d'ordonnancement des clés

- Algorithme
  - $(P1, P2, \ldots, P18, S1[], S2[], S3[], S4[]) = PI$
  - $K' = (K, K, K, \ldots)$, répéter la clé jusqu'à une longueur de $18 * 32$ bits
  - $(P1, P2, \ldots, P18) = (P1, P2, \ldots, P18)\mathrm{XOR}K'$
  - $T = (0x000000, 0x000000)$, texte en clair initial mis en place
  - $T = Blowfish(T)$, application de l'algorithme Blowfish
  - $(P1, P2) = T$, mise-à-jour des deux sous-clés
  - $T = Blowfish(T)$, autre application de Blowfish
  - $(P3, P4) = T$
  - ...
  - $T = Blowfish(T)$
  - $(P17, P18) = T$
  - $T = Blowfish(T)$
  - $(S1[0], S1[1]) = T$
  - $T = Blowfish(T)$
  - $(S1[2], S1[3]) = T$
  - ...
  - $T = Blowfish(T)$
  - $(S1[254], S1[255]) = T$
  - $T = Blowfish(T)$
  - $(S2[0], S2[1]) = T$
  - ...
  - $T = Blowfish(T)$
  - $(S4[254], S4[255]) = T$

L'algorithme d'ordonnancement des clés

- Algorithme
  - $(P1, P2, \ldots, P18, S1[], S2[], S3[], S4[]) = PI$
  - $K' = (K, K, K, \ldots)$, répéter la clé jusqu'à une longueur de $18 * 32$ bits
  - $(P1, P2, \ldots, P18) = (P1, P2, \ldots, P18)\mathrm{XOR}K'$
  - $T = (0x000000, 0x000000)$, texte en clair initial mis en place
  - $T = Blowfish(T)$, application de l'algorithme Blowfish
  - $(P1, P2) = T$, mise-à-jour des deux sous-clés
  - $T = Blowfish(T)$, autre application de Blowfish
  - $(P3, P4) = T$
  - ...
  - $T = Blowfish(T)$
  - $(P17, P18) = T$
  - $T = Blowfish(T)$
  - $(S1[0], S1[1]) = T$
  - $T = Blowfish(T)$
  - $(S1[2], S1[3]) = T$
  - ...
  - $T = Blowfish(T)$
  - $(S1[254], S1[255]) = T$
  - $T = Blowfish(T)$
  - $(S2[0], S2[1]) = T$
  - ...
  - $T = Blowfish(T)$
  - $(S4[254], S4[255]) = T$

L'algorithme d'ordonnancement des clés

- Algorithme
  - $(P1, P2, \ldots, P18, S1[], S2[], S3[], S4[]) = PI$
  - $K' = (K, K, K, \ldots)$, répéter la clé jusqu'à une longueur de $18 * 32$ bits
  - $(P1, P2, \ldots, P18) = (P1, P2, \ldots, P18)\mathrm{XOR}K'$
  - $T = (0x000000, 0x000000)$, texte en clair initial mis en place
  - $T = Blowfish(T)$, application de l'algorithme Blowfish
  - $(P1, P2) = T$, mise-à-jour des deux sous-clés
  - $T = Blowfish(T)$, autre application de Blowfish
  - $(P3, P4) = T$
  - ...
  - $T = Blowfish(T)$
  - $(P17, P18) = T$
  - $T = Blowfish(T)$
  - $(S1[0], S1[1]) = T$
  - $T = Blowfish(T)$
  - $(S1[2], S1[3]) = T$
  - ...
  - $T = Blowfish(T)$
  - $(S1[254], S1[255]) = T$
  - $T = Blowfish(T)$
  - $(S2[0], S2[1]) = T$
  - ...
  - $T = Blowfish(T)$
  - $(S4[254], S4[255]) = T$

L'algorithme d'ordonnancement des clés

- Algorithme
  - $(P1, P2, \ldots, P18, S1[], S2[], S3[], S4[]) = PI$
  - $K' = (K, K, K, \ldots)$, répéter la clé jusqu'à une longueur de $18 * 32$ bits
  - $(P1, P2, \ldots, P18) = (P1, P2, \ldots, P18)\mathrm{XOR}K'$
  - $T = (0x000000, 0x000000)$, texte en clair initial mis en place
  - $T = Blowfish(T)$, application de l'algorithme Blowfish
  - $(P1, P2) = T$, mise-à-jour des deux sous-clés
  - $T = Blowfish(T)$, autre application de Blowfish
  - $(P3, P4) = T$
  - ...
  - $T = Blowfish(T)$
  - $(P17, P18) = T$
  - $T = Blowfish(T)$
  - $(S1[0], S1[1]) = T$
  - $T = Blowfish(T)$
  - $(S1[2], S1[3]) = T$
  - ...
  - $T = Blowfish(T)$
  - $(S1[254], S1[255]) = T$
  - $T = Blowfish(T)$
  - $(S2[0], S2[1]) = T$
  - ...
  - $T = Blowfish(T)$
  - $(S4[254], S4[255]) = T$

L'algorithme d'ordonnancement des clés

- Algorithme
  - $(P1, P2, \ldots, P18, S1[], S2[], S3[], S4[]) = PI$
  - $K' = (K, K, K, \ldots)$, répéter la clé jusqu'à une longueur de $18 * 32$ bits
  - $(P1, P2, \ldots, P18) = (P1, P2, \ldots, P18)\mathrm{XOR}K'$
  - $T = (0x000000, 0x000000)$, texte en clair initial mis en place
  - $T = Blowfish(T)$, application de l'algorithme Blowfish
  - $(P1, P2) = T$, mise-à-jour des deux sous-clés
  - $T = Blowfish(T)$, autre application de Blowfish
  - $(P3, P4) = T$
  - ...
  - $T = Blowfish(T)$
  - $(P17, P18) = T$
  - $T = Blowfish(T)$
  - $(S1[0], S1[1]) = T$
  - $T = Blowfish(T)$
  - $(S1[2], S1[3]) = T$
  - ...
  - $T = Blowfish(T)$
  - $(S1[254], S1[255]) = T$
  - $T = Blowfish(T)$
  - $(S2[0], S2[1]) = T$
  - ...
  - $T = Blowfish(T)$
  - $(S4[254], S4[255]) = T$

L'algorithme d'ordonnancement des clés

- Algorithme
  - $(P1, P2, \ldots, P18, S1[], S2[], S3[], S4[]) = PI$
  - $K' = (K, K, K, \ldots)$, répéter la clé jusqu'à une longueur de $18 * 32$ bits
  - $(P1, P2, \ldots, P18) = (P1, P2, \ldots, P18)\mathrm{XOR}K'$
  - $T = (0x000000, 0x000000)$, texte en clair initial mis en place
  - $T = Blowfish(T)$, application de l'algorithme Blowfish
  - $(P1, P2) = T$, mise-à-jour des deux sous-clés
  - $T = Blowfish(T)$, autre application de Blowfish
  - $(P3, P4) = T$
  - ...
  - $T = Blowfish(T)$
  - $(P17, P18) = T$
  - $T = Blowfish(T)$
  - $(S1[0], S1[1]) = T$
  - $T = Blowfish(T)$
  - $(S1[2], S1[3]) = T$
  - ...
  - $T = Blowfish(T)$
  - $(S1[254], S1[255]) = T$
  - $T = Blowfish(T)$
  - $(S2[0], S2[1]) = T$
  - ...
  - $T = Blowfish(T)$
  - $(S4[254], S4[255]) = T$

L'algorithme d'ordonnancement des clés

- Algorithme
  - $(P1, P2, \ldots, P18, S1[], S2[], S3[], S4[]) = PI$
  - $K' = (K, K, K, \ldots)$, répéter la clé jusqu'à une longueur de $18 * 32$ bits
  - $(P1, P2, \ldots, P18) = (P1, P2, \ldots, P18)\mathrm{XOR}K'$
  - $T = (0x000000, 0x000000)$, texte en clair initial mis en place
  - $T = Blowfish(T)$, application de l'algorithme Blowfish
  - $(P1, P2) = T$, mise-à-jour des deux sous-clés
  - $T = Blowfish(T)$, autre application de Blowfish
  - $(P3, P4) = T$
  - ...
  - $T = Blowfish(T)$
  - $(P17, P18) = T$
  - $T = Blowfish(T)$
  - $(S1[0], S1[1]) = T$
  - $T = Blowfish(T)$
  - $(S1[2], S1[3]) = T$
  - ...
  - $T = Blowfish(T)$
  - $(S1[254], S1[255]) = T$
  - $T = Blowfish(T)$
  - $(S2[0], S2[1]) = T$
  - ...
  - $T = Blowfish(T)$
  - $(S4[254], S4[255]) = T$

L'algorithme d'ordonnancement des clés

- Algorithme
  - $(P1, P2, \ldots, P18, S1[], S2[], S3[], S4[]) = PI$
  - $K' = (K, K, K, \ldots)$, répéter la clé jusqu'à une longueur de $18 * 32$ bits
  - $(P1, P2, \ldots, P18) = (P1, P2, \ldots, P18)\mathrm{XOR}K'$
  - $T = (0x000000, 0x000000)$, texte en clair initial mis en place
  - $T = Blowfish(T)$, application de l'algorithme Blowfish
  - $(P1, P2) = T$, mise-à-jour des deux sous-clés
  - $T = Blowfish(T)$, autre application de Blowfish
  - $(P3, P4) = T$
  - ...
  - $T = Blowfish(T)$
  - $(P17, P18) = T$
  - $T = Blowfish(T)$
  - $(S1[0], S1[1]) = T$
  - $T = Blowfish(T)$
  - $(S1[2], S1[3]) = T$
  - ...
  - $T = Blowfish(T)$
  - $(S1[254], S1[255]) = T$
  - $T = Blowfish(T)$
  - $(S2[0], S2[1]) = T$
  - ...
  - $T = Blowfish(T)$
  - $(S4[254], S4[255]) = T$

Implémentation Java de Blowfish (Markus Hahn)

- Entrées :
  - T – 64 bits de texte en clair
  - $P1$, $P2$, ..., $P18$ – 18 sous-clés
  - $F()$ – fonction d'arrondi
- Sorties : C – 64 bits de texte encodé
- Algorithme :
  - $(L, R) = T$, divisant T dans deux parties de 32 bits
  - $L = L \mathrm{XOR} P1$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P2$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P3$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P4$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P5$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P6$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P7$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P8$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P9$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P10$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P11$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P12$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P13$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P14$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P15$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P16$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P17$
  - $R = R \mathrm{XOR} P18$

Implémentation Java de Blowfish (Markus Hahn)

- Entrées :
  - T – 64 bits de texte en clair
  - $P1, P2, \ldots, P18$ – 18 sous-clés
  - $F()$ – fonction d'arrondi
- Sorties : C – 64 bits de texte encodé
- Algorithme :
  - $(L, R) = T$, divisant T dans deux parties de 32 bits
  - $L = L \mathrm{XOR} P1$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P2$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P3$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P4$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P5$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P6$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P7$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P8$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P9$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P10$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P11$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P12$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P13$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P14$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P15$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P16$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P17$
  - $R = R \mathrm{XOR} P18$

Implémentation Java de Blowfish (Markus Hahn)

- Entrées :
  - T – 64 bits de texte en clair
  - $P1$, $P2$, ..., $P18$ – 18 sous-clés
  - $F()$ – fonction d'arrondi
- Sorties : C – 64 bits de texte encodé
- Algorithme :
  - $(L, R) = T$, divisant T dans deux parties de 32 bits
  - $L = L \mathrm{XOR} P1$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P2$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P3$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P4$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P5$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P6$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P7$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P8$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P9$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P10$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P11$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P12$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P13$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P14$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P15$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P16$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P17$
  - $R = R \mathrm{XOR} P18$

Implémentation Java de Blowfish (Markus Hahn)

- Entrées :
  - T – 64 bits de texte en clair
  - $P1$, $P2$, ..., $P18$ – 18 sous-clés
  - $F()$ – fonction d'arrondi
- Sorties : C – 64 bits de texte encodé
- Algorithme :
  - $(L, R) = T$, divisant T dans deux parties de 32 bits
  - $L = L \mathrm{XOR} P1$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P2$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P3$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P4$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P5$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P6$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P7$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P8$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P9$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P10$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P11$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P12$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P13$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P14$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P15$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P16$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P17$
  - $R = R \mathrm{XOR} P18$

Implémentation Java de Blowfish (Markus Hahn)

- Entrées :
  - T – 64 bits de texte en clair
  - $P1$, $P2$, ..., $P18$ – 18 sous-clés
  - $F()$ – fonction d'arrondi
- Sorties : C – 64 bits de texte encodé
- Algorithme :
  - $(L, R) = T$, divisant T dans deux parties de 32 bits
  - $L = L \mathrm{XOR} P1$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P2$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P3$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P4$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P5$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P6$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P7$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P8$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P9$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P10$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P11$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P12$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P13$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P14$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P15$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P16$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P17$
  - $R = R \mathrm{XOR} P18$

Implémentation Java de Blowfish (Markus Hahn)

- Entrées :
  - T – 64 bits de texte en clair
  - $P1$, $P2$, ..., $P18$ – 18 sous-clés
  - $F()$ – fonction d'arrondi
- Sorties : C – 64 bits de texte encodé
- Algorithme :
  - $(L, R) = T$, divisant T dans deux parties de 32 bits
  - $L = L \mathrm{XOR} P1$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P2$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P3$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P4$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P5$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P6$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P7$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P8$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P9$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P10$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P11$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P12$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P13$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P14$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P15$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P16$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P17$
  - $R = R \mathrm{XOR} P18$

Implémentation Java de Blowfish (Markus Hahn)

- Entrées :
  - T – 64 bits de texte en clair
  - $P1$, $P2$, ..., $P18$ – 18 sous-clés
  - $F()$ – fonction d'arrondi
- Sorties : C – 64 bits de texte encodé
- Algorithme :
  - $(L, R) = T$, divisant T dans deux parties de 32 bits
  - $L = L \mathrm{XOR} P1$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P2$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P3$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P4$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P5$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P6$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P7$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P8$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P9$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P10$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P11$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P12$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P13$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P14$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P15$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P16$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P17$
  - $R = R \mathrm{XOR} P18$

Implémentation Java de Blowfish (Markus Hahn)

- Entrées :
  - T – 64 bits de texte en clair
  - $P1$, $P2$, ..., $P18$ – 18 sous-clés
  - $F()$ – fonction d'arrondi
- Sorties : C – 64 bits de texte encodé
- Algorithme :
  - $(L, R) = T$, divisant T dans deux parties de 32 bits
  - $L = L \mathrm{XOR} P1$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P2$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P3$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P4$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P5$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P6$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P7$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P8$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P9$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P10$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P11$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P12$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P13$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P14$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P15$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P16$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P17$
  - $R = R \mathrm{XOR} P18$

Implémentation Java de Blowfish (Markus Hahn)

- Entrées :
  - T – 64 bits de texte en clair
  - $P1$, $P2$, ..., $P18$ – 18 sous-clés
  - $F()$ – fonction d'arrondi
- Sorties : C – 64 bits de texte encodé
- Algorithme :
  - $(L, R) = T$, divisant T dans deux parties de 32 bits
  - $L = L \mathrm{XOR} P1$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P2$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P3$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P4$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P5$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P6$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P7$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P8$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P9$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P10$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P11$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P12$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P13$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P14$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P15$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P16$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P17$
  - $R = R \mathrm{XOR} P18$

Implémentation Java de Blowfish (Markus Hahn)

- Entrées :
  - T – 64 bits de texte en clair
  - $P1$, $P2$, ..., $P18$ – 18 sous-clés
  - $F()$ – fonction d'arrondi
- Sorties : C – 64 bits de texte encodé
- Algorithme :
  - $(L, R) = T$, divisant T dans deux parties de 32 bits
  - $L = L \mathrm{XOR} P1$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P2$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P3$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P4$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P5$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P6$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P7$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P8$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P9$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P10$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P11$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P12$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P13$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P14$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P15$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P16$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P17$
  - $R = R \mathrm{XOR} P18$

Implémentation Java de Blowfish (Markus Hahn)

- Entrées :
  - T – 64 bits de texte en clair
  - $P1$, $P2$, ..., $P18$ – 18 sous-clés
  - $F()$ – fonction d'arrondi
- Sorties : C – 64 bits de texte encodé
- Algorithme :
  - $(L, R) = T$, divisant T dans deux parties de 32 bits
  - $L = L \mathrm{XOR} P1$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P2$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P3$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P4$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P5$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P6$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P7$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P8$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P9$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P10$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P11$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P12$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P13$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P14$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P15$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P16$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P17$
  - $R = R \mathrm{XOR} P18$

Implémentation Java de Blowfish (Markus Hahn)

- Entrées :
  - T – 64 bits de texte en clair
  - $P1$, $P2$, ..., $P18$ – 18 sous-clés
  - $F()$ – fonction d'arrondi
- Sorties : C – 64 bits de texte encodé
- Algorithme :
  - $(L, R) = T$, divisant T dans deux parties de 32 bits
  - $L = L\mathrm{XOR}P1$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P2$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P3$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P4$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P5$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P6$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P7$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P8$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P9$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P10$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P11$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P12$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P13$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P14$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P15$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P16$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P17$
  - $R = R\mathrm{XOR}P18$

Implémentation Java de Blowfish (Markus Hahn)

- Entrées :
  - T – 64 bits de texte en clair
  - $P1$, $P2$, ..., $P18$ – 18 sous-clés
  - $F()$ – fonction d'arrondi
- Sorties : C – 64 bits de texte encodé
- Algorithme :
  - $(L, R) = T$, divisant T dans deux parties de 32 bits
  - $L = L\mathrm{XOR}P1$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P2$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P3$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P4$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P5$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P6$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P7$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P8$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P9$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P10$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P11$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P12$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P13$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P14$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P15$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P16$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P17$
  - $R = R\mathrm{XOR}P18$

Implémentation Java de Blowfish (Markus Hahn)

- Entrées :
  - T – 64 bits de texte en clair
  - $P1$, $P2$, ..., $P18$ – 18 sous-clés
  - $F()$ – fonction d'arrondi
- Sorties : C – 64 bits de texte encodé
- Algorithme :
  - $(L, R) = T$, divisant T dans deux parties de 32 bits
  - $L = L\mathrm{XOR}P1$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P2$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P3$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P4$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P5$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P6$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P7$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P8$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P9$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P10$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P11$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P12$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P13$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P14$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P15$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P16$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P17$
  - $R = R\mathrm{XOR}P18$

Implémentation Java de Blowfish (Markus Hahn)

- Entrées :
  - T – 64 bits de texte en clair
  - $P1$, $P2$, ..., $P18$ – 18 sous-clés
  - $F()$ – fonction d'arrondi
- Sorties : C – 64 bits de texte encodé
- Algorithme :
  - $(L, R) = T$, divisant T dans deux parties de 32 bits
  - $L = L\mathrm{XOR}P1$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P2$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P3$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P4$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P5$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P6$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P7$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P8$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P9$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P10$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P11$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P12$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P13$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P14$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P15$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P16$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P17$
  - $R = R\mathrm{XOR}P18$

Implémentation Java de Blowfish (Markus Hahn)

- Entrées :
  - T – 64 bits de texte en clair
  - $P1$, $P2$, ..., $P18$ – 18 sous-clés
  - $F()$ – fonction d'arrondi
- Sorties : C – 64 bits de texte encodé
- Algorithme :
  - $(L, R) = T$, divisant T dans deux parties de 32 bits
  - $L = L\mathrm{XOR}P1$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P2$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P3$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P4$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P5$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P6$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P7$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P8$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P9$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P10$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P11$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P12$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P13$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P14$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P15$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P16$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P17$
  - $R = R\mathrm{XOR}P18$

Implémentation Java de Blowfish (Markus Hahn)

- Entrées :
  - T – 64 bits de texte en clair
  - $P1$, $P2$, ..., $P18$ – 18 sous-clés
  - $F()$ – fonction d'arrondi
- Sorties : C – 64 bits de texte encodé
- Algorithme :
  - $(L, R) = T$, divisant T dans deux parties de 32 bits
  - $L = L\mathrm{XOR}P1$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P2$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P3$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P4$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P5$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P6$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P7$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P8$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P9$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P10$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P11$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P12$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P13$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P14$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P15$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P16$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P17$
  - $R = R\mathrm{XOR}P18$

Implémentation Java de Blowfish (Markus Hahn)

- Entrées :
  - T – 64 bits de texte en clair
  - $P1$, $P2$, ..., $P18$ – 18 sous-clés
  - $F()$ – fonction d'arrondi
- Sorties : C – 64 bits de texte encodé
- Algorithme :
  - $(L, R) = T$, divisant T dans deux parties de 32 bits
  - $L = L\mathrm{XOR}P1$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P2$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P3$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P4$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P5$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P6$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P7$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P8$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P9$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P10$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P11$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P12$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P13$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P14$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P15$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P16$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P17$
  - $R = R\mathrm{XOR}P18$

Implémentation Java de Blowfish (Markus Hahn)

- Entrées :
  - T – 64 bits de texte en clair
  - $P1$, $P2$, ..., $P18$ – 18 sous-clés
  - $F()$ – fonction d'arrondi
- Sorties : C – 64 bits de texte encodé
- Algorithme :
  - $(L, R) = T$, divisant T dans deux parties de 32 bits
  - $L = L\mathrm{XOR}P1$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P2$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P3$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P4$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P5$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P6$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P7$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P8$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P9$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P10$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P11$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P12$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P13$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P14$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P15$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P16$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P17$
  - $R = R\mathrm{XOR}P18$

Implémentation Java de Blowfish (Markus Hahn)

- Entrées :
  - T – 64 bits de texte en clair
  - $P1$, $P2$, ..., $P18$ – 18 sous-clés
  - $F()$ – fonction d'arrondi
- Sorties : C – 64 bits de texte encodé
- Algorithme :
  - $(L, R) = T$, divisant T dans deux parties de 32 bits
  - $L = L\mathrm{XOR}P1$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P2$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P3$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P4$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P5$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P6$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P7$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P8$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P9$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P10$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P11$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P12$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P13$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P14$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P15$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P16$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P17$
  - $R = R\mathrm{XOR}P18$

Implémentation Java de Blowfish (Markus Hahn)

- Entrées :
  - T – 64 bits de texte en clair
  - $P1$, $P2$, ..., $P18$ – 18 sous-clés
  - $F()$ – fonction d'arrondi
- Sorties : C – 64 bits de texte encodé
- Algorithme :
  - $(L, R) = T$, divisant T dans deux parties de 32 bits
  - $L = L \mathrm{XOR} P1$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P2$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P3$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P4$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P5$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P6$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P7$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P8$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P9$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P10$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P11$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P12$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P13$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P14$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P15$
  - $R = R \mathrm{XOR} F(L) \mathrm{XOR} P16$
  - $L = L \mathrm{XOR} F(R) \mathrm{XOR} P17$
  - $R = R \mathrm{XOR} P18$

Implémentation Java de Blowfish (Markus Hahn)

- Entrées :
  - T – 64 bits de texte en clair
  - $P1$, $P2$, ..., $P18$ – 18 sous-clés
  - $F()$ – fonction d'arrondi
- Sorties : C – 64 bits de texte encodé
- Algorithme :
  - $(L, R) = T$, divisant T dans deux parties de 32 bits
  - $L = L\mathrm{XOR}P1$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P2$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P3$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P4$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P5$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P6$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P7$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P8$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P9$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P10$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P11$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P12$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P13$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P14$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P15$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P16$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P17$
  - $R = R\mathrm{XOR}P18$

Implémentation Java de Blowfish (Markus Hahn)

- Entrées :
  - T – 64 bits de texte en clair
  - $P1$, $P2$, ..., $P18$ – 18 sous-clés
  - $F()$ – fonction d'arrondi
- Sorties : C – 64 bits de texte encodé
- Algorithme :
  - $(L, R) = T$, divisant T dans deux parties de 32 bits
  - $L = L\mathrm{XOR}P1$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P2$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P3$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P4$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P5$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P6$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P7$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P8$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P9$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P10$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P11$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P12$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P13$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P14$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P15$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P16$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P17$
  - $R = R\mathrm{XOR}P18$

Implémentation Java de Blowfish (Markus Hahn)

- Entrées :
  - T – 64 bits de texte en clair
  - $P1$, $P2$, ..., $P18$ – 18 sous-clés
  - $F()$ – fonction d'arrondi
- Sorties : C – 64 bits de texte encodé
- Algorithme :
  - $(L, R) = T$, divisant T dans deux parties de 32 bits
  - $L = L\mathrm{XOR}P1$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P2$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P3$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P4$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P5$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P6$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P7$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P8$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P9$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P10$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P11$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P12$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P13$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P14$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P15$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P16$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P17$
  - $R = R\mathrm{XOR}P18$

Implémentation Java de Blowfish (Markus Hahn)

- Entrées :
  - T – 64 bits de texte en clair
  - $P1$, $P2$, ..., $P18$ – 18 sous-clés
  - $F()$ – fonction d'arrondi
- Sorties : C – 64 bits de texte encodé
- Algorithme :
  - $(L, R) = T$, divisant T dans deux parties de 32 bits
  - $L = L\mathrm{XOR}P1$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P2$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P3$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P4$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P5$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P6$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P7$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P8$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P9$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P10$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P11$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P12$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P13$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P14$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P15$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P16$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P17$
  - $R = R\mathrm{XOR}P18$

Implémentation Java de Blowfish (Markus Hahn)

- Entrées :
  - T – 64 bits de texte en clair
  - $P1$, $P2$, ..., $P18$ – 18 sous-clés
  - $F()$ – fonction d'arrondi
- Sorties : C – 64 bits de texte encodé
- Algorithme :
  - $(L, R) = T$, divisant T dans deux parties de 32 bits
  - $L = L\mathrm{XOR}P1$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P2$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P3$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P4$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P5$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P6$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P7$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P8$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P9$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P10$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P11$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P12$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P13$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P14$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P15$
  - $R = R\mathrm{XOR}F(L)\mathrm{XOR}P16$
  - $L = L\mathrm{XOR}F(R)\mathrm{XOR}P17$
  - $R = R\mathrm{XOR}P18$

L'effet de l'avalanche parfaite

- La fonction non-inversible choisie F
  - $\Longrightarrow$ le meilleur effet possible d'avalanche pour un réseau Feistel après trois rondes
  - $\Longrightarrow$ et à nouveau deux rondes par la ensuite
- Pourquoi ?
  - chaque bit de texte de la moitié gauche d'un tour affecte chaque bit de texte de la moitié droite.
  - effet d'avalanche – après chaque tour entre la clé et la moitié droite du texte, par la construction de F
- la limite de 448 bits a été fixée de façon à ce que chaque bit de chaque valeur du tableau P et des S-boîtes dépende de tous les bits de la clé, les quatre dernières valeurs du tableau P n'affectant pas tous les bits du bloc chiffré

L'effet de l'avalanche parfaite

- La fonction non-inversible choisie F
  - $\Longrightarrow$ le meilleur effet possible d'avalanche pour un réseau Feistel après trois rondes
  - $\Longrightarrow$ et à nouveau deux rondes par la ensuite
- Pourquoi ?
  - chaque bit de texte de la moitié gauche d'un tour affecte chaque bit de texte de la moitié droite.
  - effet d'avalanche – après chaque tour entre la clé et la moitié droite du texte, par la construction de F
- la limite de 448 bits a été fixée de façon à ce que chaque bit de chaque valeur du tableau P et des S-boîtes dépende de tous les bits de la clé, les quatre dernières valeurs du tableau P n'affectant pas tous les bits du bloc chiffré

L'effet de l'avalanche parfaite

- La fonction non-inversible choisie F
  - $\Longrightarrow$ le meilleur effet possible d'avalanche pour un réseau Feistel après trois rondes
  - $\Longrightarrow$ et à nouveau deux rondes par la ensuite
- Pourquoi ?
  - chaque bit de texte de la moitié gauche d'un tour affecte chaque bit de texte de la moitié droite.
  - effet d'avalanche – après chaque tour entre la clé et la moitié droite du texte, par la construction de F
- la limite de 448 bits a été fixée de façon à ce que chaque bit de chaque valeur du tableau P et des S-boîtes dépende de tous les bits de la clé, les quatre dernières valeurs du tableau P n'affectant pas tous les bits du bloc chiffré

L'effet de l'avalanche parfaite

- La fonction non-inversible choisie F
  - $\Longrightarrow$ le meilleur effet possible d'avalanche pour un réseau Feistel après trois rondes
  - $\Longrightarrow$ et à nouveau deux rondes par la ensuite
- Pourquoi ?
  - chaque bit de texte de la moitié gauche d'un tour affecte chaque bit de texte de la moitié droite.
  - effet d'avalanche – après chaque tour entre la clé et la moitié droite du texte, par la construction de F
- la limite de 448 bits a été fixée de façon à ce que chaque bit de chaque valeur du tableau P et des S-boîtes dépende de tous les bits de la clé, les quatre dernières valeurs du tableau P n'affectant pas tous les bits du bloc chiffré

L'effet de l'avalanche parfaite

- La fonction non-inversible choisie F
  - $\Longrightarrow$ le meilleur effet possible d'avalanche pour un réseau Feistel après trois rondes
  - $\Longrightarrow$ et à nouveau deux rondes par la ensuite
- Pourquoi ?
  - chaque bit de texte de la moitié gauche d'un tour affecte chaque bit de texte de la moitié droite.
  - effet d'avalanche – après chaque tour entre la clé et la moitié droite du texte, par la construction de F
- la limite de 448 bits a été fixée de façon à ce que chaque bit de chaque valeur du tableau P et des S-boîtes dépende de tous les bits de la clé, les quatre dernières valeurs du tableau P n'affectant pas tous les bits du bloc chiffré

L'effet de l'avalanche parfaite

- La fonction non-inversible choisie F
  - $\Longrightarrow$ le meilleur effet possible d'avalanche pour un réseau Feistel après trois rondes
  - $\Longrightarrow$ et à nouveau deux rondes par la ensuite
- Pourquoi ?
  - chaque bit de texte de la moitié gauche d'un tour affecte chaque bit de texte de la moitié droite.
  - effet d'avalanche – après chaque tour entre la clé et la moitié droite du texte, par la construction de F
- la limite de 448 bits a été fixée de façon à ce que chaque bit de chaque valeur du tableau P et des S-boîtes dépende de tous les bits de la clé, les quatre dernières valeurs du tableau P n'affectant pas tous les bits du bloc chiffré

L'effet de l'avalanche parfaite

- La fonction non-inversible choisie F
  - $\Longrightarrow$ le meilleur effet possible d'avalanche pour un réseau Feistel après trois rondes
  - $\Longrightarrow$ et à nouveau deux rondes par la ensuite
- Pourquoi ?
  - chaque bit de texte de la moitié gauche d'un tour affecte chaque bit de texte de la moitié droite.
  - effet d'avalanche – après chaque tour entre la clé et la moitié droite du texte, par la construction de F
- la limite de 448 bits a été fixée de faọn à ce que chaque bit de chaque valeur du tableau P et des S-boîtes dépende de tous les bits de la clé, les quatre dernières valeurs du tableau P n'affectant pas tous les bits du bloc chiffré

Algorithme de décryptage

- Entrées :
  - CC – 64 bits of cipher text
  - $P1$, $P2$, ..., $P18$: 18 sous-clés
  - $F()$ – Round function
- Sortie : TT – 64 bits de texte en clair
- Algorithme :
  - $(LL0, RR0) = CC$, divisant CC dans deux parties de 32 bits
  - $LL1 = LL0 \mathrm{XOR} P18$
  - $RR2 = RR0 \mathrm{XOR} F(LL1) \mathrm{XOR} P17$
  - $LL3 = LL1 \mathrm{XOR} F(RR2) \mathrm{XOR} P16$
  - $RR4 = RR2 \mathrm{XOR} F(LL3) \mathrm{XOR} P15$
  - $LL5 = LL3 \mathrm{XOR} F(RR4) \mathrm{XOR} P14$
  - $RR6 = RR4 \mathrm{XOR} F(LL5) \mathrm{XOR} P13$
  - $LL7 = LL5 \mathrm{XOR} F(RR6) \mathrm{XOR} P12$
  - $RR8 = RR6 \mathrm{XOR} F(LL7) \mathrm{XOR} P11$
  - $LL9 = LL7 \mathrm{XOR} F(RR8) \mathrm{XOR} P10$
  - $RR10 = RR8 \mathrm{XOR} F(LL9) \mathrm{XOR} P9$
  - $LL11 = LL9 \mathrm{XOR} F(RR10) \mathrm{XOR} P8$
  - $RR12 = RR10 \mathrm{XOR} F(LL11) \mathrm{XOR} P7$
  - $LL13 = LL11 \mathrm{XOR} F(RR12) \mathrm{XOR} P6$
  - $RR14 = RR12 \mathrm{XOR} F(LL13) \mathrm{XOR} P5$
  - $LL15 = LL13 \mathrm{XOR} F(RR14) \mathrm{XOR} P4$
  - $RR16 = RR14 \mathrm{XOR} F(LL15) \mathrm{XOR} P3$
  - $LL17 = LL15 \mathrm{XOR} F(RR16) \mathrm{XOR} P2$
  - $RR18 = R16 \mathrm{XOR} P1$

Algorithme de décryptage

- Entrées :
  - CC – 64 bits of cipher text
  - $P1$, $P2$, ..., $P18$: 18 sous-clés
  - $F()$ – Round function
- Sortie : TT – 64 bits de texte en clair
- Algorithme :
  - $(LL0, RR0) = CC$, divisant CC dans deux parties de 32 bits
  - $LL1 = LL0 \mathrm{XOR} P18$
  - $RR2 = RR0 \mathrm{XOR} F(LL1) \mathrm{XOR} P17$
  - $LL3 = LL1 \mathrm{XOR} F(RR2) \mathrm{XOR} P16$
  - $RR4 = RR2 \mathrm{XOR} F(LL3) \mathrm{XOR} P15$
  - $LL5 = LL3 \mathrm{XOR} F(RR4) \mathrm{XOR} P14$
  - $RR6 = RR4 \mathrm{XOR} F(LL5) \mathrm{XOR} P13$
  - $LL7 = LL5 \mathrm{XOR} F(RR6) \mathrm{XOR} P12$
  - $RR8 = RR6 \mathrm{XOR} F(LL7) \mathrm{XOR} P11$
  - $LL9 = LL7 \mathrm{XOR} F(RR8) \mathrm{XOR} P10$
  - $RR10 = RR8 \mathrm{XOR} F(LL9) \mathrm{XOR} P9$
  - $LL11 = LL9 \mathrm{XOR} F(RR10) \mathrm{XOR} P8$
  - $RR12 = RR10 \mathrm{XOR} F(LL11) \mathrm{XOR} P7$
  - $LL13 = LL11 \mathrm{XOR} F(RR12) \mathrm{XOR} P6$
  - $RR14 = RR12 \mathrm{XOR} F(LL13) \mathrm{XOR} P5$
  - $LL15 = LL13 \mathrm{XOR} F(RR14) \mathrm{XOR} P4$
  - $RR16 = RR14 \mathrm{XOR} F(LL15) \mathrm{XOR} P3$
  - $LL17 = LL15 \mathrm{XOR} F(RR16) \mathrm{XOR} P2$
  - $RR18 = R16 \mathrm{XOR} P1$

Algorithme de décryptage

- Entrées :
  - CC – 64 bits of cipher text
  - $P1$, $P2$, ..., $P18$: 18 sous-clés
  - $F()$ – Round function
- Sortie : TT – 64 bits de texte en clair
- Algorithme :
  - $(LL0, RR0) = CC$, divisant CC dans deux parties de 32 bits
  - $LL1 = LL0 \mathrm{XOR} P18$
  - $RR2 = RR0 \mathrm{XOR} F(LL1) \mathrm{XOR} P17$
  - $LL3 = LL1 \mathrm{XOR} F(RR2) \mathrm{XOR} P16$
  - $RR4 = RR2 \mathrm{XOR} F(LL3) \mathrm{XOR} P15$
  - $LL5 = LL3 \mathrm{XOR} F(RR4) \mathrm{XOR} P14$
  - $RR6 = RR4 \mathrm{XOR} F(LL5) \mathrm{XOR} P13$
  - $LL7 = LL5 \mathrm{XOR} F(RR6) \mathrm{XOR} P12$
  - $RR8 = RR6 \mathrm{XOR} F(LL7) \mathrm{XOR} P11$
  - $LL9 = LL7 \mathrm{XOR} F(RR8) \mathrm{XOR} P10$
  - $RR10 = RR8 \mathrm{XOR} F(LL9) \mathrm{XOR} P9$
  - $LL11 = LL9 \mathrm{XOR} F(RR10) \mathrm{XOR} P8$
  - $RR12 = RR10 \mathrm{XOR} F(LL11) \mathrm{XOR} P7$
  - $LL13 = LL11 \mathrm{XOR} F(RR12) \mathrm{XOR} P6$
  - $RR14 = RR12 \mathrm{XOR} F(LL13) \mathrm{XOR} P5$
  - $LL15 = LL13 \mathrm{XOR} F(RR14) \mathrm{XOR} P4$
  - $RR16 = RR14 \mathrm{XOR} F(LL15) \mathrm{XOR} P3$
  - $LL17 = LL15 \mathrm{XOR} F(RR16) \mathrm{XOR} P2$
  - $RR18 = R16 \mathrm{XOR} P1$

Algorithme de décryptage

- Entrées :
  - CC – 64 bits of cipher text
  - $P1$, $P2$, ..., $P18$: 18 sous-clés
  - $F()$ – Round function
- Sortie : TT – 64 bits de texte en clair
- Algorithme :
  - $(LL0, RR0) = CC$, divisant CC dans deux parties de 32 bits
  - $LL1 = LL0 \mathrm{XOR} P18$
  - $RR2 = RR0 \mathrm{XOR} F(LL1) \mathrm{XOR} P17$
  - $LL3 = LL1 \mathrm{XOR} F(RR2) \mathrm{XOR} P16$
  - $RR4 = RR2 \mathrm{XOR} F(LL3) \mathrm{XOR} P15$
  - $LL5 = LL3 \mathrm{XOR} F(RR4) \mathrm{XOR} P14$
  - $RR6 = RR4 \mathrm{XOR} F(LL5) \mathrm{XOR} P13$
  - $LL7 = LL5 \mathrm{XOR} F(RR6) \mathrm{XOR} P12$
  - $RR8 = RR6 \mathrm{XOR} F(LL7) \mathrm{XOR} P11$
  - $LL9 = LL7 \mathrm{XOR} F(RR8) \mathrm{XOR} P10$
  - $RR10 = RR8 \mathrm{XOR} F(LL9) \mathrm{XOR} P9$
  - $LL11 = LL9 \mathrm{XOR} F(RR10) \mathrm{XOR} P8$
  - $RR12 = RR10 \mathrm{XOR} F(LL11) \mathrm{XOR} P7$
  - $LL13 = LL11 \mathrm{XOR} F(RR12) \mathrm{XOR} P6$
  - $RR14 = RR12 \mathrm{XOR} F(LL13) \mathrm{XOR} P5$
  - $LL15 = LL13 \mathrm{XOR} F(RR14) \mathrm{XOR} P4$
  - $RR16 = RR14 \mathrm{XOR} F(LL15) \mathrm{XOR} P3$
  - $LL17 = LL15 \mathrm{XOR} F(RR16) \mathrm{XOR} P2$
  - $RR18 = R16 \mathrm{XOR} P1$

Algorithme de décryptage

- Entrées :
  - CC – 64 bits of cipher text
  - $P1$, $P2$, ..., $P18$: 18 sous-clés
  - $F()$ – Round function
- Sortie : TT – 64 bits de texte en clair
- Algorithme :
  - $(LL0, RR0) = CC$, divisant CC dans deux parties de 32 bits
  - $LL1 = LL0 \mathrm{XOR} P18$
  - $RR2 = RR0 \mathrm{XOR} F(LL1) \mathrm{XOR} P17$
  - $LL3 = LL1 \mathrm{XOR} F(RR2) \mathrm{XOR} P16$
  - $RR4 = RR2 \mathrm{XOR} F(LL3) \mathrm{XOR} P15$
  - $LL5 = LL3 \mathrm{XOR} F(RR4) \mathrm{XOR} P14$
  - $RR6 = RR4 \mathrm{XOR} F(LL5) \mathrm{XOR} P13$
  - $LL7 = LL5 \mathrm{XOR} F(RR6) \mathrm{XOR} P12$
  - $RR8 = RR6 \mathrm{XOR} F(LL7) \mathrm{XOR} P11$
  - $LL9 = LL7 \mathrm{XOR} F(RR8) \mathrm{XOR} P10$
  - $RR10 = RR8 \mathrm{XOR} F(LL9) \mathrm{XOR} P9$
  - $LL11 = LL9 \mathrm{XOR} F(RR10) \mathrm{XOR} P8$
  - $RR12 = RR10 \mathrm{XOR} F(LL11) \mathrm{XOR} P7$
  - $LL13 = LL11 \mathrm{XOR} F(RR12) \mathrm{XOR} P6$
  - $RR14 = RR12 \mathrm{XOR} F(LL13) \mathrm{XOR} P5$
  - $LL15 = LL13 \mathrm{XOR} F(RR14) \mathrm{XOR} P4$
  - $RR16 = RR14 \mathrm{XOR} F(LL15) \mathrm{XOR} P3$
  - $LL17 = LL15 \mathrm{XOR} F(RR16) \mathrm{XOR} P2$
  - $RR18 = P16 \mathrm{XOR} P1$

Algorithme de décryptage

- Entrées :
  - CC – 64 bits of cipher text
  - $P1$, $P2$, ..., $P18$: 18 sous-clés
  - $F()$ – Round function
- Sortie : TT – 64 bits de texte en clair
- Algorithme :
  - $(LL0, RR0) = CC$, divisant CC dans deux parties de 32 bits
  - $LL1 = LL0 \mathrm{XOR} P18$
  - $RR2 = RR0 \mathrm{XOR} F(LL1) \mathrm{XOR} P17$
  - $LL3 = LL1 \mathrm{XOR} F(RR2) \mathrm{XOR} P16$
  - $RR4 = RR2 \mathrm{XOR} F(LL3) \mathrm{XOR} P15$
  - $LL5 = LL3 \mathrm{XOR} F(RR4) \mathrm{XOR} P14$
  - $RR6 = RR4 \mathrm{XOR} F(LL5) \mathrm{XOR} P13$
  - $LL7 = LL5 \mathrm{XOR} F(RR6) \mathrm{XOR} P12$
  - $RR8 = RR6 \mathrm{XOR} F(LL7) \mathrm{XOR} P11$
  - $LL9 = LL7 \mathrm{XOR} F(RR8) \mathrm{XOR} P10$
  - $RR10 = RR8 \mathrm{XOR} F(LL9) \mathrm{XOR} P9$
  - $LL11 = LL9 \mathrm{XOR} F(RR10) \mathrm{XOR} P8$
  - $RR12 = RR10 \mathrm{XOR} F(LL11) \mathrm{XOR} P7$
  - $LL13 = LL11 \mathrm{XOR} F(RR12) \mathrm{XOR} P6$
  - $RR14 = RR12 \mathrm{XOR} F(LL13) \mathrm{XOR} P5$
  - $LL15 = LL13 \mathrm{XOR} F(RR14) \mathrm{XOR} P4$
  - $RR16 = RR14 \mathrm{XOR} F(LL15) \mathrm{XOR} P3$
  - $LL17 = LL15 \mathrm{XOR} F(RR16) \mathrm{XOR} P2$
  - $RR18 = R16 \mathrm{XOR} P1$

Algorithme de décryptage

- Entrées :
  - CC – 64 bits of cipher text
  - $P1$, $P2$, ..., $P18$: 18 sous-clés
  - $F()$ – Round function
- Sortie : TT – 64 bits de texte en clair
- Algorithme :
  - $(LL0, RR0) = CC$, divisant CC dans deux parties de 32 bits
  - $LL1 = LL0 \mathrm{XOR} P18$
  - $RR2 = RR0 \mathrm{XOR} F(LL1) \mathrm{XOR} P17$
  - $LL3 = LL1 \mathrm{XOR} F(RR2) \mathrm{XOR} P16$
  - $RR4 = RR2 \mathrm{XOR} F(LL3) \mathrm{XOR} P15$
  - $LL5 = LL3 \mathrm{XOR} F(RR4) \mathrm{XOR} P14$
  - $RR6 = RR4 \mathrm{XOR} F(LL5) \mathrm{XOR} P13$
  - $LL7 = LL5 \mathrm{XOR} F(RR6) \mathrm{XOR} P12$
  - $RR8 = RR6 \mathrm{XOR} F(LL7) \mathrm{XOR} P11$
  - $LL9 = LL7 \mathrm{XOR} F(RR8) \mathrm{XOR} P10$
  - $RR10 = RR8 \mathrm{XOR} F(LL9) \mathrm{XOR} P9$
  - $LL11 = LL9 \mathrm{XOR} F(RR10) \mathrm{XOR} P8$
  - $RR12 = RR10 \mathrm{XOR} F(LL11) \mathrm{XOR} P7$
  - $LL13 = LL11 \mathrm{XOR} F(RR12) \mathrm{XOR} P6$
  - $RR14 = RR12 \mathrm{XOR} F(LL13) \mathrm{XOR} P5$
  - $LL15 = LL13 \mathrm{XOR} F(RR14) \mathrm{XOR} P4$
  - $RR16 = RR14 \mathrm{XOR} F(LL15) \mathrm{XOR} P3$
  - $LL17 = LL15 \mathrm{XOR} F(RR16) \mathrm{XOR} P2$
  - $RR18 = R16 \mathrm{XOR} P1$

Algorithme de décryptage

- Entrées :
  - CC – 64 bits of cipher text
  - $P1$, $P2$, ..., $P18$: 18 sous-clés
  - $F()$ – Round function
- Sortie : TT – 64 bits de texte en clair
- Algorithme :
  - $(LL0, RR0) = CC$, divisant CC dans deux parties de 32 bits
  - $LL1 = LL0 \mathrm{XOR} P18$
  - $RR2 = RR0 \mathrm{XOR} F(LL1) \mathrm{XOR} P17$
  - $LL3 = LL1 \mathrm{XOR} F(RR2) \mathrm{XOR} P16$
  - $RR4 = RR2 \mathrm{XOR} F(LL3) \mathrm{XOR} P15$
  - $LL5 = LL3 \mathrm{XOR} F(RR4) \mathrm{XOR} P14$
  - $RR6 = RR4 \mathrm{XOR} F(LL5) \mathrm{XOR} P13$
  - $LL7 = LL5 \mathrm{XOR} F(RR6) \mathrm{XOR} P12$
  - $RR8 = RR6 \mathrm{XOR} F(LL7) \mathrm{XOR} P11$
  - $LL9 = LL7 \mathrm{XOR} F(RR8) \mathrm{XOR} P10$
  - $RR10 = RR8 \mathrm{XOR} F(LL9) \mathrm{XOR} P9$
  - $LL11 = LL9 \mathrm{XOR} F(RR10) \mathrm{XOR} P8$
  - $RR12 = RR10 \mathrm{XOR} F(LL11) \mathrm{XOR} P7$
  - $LL13 = LL11 \mathrm{XOR} F(RR12) \mathrm{XOR} P6$
  - $RR14 = RR12 \mathrm{XOR} F(LL13) \mathrm{XOR} P5$
  - $LL15 = LL13 \mathrm{XOR} F(RR14) \mathrm{XOR} P4$
  - $RR16 = RR14 \mathrm{XOR} F(LL15) \mathrm{XOR} P3$
  - $LL17 = LL15 \mathrm{XOR} F(RR16) \mathrm{XOR} P2$
  - $RR18 = R16 \mathrm{XOR} P1$

Algorithme de décryptage

- Entrées :
  - CC – 64 bits of cipher text
  - $P1$, $P2$, ..., $P18$: 18 sous-clés
  - $F()$ – Round function
- Sortie : TT – 64 bits de texte en clair
- Algorithme :
  - $(LL0, RR0) = CC$, divisant CC dans deux parties de 32 bits
  - $LL1 = LL0 \mathrm{XOR} P18$
  - $RR2 = RR0 \mathrm{XOR} F(LL1) \mathrm{XOR} P17$
  - $LL3 = LL1 \mathrm{XOR} F(RR2) \mathrm{XOR} P16$
  - $RR4 = RR2 \mathrm{XOR} F(LL3) \mathrm{XOR} P15$
  - $LL5 = LL3 \mathrm{XOR} F(RR4) \mathrm{XOR} P14$
  - $RR6 = RR4 \mathrm{XOR} F(LL5) \mathrm{XOR} P13$
  - $LL7 = LL5 \mathrm{XOR} F(RR6) \mathrm{XOR} P12$
  - $RR8 = RR6 \mathrm{XOR} F(LL7) \mathrm{XOR} P11$
  - $LL9 = LL7 \mathrm{XOR} F(RR8) \mathrm{XOR} P10$
  - $RR10 = RR8 \mathrm{XOR} F(LL9) \mathrm{XOR} P9$
  - $LL11 = LL9 \mathrm{XOR} F(RR10) \mathrm{XOR} P8$
  - $RR12 = RR10 \mathrm{XOR} F(LL11) \mathrm{XOR} P7$
  - $LL13 = LL11 \mathrm{XOR} F(RR12) \mathrm{XOR} P6$
  - $RR14 = RR12 \mathrm{XOR} F(LL13) \mathrm{XOR} P5$
  - $LL15 = LL13 \mathrm{XOR} F(RR14) \mathrm{XOR} P4$
  - $RR16 = RR14 \mathrm{XOR} F(LL15) \mathrm{XOR} P3$
  - $LL17 = LL15 \mathrm{XOR} F(RR16) \mathrm{XOR} P2$
  - $RR18 = R16 \mathrm{XOR} P1$

Algorithme de décryptage

- Entrées :
  - CC – 64 bits of cipher text
  - $P1$, $P2$, ..., $P18$: 18 sous-clés
  - $F()$ – Round function
- Sortie : TT – 64 bits de texte en clair
- Algorithme :
  - $(LL0, RR0) = CC$, divisant CC dans deux parties de 32 bits
  - $LL1 = LL0 \mathrm{XOR} P18$
  - $RR2 = RR0 \mathrm{XOR} F(LL1) \mathrm{XOR} P17$
  - $LL3 = LL1 \mathrm{XOR} F(RR2) \mathrm{XOR} P16$
  - $RR4 = RR2 \mathrm{XOR} F(LL3) \mathrm{XOR} P15$
  - $LL5 = LL3 \mathrm{XOR} F(RR4) \mathrm{XOR} P14$
  - $RR6 = RR4 \mathrm{XOR} F(LL5) \mathrm{XOR} P13$
  - $LL7 = LL5 \mathrm{XOR} F(RR6) \mathrm{XOR} P12$
  - $RR8 = RR6 \mathrm{XOR} F(LL7) \mathrm{XOR} P11$
  - $LL9 = LL7 \mathrm{XOR} F(RR8) \mathrm{XOR} P10$
  - $RR10 = RR8 \mathrm{XOR} F(LL9) \mathrm{XOR} P9$
  - $LL11 = LL9 \mathrm{XOR} F(RR10) \mathrm{XOR} P8$
  - $RR12 = RR10 \mathrm{XOR} F(LL11) \mathrm{XOR} P7$
  - $LL13 = LL11 \mathrm{XOR} F(RR12) \mathrm{XOR} P6$
  - $RR14 = RR12 \mathrm{XOR} F(LL13) \mathrm{XOR} P5$
  - $LL15 = LL13 \mathrm{XOR} F(RR14) \mathrm{XOR} P4$
  - $RR16 = RR14 \mathrm{XOR} F(LL15) \mathrm{XOR} P3$
  - $LL17 = LL15 \mathrm{XOR} F(RR16) \mathrm{XOR} P2$
  - $RR18 = R16 \mathrm{XOR} P1$

Algorithme de décryptage

- Entrées :
  - CC – 64 bits of cipher text
  - $P1$, $P2$, ..., $P18$: 18 sous-clés
  - $F()$ – Round function
- Sortie : TT – 64 bits de texte en clair
- Algorithme :
  - $(LL0, RR0) = CC$, divisant CC dans deux parties de 32 bits
  - $LL1 = LL0 \mathrm{XOR} P18$
  - $RR2 = RR0 \mathrm{XOR} F(LL1) \mathrm{XOR} P17$
  - $LL3 = LL1 \mathrm{XOR} F(RR2) \mathrm{XOR} P16$
  - $RR4 = RR2 \mathrm{XOR} F(LL3) \mathrm{XOR} P15$
  - $LL5 = LL3 \mathrm{XOR} F(RR4) \mathrm{XOR} P14$
  - $RR6 = RR4 \mathrm{XOR} F(LL5) \mathrm{XOR} P13$
  - $LL7 = LL5 \mathrm{XOR} F(RR6) \mathrm{XOR} P12$
  - $RR8 = RR6 \mathrm{XOR} F(LL7) \mathrm{XOR} P11$
  - $LL9 = LL7 \mathrm{XOR} F(RR8) \mathrm{XOR} P10$
  - $RR10 = RR8 \mathrm{XOR} F(LL9) \mathrm{XOR} P9$
  - $LL11 = LL9 \mathrm{XOR} F(RR10) \mathrm{XOR} P8$
  - $RR12 = RR10 \mathrm{XOR} F(LL11) \mathrm{XOR} P7$
  - $LL13 = LL11 \mathrm{XOR} F(RR12) \mathrm{XOR} P6$
  - $RR14 = RR12 \mathrm{XOR} F(LL13) \mathrm{XOR} P5$
  - $LL15 = LL13 \mathrm{XOR} F(RR14) \mathrm{XOR} P4$
  - $RR16 = RR14 \mathrm{XOR} F(LL15) \mathrm{XOR} P3$
  - $LL17 = LL15 \mathrm{XOR} F(RR16) \mathrm{XOR} P2$
  - $RR18 = R16 \mathrm{XOR} P1$

Algorithme de décryptage

- Entrées :
  - CC – 64 bits of cipher text
  - $P1$, $P2$, ..., $P18$: 18 sous-clés
  - $F()$ – Round function
- Sortie : TT – 64 bits de texte en clair
- Algorithme :
  - $(LL0, RR0) = CC$, divisant CC dans deux parties de 32 bits
  - $LL1 = LL0 \mathrm{XOR} P18$
  - $RR2 = RR0 \mathrm{XOR} F(LL1) \mathrm{XOR} P17$
  - $LL3 = LL1 \mathrm{XOR} F(RR2) \mathrm{XOR} P16$
  - $RR4 = RR2 \mathrm{XOR} F(LL3) \mathrm{XOR} P15$
  - $LL5 = LL3 \mathrm{XOR} F(RR4) \mathrm{XOR} P14$
  - $RR6 = RR4 \mathrm{XOR} F(LL5) \mathrm{XOR} P13$
  - $LL7 = LL5 \mathrm{XOR} F(RR6) \mathrm{XOR} P12$
  - $RR8 = RR6 \mathrm{XOR} F(LL7) \mathrm{XOR} P11$
  - $LL9 = LL7 \mathrm{XOR} F(RR8) \mathrm{XOR} P10$
  - $RR10 = RR8 \mathrm{XOR} F(LL9) \mathrm{XOR} P9$
  - $LL11 = LL9 \mathrm{XOR} F(RR10) \mathrm{XOR} P8$
  - $RR12 = RR10 \mathrm{XOR} F(LL11) \mathrm{XOR} P7$
  - $LL13 = LL11 \mathrm{XOR} F(RR12) \mathrm{XOR} P6$
  - $RR14 = RR12 \mathrm{XOR} F(LL13) \mathrm{XOR} P5$
  - $LL15 = LL13 \mathrm{XOR} F(RR14) \mathrm{XOR} P4$
  - $RR16 = RR14 \mathrm{XOR} F(LL15) \mathrm{XOR} P3$
  - $LL17 = LL15 \mathrm{XOR} F(RR16) \mathrm{XOR} P2$
  - $RR18 = R16 \mathrm{XOR} P1$

Algorithme de décryptage

- Entrées :
  - CC – 64 bits of cipher text
  - $P1$, $P2$, ..., $P18$: 18 sous-clés
  - $F()$ – Round function
- Sortie : TT – 64 bits de texte en clair
- Algorithme :
  - $(LL0, RR0) = CC$, divisant CC dans deux parties de 32 bits
  - $LL1 = LL0 \mathrm{XOR} P18$
  - $RR2 = RR0 \mathrm{XOR} F(LL1) \mathrm{XOR} P17$
  - $LL3 = LL1 \mathrm{XOR} F(RR2) \mathrm{XOR} P16$
  - $RR4 = RR2 \mathrm{XOR} F(LL3) \mathrm{XOR} P15$
  - $LL5 = LL3 \mathrm{XOR} F(RR4) \mathrm{XOR} P14$
  - $RR6 = RR4 \mathrm{XOR} F(LL5) \mathrm{XOR} P13$
  - $LL7 = LL5 \mathrm{XOR} F(RR6) \mathrm{XOR} P12$
  - $RR8 = RR6 \mathrm{XOR} F(LL7) \mathrm{XOR} P11$
  - $LL9 = LL7 \mathrm{XOR} F(RR8) \mathrm{XOR} P10$
  - $RR10 = RR8 \mathrm{XOR} F(LL9) \mathrm{XOR} P9$
  - $LL11 = LL9 \mathrm{XOR} F(RR10) \mathrm{XOR} P8$
  - $RR12 = RR10 \mathrm{XOR} F(LL11) \mathrm{XOR} P7$
  - $LL13 = LL11 \mathrm{XOR} F(RR12) \mathrm{XOR} P6$
  - $RR14 = RR12 \mathrm{XOR} F(LL13) \mathrm{XOR} P5$
  - $LL15 = LL13 \mathrm{XOR} F(RR14) \mathrm{XOR} P4$
  - $RR16 = RR14 \mathrm{XOR} F(LL15) \mathrm{XOR} P3$
  - $LL17 = LL15 \mathrm{XOR} F(RR16) \mathrm{XOR} P2$
  - $RR18 = R16 \mathrm{XOR} P1$

Algorithme de décryptage

- Entrées :
  - CC – 64 bits of cipher text
  - $P1$, $P2$, ..., $P18$: 18 sous-clés
  - $F()$ – Round function
- Sortie : TT – 64 bits de texte en clair
- Algorithme :
  - $(LL0, RR0) = CC$, divisant CC dans deux parties de 32 bits
  - $LL1 = LL0 \mathrm{XOR} P18$
  - $RR2 = RR0 \mathrm{XOR} F(LL1) \mathrm{XOR} P17$
  - $LL3 = LL1 \mathrm{XOR} F(RR2) \mathrm{XOR} P16$
  - $RR4 = RR2 \mathrm{XOR} F(LL3) \mathrm{XOR} P15$
  - $LL5 = LL3 \mathrm{XOR} F(RR4) \mathrm{XOR} P14$
  - $RR6 = RR4 \mathrm{XOR} F(LL5) \mathrm{XOR} P13$
  - $LL7 = LL5 \mathrm{XOR} F(RR6) \mathrm{XOR} P12$
  - $RR8 = RR6 \mathrm{XOR} F(LL7) \mathrm{XOR} P11$
  - $LL9 = LL7 \mathrm{XOR} F(RR8) \mathrm{XOR} P10$
  - $RR10 = RR8 \mathrm{XOR} F(LL9) \mathrm{XOR} P9$
  - $LL11 = LL9 \mathrm{XOR} F(RR10) \mathrm{XOR} P8$
  - $RR12 = RR10 \mathrm{XOR} F(LL11) \mathrm{XOR} P7$
  - $LL13 = LL11 \mathrm{XOR} F(RR12) \mathrm{XOR} P6$
  - $RR14 = RR12 \mathrm{XOR} F(LL13) \mathrm{XOR} P5$
  - $LL15 = LL13 \mathrm{XOR} F(RR14) \mathrm{XOR} P4$
  - $RR16 = RR14 \mathrm{XOR} F(LL15) \mathrm{XOR} P3$
  - $LL17 = LL15 \mathrm{XOR} F(RR16) \mathrm{XOR} P2$
  - $RR18 = R16 \mathrm{XOR} P1$

Algorithme de décryptage

- Entrées :
  - CC – 64 bits of cipher text
  - $P1$, $P2$, ..., $P18$: 18 sous-clés
  - $F()$ – Round function
- Sortie : TT – 64 bits de texte en clair
- Algorithme :
  - $(LL0, RR0) = CC$, divisant CC dans deux parties de 32 bits
  - $LL1 = LL0 \mathrm{XOR} P18$
  - $RR2 = RR0 \mathrm{XOR} F(LL1) \mathrm{XOR} P17$
  - $LL3 = LL1 \mathrm{XOR} F(RR2) \mathrm{XOR} P16$
  - $RR4 = RR2 \mathrm{XOR} F(LL3) \mathrm{XOR} P15$
  - $LL5 = LL3 \mathrm{XOR} F(RR4) \mathrm{XOR} P14$
  - $RR6 = RR4 \mathrm{XOR} F(LL5) \mathrm{XOR} P13$
  - $LL7 = LL5 \mathrm{XOR} F(RR6) \mathrm{XOR} P12$
  - $RR8 = RR6 \mathrm{XOR} F(LL7) \mathrm{XOR} P11$
  - $LL9 = LL7 \mathrm{XOR} F(RR8) \mathrm{XOR} P10$
  - $RR10 = RR8 \mathrm{XOR} F(LL9) \mathrm{XOR} P9$
  - $LL11 = LL9 \mathrm{XOR} F(RR10) \mathrm{XOR} P8$
  - $RR12 = RR10 \mathrm{XOR} F(LL11) \mathrm{XOR} P7$
  - $LL13 = LL11 \mathrm{XOR} F(RR12) \mathrm{XOR} P6$
  - $RR14 = RR12 \mathrm{XOR} F(LL13) \mathrm{XOR} P5$
  - $LL15 = LL13 \mathrm{XOR} F(RR14) \mathrm{XOR} P4$
  - $RR16 = RR14 \mathrm{XOR} F(LL15) \mathrm{XOR} P3$
  - $LL17 = LL15 \mathrm{XOR} F(RR16) \mathrm{XOR} P2$
  - $RR18 = R16 \mathrm{XOR} P1$

Algorithme de décryptage

- Entrées :
  - CC – 64 bits of cipher text
  - $P1$, $P2$, ..., $P18$: 18 sous-clés
  - $F()$ – Round function
- Sortie : TT – 64 bits de texte en clair
- Algorithme :
  - $(LL0, RR0) = CC$, divisant CC dans deux parties de 32 bits
  - $LL1 = LL0 \mathrm{XOR} P18$
  - $RR2 = RR0 \mathrm{XOR} F(LL1) \mathrm{XOR} P17$
  - $LL3 = LL1 \mathrm{XOR} F(RR2) \mathrm{XOR} P16$
  - $RR4 = RR2 \mathrm{XOR} F(LL3) \mathrm{XOR} P15$
  - $LL5 = LL3 \mathrm{XOR} F(RR4) \mathrm{XOR} P14$
  - $RR6 = RR4 \mathrm{XOR} F(LL5) \mathrm{XOR} P13$
  - $LL7 = LL5 \mathrm{XOR} F(RR6) \mathrm{XOR} P12$
  - $RR8 = RR6 \mathrm{XOR} F(LL7) \mathrm{XOR} P11$
  - $LL9 = LL7 \mathrm{XOR} F(RR8) \mathrm{XOR} P10$
  - $RR10 = RR8 \mathrm{XOR} F(LL9) \mathrm{XOR} P9$
  - $LL11 = LL9 \mathrm{XOR} F(RR10) \mathrm{XOR} P8$
  - $RR12 = RR10 \mathrm{XOR} F(LL11) \mathrm{XOR} P7$
  - $LL13 = LL11 \mathrm{XOR} F(RR12) \mathrm{XOR} P6$
  - $RR14 = RR12 \mathrm{XOR} F(LL13) \mathrm{XOR} P5$
  - $LL15 = LL13 \mathrm{XOR} F(RR14) \mathrm{XOR} P4$
  - $RR16 = RR14 \mathrm{XOR} F(LL15) \mathrm{XOR} P3$
  - $LL17 = LL15 \mathrm{XOR} F(RR16) \mathrm{XOR} P2$
  - $RR18 = R16 \mathrm{XOR} P1$

Algorithme de décryptage

- Entrées :
  - CC – 64 bits of cipher text
  - $P1$, $P2$, ..., $P18$: 18 sous-clés
  - $F()$ – Round function
- Sortie : TT – 64 bits de texte en clair
- Algorithme :
  - $(LL0, RR0) = CC$, divisant CC dans deux parties de 32 bits
  - $LL1 = LL0 \mathrm{XOR} P18$
  - $RR2 = RR0 \mathrm{XOR} F(LL1) \mathrm{XOR} P17$
  - $LL3 = LL1 \mathrm{XOR} F(RR2) \mathrm{XOR} P16$
  - $RR4 = RR2 \mathrm{XOR} F(LL3) \mathrm{XOR} P15$
  - $LL5 = LL3 \mathrm{XOR} F(RR4) \mathrm{XOR} P14$
  - $RR6 = RR4 \mathrm{XOR} F(LL5) \mathrm{XOR} P13$
  - $LL7 = LL5 \mathrm{XOR} F(RR6) \mathrm{XOR} P12$
  - $RR8 = RR6 \mathrm{XOR} F(LL7) \mathrm{XOR} P11$
  - $LL9 = LL7 \mathrm{XOR} F(RR8) \mathrm{XOR} P10$
  - $RR10 = RR8 \mathrm{XOR} F(LL9) \mathrm{XOR} P9$
  - $LL11 = LL9 \mathrm{XOR} F(RR10) \mathrm{XOR} P8$
  - $RR12 = RR10 \mathrm{XOR} F(LL11) \mathrm{XOR} P7$
  - $LL13 = LL11 \mathrm{XOR} F(RR12) \mathrm{XOR} P6$
  - $RR14 = RR12 \mathrm{XOR} F(LL13) \mathrm{XOR} P5$
  - $LL15 = LL13 \mathrm{XOR} F(RR14) \mathrm{XOR} P4$
  - $RR16 = RR14 \mathrm{XOR} F(LL15) \mathrm{XOR} P3$
  - $LL17 = LL15 \mathrm{XOR} F(RR16) \mathrm{XOR} P2$
  - $RR18 = R16 \mathrm{XOR} P1$

Algorithme de décryptage

- Entrées :
  - CC – 64 bits of cipher text
  - $P1$, $P2$, ..., $P18$: 18 sous-clés
  - $F()$ – Round function
- Sortie : TT – 64 bits de texte en clair
- Algorithme :
  - $(LL0, RR0) = CC$, divisant CC dans deux parties de 32 bits
  - $LL1 = LL0 \mathrm{XOR} P18$
  - $RR2 = RR0 \mathrm{XOR} F(LL1) \mathrm{XOR} P17$
  - $LL3 = LL1 \mathrm{XOR} F(RR2) \mathrm{XOR} P16$
  - $RR4 = RR2 \mathrm{XOR} F(LL3) \mathrm{XOR} P15$
  - $LL5 = LL3 \mathrm{XOR} F(RR4) \mathrm{XOR} P14$
  - $RR6 = RR4 \mathrm{XOR} F(LL5) \mathrm{XOR} P13$
  - $LL7 = LL5 \mathrm{XOR} F(RR6) \mathrm{XOR} P12$
  - $RR8 = RR6 \mathrm{XOR} F(LL7) \mathrm{XOR} P11$
  - $LL9 = LL7 \mathrm{XOR} F(RR8) \mathrm{XOR} P10$
  - $RR10 = RR8 \mathrm{XOR} F(LL9) \mathrm{XOR} P9$
  - $LL11 = LL9 \mathrm{XOR} F(RR10) \mathrm{XOR} P8$
  - $RR12 = RR10 \mathrm{XOR} F(LL11) \mathrm{XOR} P7$
  - $LL13 = LL11 \mathrm{XOR} F(RR12) \mathrm{XOR} P6$
  - $RR14 = RR12 \mathrm{XOR} F(LL13) \mathrm{XOR} P5$
  - $LL15 = LL13 \mathrm{XOR} F(RR14) \mathrm{XOR} P4$
  - $RR16 = RR14 \mathrm{XOR} F(LL15) \mathrm{XOR} P3$
  - $LL17 = LL15 \mathrm{XOR} F(RR16) \mathrm{XOR} P2$
  - $RR18 = R16 \mathrm{XOR} P1$

Algorithme de décryptage

- Entrées :
  - CC – 64 bits of cipher text
  - $P1$, $P2$, ..., $P18$: 18 sous-clés
  - $F()$ – Round function
- Sortie : TT – 64 bits de texte en clair
- Algorithme :
  - $(LL0, RR0) = CC$, divisant CC dans deux parties de 32 bits
  - $LL1 = LL0 \mathrm{XOR} P18$
  - $RR2 = RR0 \mathrm{XOR} F(LL1) \mathrm{XOR} P17$
  - $LL3 = LL1 \mathrm{XOR} F(RR2) \mathrm{XOR} P16$
  - $RR4 = RR2 \mathrm{XOR} F(LL3) \mathrm{XOR} P15$
  - $LL5 = LL3 \mathrm{XOR} F(RR4) \mathrm{XOR} P14$
  - $RR6 = RR4 \mathrm{XOR} F(LL5) \mathrm{XOR} P13$
  - $LL7 = LL5 \mathrm{XOR} F(RR6) \mathrm{XOR} P12$
  - $RR8 = RR6 \mathrm{XOR} F(LL7) \mathrm{XOR} P11$
  - $LL9 = LL7 \mathrm{XOR} F(RR8) \mathrm{XOR} P10$
  - $RR10 = RR8 \mathrm{XOR} F(LL9) \mathrm{XOR} P9$
  - $LL11 = LL9 \mathrm{XOR} F(RR10) \mathrm{XOR} P8$
  - $RR12 = RR10 \mathrm{XOR} F(LL11) \mathrm{XOR} P7$
  - $LL13 = LL11 \mathrm{XOR} F(RR12) \mathrm{XOR} P6$
  - $RR14 = RR12 \mathrm{XOR} F(LL13) \mathrm{XOR} P5$
  - $LL15 = LL13 \mathrm{XOR} F(RR14) \mathrm{XOR} P4$
  - $RR16 = RR14 \mathrm{XOR} F(LL15) \mathrm{XOR} P3$
  - $LL17 = LL15 \mathrm{XOR} F(RR16) \mathrm{XOR} P2$
  - $RR18 = R16 \mathrm{XOR} P1$

Algorithme de décryptage

- Entrées :
  - CC – 64 bits of cipher text
  - $P1$, $P2$, ..., $P18$: 18 sous-clés
  - $F()$ – Round function
- Sortie : TT – 64 bits de texte en clair
- Algorithme :
  - $(LL0, RR0) = CC$, divisant CC dans deux parties de 32 bits
  - $LL1 = LL0 \mathrm{XOR} P18$
  - $RR2 = RR0 \mathrm{XOR} F(LL1) \mathrm{XOR} P17$
  - $LL3 = LL1 \mathrm{XOR} F(RR2) \mathrm{XOR} P16$
  - $RR4 = RR2 \mathrm{XOR} F(LL3) \mathrm{XOR} P15$
  - $LL5 = LL3 \mathrm{XOR} F(RR4) \mathrm{XOR} P14$
  - $RR6 = RR4 \mathrm{XOR} F(LL5) \mathrm{XOR} P13$
  - $LL7 = LL5 \mathrm{XOR} F(RR6) \mathrm{XOR} P12$
  - $RR8 = RR6 \mathrm{XOR} F(LL7) \mathrm{XOR} P11$
  - $LL9 = LL7 \mathrm{XOR} F(RR8) \mathrm{XOR} P10$
  - $RR10 = RR8 \mathrm{XOR} F(LL9) \mathrm{XOR} P9$
  - $LL11 = LL9 \mathrm{XOR} F(RR10) \mathrm{XOR} P8$
  - $RR12 = RR10 \mathrm{XOR} F(LL11) \mathrm{XOR} P7$
  - $LL13 = LL11 \mathrm{XOR} F(RR12) \mathrm{XOR} P6$
  - $RR14 = RR12 \mathrm{XOR} F(LL13) \mathrm{XOR} P5$
  - $LL15 = LL13 \mathrm{XOR} F(RR14) \mathrm{XOR} P4$
  - $RR16 = RR14 \mathrm{XOR} F(LL15) \mathrm{XOR} P3$
  - $LL17 = LL15 \mathrm{XOR} F(RR16) \mathrm{XOR} P2$
  - $RR18 = R16 \mathrm{XOR} P1$

Algorithme de décryptage

- Entrées :
  - CC – 64 bits of cipher text
  - $P1$, $P2$, ..., $P18$: 18 sous-clés
  - $F()$ – Round function
- Sortie : TT – 64 bits de texte en clair
- Algorithme :
  - $(LL0, RR0) = CC$, divisant CC dans deux parties de 32 bits
  - $LL1 = LL0 \mathrm{XOR} P18$
  - $RR2 = RR0 \mathrm{XOR} F(LL1) \mathrm{XOR} P17$
  - $LL3 = LL1 \mathrm{XOR} F(RR2) \mathrm{XOR} P16$
  - $RR4 = RR2 \mathrm{XOR} F(LL3) \mathrm{XOR} P15$
  - $LL5 = LL3 \mathrm{XOR} F(RR4) \mathrm{XOR} P14$
  - $RR6 = RR4 \mathrm{XOR} F(LL5) \mathrm{XOR} P13$
  - $LL7 = LL5 \mathrm{XOR} F(RR6) \mathrm{XOR} P12$
  - $RR8 = RR6 \mathrm{XOR} F(LL7) \mathrm{XOR} P11$
  - $LL9 = LL7 \mathrm{XOR} F(RR8) \mathrm{XOR} P10$
  - $RR10 = RR8 \mathrm{XOR} F(LL9) \mathrm{XOR} P9$
  - $LL11 = LL9 \mathrm{XOR} F(RR10) \mathrm{XOR} P8$
  - $RR12 = RR10 \mathrm{XOR} F(LL11) \mathrm{XOR} P7$
  - $LL13 = LL11 \mathrm{XOR} F(RR12) \mathrm{XOR} P6$
  - $RR14 = RR12 \mathrm{XOR} F(LL13) \mathrm{XOR} P5$
  - $LL15 = LL13 \mathrm{XOR} F(RR14) \mathrm{XOR} P4$
  - $RR16 = RR14 \mathrm{XOR} F(LL15) \mathrm{XOR} P3$
  - $LL17 = LL15 \mathrm{XOR} F(RR16) \mathrm{XOR} P2$
  - $RR18 = RR16 \mathrm{XOR} P1$

Algorithme de décryptage

- Entrées :
  - CC – 64 bits of cipher text
  - $P1$, $P2$, ..., $P18$: 18 sous-clés
  - $F()$ – Round function
- Sortie : TT – 64 bits de texte en clair
- Algorithme :
  - $(LL0, RR0) = CC$, divisant CC dans deux parties de 32 bits
  - $LL1 = LL0 \mathrm{XOR} P18$
  - $RR2 = RR0 \mathrm{XOR} F(LL1) \mathrm{XOR} P17$
  - $LL3 = LL1 \mathrm{XOR} F(RR2) \mathrm{XOR} P16$
  - $RR4 = RR2 \mathrm{XOR} F(LL3) \mathrm{XOR} P15$
  - $LL5 = LL3 \mathrm{XOR} F(RR4) \mathrm{XOR} P14$
  - $RR6 = RR4 \mathrm{XOR} F(LL5) \mathrm{XOR} P13$
  - $LL7 = LL5 \mathrm{XOR} F(RR6) \mathrm{XOR} P12$
  - $RR8 = RR6 \mathrm{XOR} F(LL7) \mathrm{XOR} P11$
  - $LL9 = LL7 \mathrm{XOR} F(RR8) \mathrm{XOR} P10$
  - $RR10 = RR8 \mathrm{XOR} F(LL9) \mathrm{XOR} P9$
  - $LL11 = LL9 \mathrm{XOR} F(RR10) \mathrm{XOR} P8$
  - $RR12 = RR10 \mathrm{XOR} F(LL11) \mathrm{XOR} P7$
  - $LL13 = LL11 \mathrm{XOR} F(RR12) \mathrm{XOR} P6$
  - $RR14 = RR12 \mathrm{XOR} F(LL13) \mathrm{XOR} P5$
  - $LL15 = LL13 \mathrm{XOR} F(RR14) \mathrm{XOR} P4$
  - $RR16 = RR14 \mathrm{XOR} F(LL15) \mathrm{XOR} P3$
  - $LL17 = LL15 \mathrm{XOR} F(RR16) \mathrm{XOR} P2$
  - $RR18 = R16 \mathrm{XOR} P1$

Algorithme de décryptage

- Entrées :
  - CC – 64 bits of cipher text
  - $P1$, $P2$, ..., $P18$: 18 sous-clés
  - $F()$ – Round function
- Sortie : TT – 64 bits de texte en clair
- Algorithme :
  - $(LL0, RR0) = CC$, divisant CC dans deux parties de 32 bits
  - $LL1 = LL0 \mathrm{XOR} P18$
  - $RR2 = RR0 \mathrm{XOR} F(LL1) \mathrm{XOR} P17$
  - $LL3 = LL1 \mathrm{XOR} F(RR2) \mathrm{XOR} P16$
  - $RR4 = RR2 \mathrm{XOR} F(LL3) \mathrm{XOR} P15$
  - $LL5 = LL3 \mathrm{XOR} F(RR4) \mathrm{XOR} P14$
  - $RR6 = RR4 \mathrm{XOR} F(LL5) \mathrm{XOR} P13$
  - $LL7 = LL5 \mathrm{XOR} F(RR6) \mathrm{XOR} P12$
  - $RR8 = RR6 \mathrm{XOR} F(LL7) \mathrm{XOR} P11$
  - $LL9 = LL7 \mathrm{XOR} F(RR8) \mathrm{XOR} P10$
  - $RR10 = RR8 \mathrm{XOR} F(LL9) \mathrm{XOR} P9$
  - $LL11 = LL9 \mathrm{XOR} F(RR10) \mathrm{XOR} P8$
  - $RR12 = RR10 \mathrm{XOR} F(LL11) \mathrm{XOR} P7$
  - $LL13 = LL11 \mathrm{XOR} F(RR12) \mathrm{XOR} P6$
  - $RR14 = RR12 \mathrm{XOR} F(LL13) \mathrm{XOR} P5$
  - $LL15 = LL13 \mathrm{XOR} F(RR14) \mathrm{XOR} P4$
  - $RR16 = RR14 \mathrm{XOR} F(LL15) \mathrm{XOR} P3$
  - $LL17 = LL15 \mathrm{XOR} F(RR16) \mathrm{XOR} P2$
  - $RR18 = R16 \mathrm{XOR} P1$

Algorithme de décryptage

- Entrées :
  - CC – 64 bits of cipher text
  - $P1$, $P2$, ..., $P18$: 18 sous-clés
  - $F()$ – Round function
- Sortie : TT – 64 bits de texte en clair
- Algorithme :
  - $(LL0, RR0) = CC$, divisant CC dans deux parties de 32 bits
  - $LL1 = LL0 \mathrm{XOR} P18$
  - $RR2 = RR0 \mathrm{XOR} F(LL1) \mathrm{XOR} P17$
  - $LL3 = LL1 \mathrm{XOR} F(RR2) \mathrm{XOR} P16$
  - $RR4 = RR2 \mathrm{XOR} F(LL3) \mathrm{XOR} P15$
  - $LL5 = LL3 \mathrm{XOR} F(RR4) \mathrm{XOR} P14$
  - $RR6 = RR4 \mathrm{XOR} F(LL5) \mathrm{XOR} P13$
  - $LL7 = LL5 \mathrm{XOR} F(RR6) \mathrm{XOR} P12$
  - $RR8 = RR6 \mathrm{XOR} F(LL7) \mathrm{XOR} P11$
  - $LL9 = LL7 \mathrm{XOR} F(RR8) \mathrm{XOR} P10$
  - $RR10 = RR8 \mathrm{XOR} F(LL9) \mathrm{XOR} P9$
  - $LL11 = LL9 \mathrm{XOR} F(RR10) \mathrm{XOR} P8$
  - $RR12 = RR10 \mathrm{XOR} F(LL11) \mathrm{XOR} P7$
  - $LL13 = LL11 \mathrm{XOR} F(RR12) \mathrm{XOR} P6$
  - $RR14 = RR12 \mathrm{XOR} F(LL13) \mathrm{XOR} P5$
  - $LL15 = LL13 \mathrm{XOR} F(RR14) \mathrm{XOR} P4$
  - $RR16 = RR14 \mathrm{XOR} F(LL15) \mathrm{XOR} P3$
  - $LL17 = LL15 \mathrm{XOR} F(RR16) \mathrm{XOR} P2$
  - $RR18 = R16 \mathrm{XOR} P1$

Algorithme de décryptage

- Entrées :
  - CC – 64 bits of cipher text
  - $P1$, $P2$, ..., $P18$: 18 sous-clés
  - $F()$ – Round function
- Sortie : TT – 64 bits de texte en clair
- Algorithme :
  - $(LL0, RR0) = CC$, divisant CC dans deux parties de 32 bits
  - $LL1 = LL0 \mathrm{XOR} P18$
  - $RR2 = RR0 \mathrm{XOR} F(LL1) \mathrm{XOR} P17$
  - $LL3 = LL1 \mathrm{XOR} F(RR2) \mathrm{XOR} P16$
  - $RR4 = RR2 \mathrm{XOR} F(LL3) \mathrm{XOR} P15$
  - $LL5 = LL3 \mathrm{XOR} F(RR4) \mathrm{XOR} P14$
  - $RR6 = RR4 \mathrm{XOR} F(LL5) \mathrm{XOR} P13$
  - $LL7 = LL5 \mathrm{XOR} F(RR6) \mathrm{XOR} P12$
  - $RR8 = RR6 \mathrm{XOR} F(LL7) \mathrm{XOR} P11$
  - $LL9 = LL7 \mathrm{XOR} F(RR8) \mathrm{XOR} P10$
  - $RR10 = RR8 \mathrm{XOR} F(LL9) \mathrm{XOR} P9$
  - $LL11 = LL9 \mathrm{XOR} F(RR10) \mathrm{XOR} P8$
  - $RR12 = RR10 \mathrm{XOR} F(LL11) \mathrm{XOR} P7$
  - $LL13 = LL11 \mathrm{XOR} F(RR12) \mathrm{XOR} P6$
  - $RR14 = RR12 \mathrm{XOR} F(LL13) \mathrm{XOR} P5$
  - $LL15 = LL13 \mathrm{XOR} F(RR14) \mathrm{XOR} P4$
  - $RR16 = RR14 \mathrm{XOR} F(LL15) \mathrm{XOR} P3$
  - $LL17 = LL15 \mathrm{XOR} F(RR16) \mathrm{XOR} P2$
  - $RR18 = R16 \mathrm{XOR} P1$

Algorithme de décryptage

- Entrées :
  - CC – 64 bits of cipher text
  - $P1$, $P2$, ..., $P18$: 18 sous-clés
  - $F()$ – Round function
- Sortie : TT – 64 bits de texte en clair
- Algorithme :
  - $(LL0, RR0) = CC$, divisant CC dans deux parties de 32 bits
  - $LL1 = LL0 \mathrm{XOR} P18$
  - $RR2 = RR0 \mathrm{XOR} F(LL1) \mathrm{XOR} P17$
  - $LL3 = LL1 \mathrm{XOR} F(RR2) \mathrm{XOR} P16$
  - $RR4 = RR2 \mathrm{XOR} F(LL3) \mathrm{XOR} P15$
  - $LL5 = LL3 \mathrm{XOR} F(RR4) \mathrm{XOR} P14$
  - $RR6 = RR4 \mathrm{XOR} F(LL5) \mathrm{XOR} P13$
  - $LL7 = LL5 \mathrm{XOR} F(RR6) \mathrm{XOR} P12$
  - $RR8 = RR6 \mathrm{XOR} F(LL7) \mathrm{XOR} P11$
  - $LL9 = LL7 \mathrm{XOR} F(RR8) \mathrm{XOR} P10$
  - $RR10 = RR8 \mathrm{XOR} F(LL9) \mathrm{XOR} P9$
  - $LL11 = LL9 \mathrm{XOR} F(RR10) \mathrm{XOR} P8$
  - $RR12 = RR10 \mathrm{XOR} F(LL11) \mathrm{XOR} P7$
  - $LL13 = LL11 \mathrm{XOR} F(RR12) \mathrm{XOR} P6$
  - $RR14 = RR12 \mathrm{XOR} F(LL13) \mathrm{XOR} P5$
  - $LL15 = LL13 \mathrm{XOR} F(RR14) \mathrm{XOR} P4$
  - $RR16 = RR14 \mathrm{XOR} F(LL15) \mathrm{XOR} P3$
  - $LL17 = LL15 \mathrm{XOR} F(RR16) \mathrm{XOR} P2$
  - $RR18 = R16 \mathrm{XOR} P1$

Bibliographie

- Practical Cryptology and Web Security (2005)- P.K. Yuen
- Applied Cryptography (1996) - Bruce Schneier
- Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish) (1993)- Bruce Schneier
- http ://www.schneier.com/paper-blowfish-fse.html, http ://www.schneier.com/blowfish-bug.txt
- On the Weak Keys of Blowfish. (1996)- Serge Vaudenay
- Procs of the 3rd Intl Wkshp on Fast Soft Encrpt 1996 ISBN 3-540-60865-6
- http ://citeseerx.ist.psu.edu/viewdoc/summary ?doi=10.1.1.31.7942
- Systematic Generation of Cryptographically Robust S-Boxes, Procs 1st ACM Conf on Comp and Comm Security, 1993, Fairfax, VA, USA. ACM, pp. 171-182 - Jennifer Seberry, Xian-Mo Zhang, Yuliang Zheng
- http ://citeseerx.ist.psu.edu/viewdoc/summary ?doi=10.1.1.158.3013
- Cryptography Tutorials - Herong's Tutorial Examples http ://www.herongyang.com/Cryptography
- Blowfish Source Code : http ://www.schneier.com/blowfish-download.html
- Fcts Blowfish (CryptoSys API) : http ://www.cryptosys.net/blowfish.html
- Les chiffres hexa de π, en 4 s_boxes & 1 p_array : http ://www.schneier.com/code/constants.txt
- Test Vectors : http ://www.schneier.com/code/vectors.txt

Bibliographie

- Practical Cryptology and Web Security (2005)- P.K. Yuen
- Applied Cryptography (1996) - Bruce Schneier
- Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish) (1993)- Bruce Schneier
- http ://www.schneier.com/paper-blowfish-fse.html, http ://www.schneier.com/blowfish-bug.txt
- On the Weak Keys of Blowfish. (1996)- Serge Vaudenay
- Procs of the 3rd Intl Wkshp on Fast Soft Encrpt 1996 ISBN 3-540-60865-6
- http ://citeseerx.ist.psu.edu/viewdoc/summary ?doi=10.1.1.31.7942
- Systematic Generation of Cryptographically Robust S-Boxes, Procs 1st ACM Conf on Comp and Comm Security, 1993, Fairfax, VA, USA. ACM, pp. 171-182 - Jennifer Seberry, Xian-Mo Zhang, Yuliang Zheng
- http ://citeseerx.ist.psu.edu/viewdoc/summary ?doi=10.1.1.158.3013
- Cryptography Tutorials - Herong's Tutorial Examples http ://www.herongyang.com/Cryptography
- Blowfish Source Code : http ://www.schneier.com/blowfish-download.html
- Fcts Blowfish (CryptoSys API) : http ://www.cryptosys.net/blowfish.html
- Les chiffres hexa de π, en 4 s_boxes & 1 p_array : http ://www.schneier.com/code/constants.txt
- Test Vectors : http ://www.schneier.com/code/vectors.txt

Bibliographie

- Practical Cryptology and Web Security (2005)- P.K. Yuen
- Applied Cryptography (1996) - Bruce Schneier
- Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish) (1993)- Bruce Schneier
- http ://www.schneier.com/paper-blowfish-fse.html, http ://www.schneier.com/blowfish-bug.txt
- On the Weak Keys of Blowfish. (1996)- Serge Vaudenay
- Procs of the 3rd Intl Wkshp on Fast Soft Encrpt 1996 ISBN 3-540-60865-6
- http ://citeseerx.ist.psu.edu/viewdoc/summary ?doi=10.1.1.31.7942
- Systematic Generation of Cryptographically Robust S-Boxes, Procs 1st ACM Conf on Comp and Comm Security, 1993, Fairfax, VA, USA. ACM, pp. 171-182 - Jennifer Seberry, Xian-Mo Zhang, Yuliang Zheng
- http ://citeseerx.ist.psu.edu/viewdoc/summary ?doi=10.1.1.158.3013
- Cryptography Tutorials - Herong's Tutorial Examples http ://www.herongyang.com/Cryptography
- Blowfish Source Code : http ://www.schneier.com/blowfish-download.html
- Fcts Blowfish (CryptoSys API) : http ://www.cryptosys.net/blowfish.html
- Les chiffres hexa de π, en 4 s_boxes & 1 p_array : http ://www.schneier.com/code/constants.txt
- Test Vectors : http ://www.schneier.com/code/vectors.txt

Bibliographie

- Practical Cryptology and Web Security (2005)- P.K. Yuen
- Applied Cryptography (1996) - Bruce Schneier
- Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish) (1993)- Bruce Schneier
- http ://www.schneier.com/paper-blowfish-fse.html, http ://www.schneier.com/blowfish-bug.txt
- On the Weak Keys of Blowfish. (1996)- Serge Vaudenay
- Procs of the 3rd Intl Wkshp on Fast Soft Encrpt 1996 ISBN 3-540-60865-6
- http ://citeseerx.ist.psu.edu/viewdoc/summary ?doi=10.1.1.31.7942
- Systematic Generation of Cryptographically Robust S-Boxes, Procs 1st ACM Conf on Comp and Comm Security, 1993, Fairfax, VA, USA. ACM, pp. 171-182 - Jennifer Seberry, Xian-Mo Zhang, Yuliang Zheng
- http ://citeseerx.ist.psu.edu/viewdoc/summary ?doi=10.1.1.158.3013
- Cryptography Tutorials - Herong's Tutorial Examples http ://www.herongyang.com/Cryptography
- Blowfish Source Code : http ://www.schneier.com/blowfish-download.html
- Fcts Blowfish (CryptoSys API) : http ://www.cryptosys.net/blowfish.html
- Les chiffres hexa de π, en 4 s_boxes & 1 p_array : http ://www.schneier.com/code/constants.txt
- Test Vectors : http ://www.schneier.com/code/vectors.txt

Bibliographie

- Practical Cryptology and Web Security (2005)- P.K. Yuen
- Applied Cryptography (1996) - Bruce Schneier
- Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish) (1993)- Bruce Schneier
- http ://www.schneier.com/paper-blowfish-fse.html, http ://www.schneier.com/blowfish-bug.txt
- On the Weak Keys of Blowfish. (1996)- Serge Vaudenay
- Procs of the 3rd Intl Wkshp on Fast Soft Encrpt 1996 ISBN 3-540-60865-6
- http ://citeseerx.ist.psu.edu/viewdoc/summary ?doi=10.1.1.31.7942
- Systematic Generation of Cryptographically Robust S-Boxes, Procs 1st ACM Conf on Comp and Comm Security, 1993, Fairfax, VA, USA. ACM, pp. 171-182 - Jennifer Seberry, Xian-Mo Zhang, Yuliang Zheng
- http ://citeseerx.ist.psu.edu/viewdoc/summary ?doi=10.1.1.158.3013
- Cryptography Tutorials - Herong's Tutorial Examples http ://www.herongyang.com/Cryptography
- Blowfish Source Code : http ://www.schneier.com/blowfish-download.html
- Fcts Blowfish (CryptoSys API) : http ://www.cryptosys.net/blowfish.html
- Les chiffres hexa de π, en 4 s_boxes & 1 p_array : http ://www.schneier.com/code/constants.txt
- Test Vectors : http ://www.schneier.com/code/vectors.txt

Bibliographie

- Practical Cryptology and Web Security (2005)- P.K. Yuen
- Applied Cryptography (1996) - Bruce Schneier
- Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish) (1993)- Bruce Schneier
- http ://www.schneier.com/paper-blowfish-fse.html, http ://www.schneier.com/blowfish-bug.txt
- On the Weak Keys of Blowfish. (1996)- Serge Vaudenay
- Procs of the 3rd Intl Wkshp on Fast Soft Encrpt 1996 ISBN 3-540-60865-6
- http ://citeseerx.ist.psu.edu/viewdoc/summary ?doi=10.1.1.31.7942
- Systematic Generation of Cryptographically Robust S-Boxes, Procs 1st ACM Conf on Comp and Comm Security, 1993, Fairfax, VA, USA. ACM, pp. 171-182 - Jennifer Seberry, Xian-Mo Zhang, Yuliang Zheng
- http ://citeseerx.ist.psu.edu/viewdoc/summary ?doi=10.1.1.158.3013
- Cryptography Tutorials - Herong's Tutorial Examples http ://www.herongyang.com/Cryptography
- Blowfish Source Code : http ://www.schneier.com/blowfish-download.html
- Fcts Blowfish (CryptoSys API) : http ://www.cryptosys.net/blowfish.html
- Les chiffres hexa de π, en 4 s_boxes & 1 p_array : http ://www.schneier.com/code/constants.txt
- Test Vectors : http ://www.schneier.com/code/vectors.txt

Bibliographie

- Practical Cryptology and Web Security (2005)- P.K. Yuen
- Applied Cryptography (1996) - Bruce Schneier
- Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish) (1993)- Bruce Schneier
- http ://www.schneier.com/paper-blowfish-fse.html, http ://www.schneier.com/blowfish-bug.txt
- On the Weak Keys of Blowfish. (1996)- Serge Vaudenay
- Procs of the 3rd Intl Wkshp on Fast Soft Encrpt 1996 ISBN 3-540-60865-6
- http ://citeseerx.ist.psu.edu/viewdoc/summary ?doi=10.1.1.31.7942
- Systematic Generation of Cryptographically Robust S-Boxes, Procs 1st ACM Conf on Comp and Comm Security, 1993, Fairfax, VA, USA. ACM, pp. 171-182 - Jennifer Seberry, Xian-Mo Zhang, Yuliang Zheng
- http ://citeseerx.ist.psu.edu/viewdoc/summary ?doi=10.1.1.158.3013
- Cryptography Tutorials - Herong's Tutorial Examples http ://www.herongyang.com/Cryptography
- Blowfish Source Code : http ://www.schneier.com/blowfish-download.html
- Fcts Blowfish (CryptoSys API) : http ://www.cryptosys.net/blowfish.html
- Les chiffres hexa de π, en 4 s_boxes & 1 p_array : http ://www.schneier.com/code/constants.txt
- Test Vectors : http ://www.schneier.com/code/vectors.txt

Bibliographie

- Practical Cryptology and Web Security (2005)- P.K. Yuen
- Applied Cryptography (1996) - Bruce Schneier
- Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish) (1993)- Bruce Schneier
- http ://www.schneier.com/paper-blowfish-fse.html, http ://www.schneier.com/blowfish-bug.txt
- On the Weak Keys of Blowfish. (1996)- Serge Vaudenay
- Procs of the 3rd Intl Wkshp on Fast Soft Encrpt 1996 ISBN 3-540-60865-6
- http ://citeseerx.ist.psu.edu/viewdoc/summary ?doi=10.1.1.31.7942
- Systematic Generation of Cryptographically Robust S-Boxes, Procs 1st ACM Conf on Comp and Comm Security, 1993, Fairfax, VA, USA. ACM, pp. 171-182 - Jennifer Seberry, Xian-Mo Zhang, Yuliang Zheng
- http ://citeseerx.ist.psu.edu/viewdoc/summary ?doi=10.1.1.158.3013
- Cryptography Tutorials - Herong's Tutorial Examples http ://www.herongyang.com/Cryptography
- Blowfish Source Code : http ://www.schneier.com/blowfish-download.html
- Fcts Blowfish (CryptoSys API) : http ://www.cryptosys.net/blowfish.html
- Les chiffres hexa de π, en 4 s_boxes & 1 p_array : http ://www.schneier.com/code/constants.txt
- Test Vectors : http ://www.schneier.com/code/vectors.txt

Bibliographie

- Practical Cryptology and Web Security (2005)- P.K. Yuen
- Applied Cryptography (1996) - Bruce Schneier
- Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish) (1993)- Bruce Schneier
- http ://www.schneier.com/paper-blowfish-fse.html, http ://www.schneier.com/blowfish-bug.txt
- On the Weak Keys of Blowfish. (1996)- Serge Vaudenay
- Procs of the 3rd Intl Wkshp on Fast Soft Encrpt 1996 ISBN 3-540-60865-6
- http ://citeseerx.ist.psu.edu/viewdoc/summary ?doi=10.1.1.31.7942
- Systematic Generation of Cryptographically Robust S-Boxes, Procs 1st ACM Conf on Comp and Comm Security, 1993, Fairfax, VA, USA. ACM, pp. 171-182 - Jennifer Seberry, Xian-Mo Zhang, Yuliang Zheng
- http ://citeseerx.ist.psu.edu/viewdoc/summary ?doi=10.1.1.158.3013
- Cryptography Tutorials - Herong's Tutorial Examples http ://www.herongyang.com/Cryptography
- Blowfish Source Code : http ://www.schneier.com/blowfish-download.html
- Fcts Blowfish (CryptoSys API) : http ://www.cryptosys.net/blowfish.html
- Les chiffres hexa de π, en 4 s_boxes & 1 p_array : http ://www.schneier.com/code/constants.txt
- Test Vectors : http ://www.schneier.com/code/vectors.txt

Bibliographie

- Practical Cryptology and Web Security (2005)- P.K. Yuen
- Applied Cryptography (1996) - Bruce Schneier
- Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish) (1993)- Bruce Schneier
- http ://www.schneier.com/paper-blowfish-fse.html, http ://www.schneier.com/blowfish-bug.txt
- On the Weak Keys of Blowfish. (1996)- Serge Vaudenay
- Procs of the 3rd Intl Wkshp on Fast Soft Encrpt 1996 ISBN 3-540-60865-6
- http ://citeseerx.ist.psu.edu/viewdoc/summary ?doi=10.1.1.31.7942
- Systematic Generation of Cryptographically Robust S-Boxes, Procs 1st ACM Conf on Comp and Comm Security, 1993, Fairfax, VA, USA. ACM, pp. 171-182 - Jennifer Seberry, Xian-Mo Zhang, Yuliang Zheng
- http ://citeseerx.ist.psu.edu/viewdoc/summary ?doi=10.1.1.158.3013
- Cryptography Tutorials - Herong's Tutorial Examples http ://www.herongyang.com/Cryptography
- Blowfish Source Code : http ://www.schneier.com/blowfish-download.html
- Fcts Blowfish (CryptoSys API) : http ://www.cryptosys.net/blowfish.html
- Les chiffres hexa de π, en 4 s_boxes & 1 p_array : http ://www.schneier.com/code/constants.txt
- Test Vectors : http ://www.schneier.com/code/vectors.txt

Bibliographie

- Practical Cryptology and Web Security (2005)- P.K. Yuen
- Applied Cryptography (1996) - Bruce Schneier
- Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish) (1993)- Bruce Schneier
- http ://www.schneier.com/paper-blowfish-fse.html, http ://www.schneier.com/blowfish-bug.txt
- On the Weak Keys of Blowfish. (1996)- Serge Vaudenay
- Procs of the 3rd Intl Wkshp on Fast Soft Encrpt 1996 ISBN 3-540-60865-6
- http ://citeseerx.ist.psu.edu/viewdoc/summary ?doi=10.1.1.31.7942
- Systematic Generation of Cryptographically Robust S-Boxes, Procs 1st ACM Conf on Comp and Comm Security, 1993, Fairfax, VA, USA. ACM, pp. 171-182 - Jennifer Seberry, Xian-Mo Zhang, Yuliang Zheng
- http ://citeseerx.ist.psu.edu/viewdoc/summary ?doi=10.1.1.158.3013
- Cryptography Tutorials - Herong's Tutorial Examples http ://www.herongyang.com/Cryptography
- Blowfish Source Code : http ://www.schneier.com/blowfish-download.html
- Fcts Blowfish (CryptoSys API) : http ://www.cryptosys.net/blowfish.html
- Les chiffres hexa de π, en 4 s_boxes & 1 p_array : http ://www.schneier.com/code/constants.txt
- Test Vectors : http ://www.schneier.com/code/vectors.txt

Bibliographie

- Practical Cryptology and Web Security (2005)- P.K. Yuen
- Applied Cryptography (1996) - Bruce Schneier
- Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish) (1993)- Bruce Schneier
- http ://www.schneier.com/paper-blowfish-fse.html, http ://www.schneier.com/blowfish-bug.txt
- On the Weak Keys of Blowfish. (1996)- Serge Vaudenay
- Procs of the 3rd Intl Wkshp on Fast Soft Encrpt 1996 ISBN 3-540-60865-6
- http ://citeseerx.ist.psu.edu/viewdoc/summary ?doi=10.1.1.31.7942
- Systematic Generation of Cryptographically Robust S-Boxes, Procs 1st ACM Conf on Comp and Comm Security, 1993, Fairfax, VA, USA. ACM, pp. 171-182 - Jennifer Seberry, Xian-Mo Zhang, Yuliang Zheng
- http ://citeseerx.ist.psu.edu/viewdoc/summary ?doi=10.1.1.158.3013
- Cryptography Tutorials - Herong's Tutorial Examples http ://www.herongyang.com/Cryptography
- Blowfish Source Code : http ://www.schneier.com/blowfish-download.html
- Fcts Blowfish (CryptoSys API) : http ://www.cryptosys.net/blowfish.html
- Les chiffres hexa de π, en 4 s_boxes & 1 p_array : http ://www.schneier.com/code/constants.txt
- Test Vectors : http ://www.schneier.com/code/vectors.txt

Bibliographie

- Practical Cryptology and Web Security (2005)- P.K. Yuen
- Applied Cryptography (1996) - Bruce Schneier
- Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish) (1993)- Bruce Schneier
- http ://www.schneier.com/paper-blowfish-fse.html, http ://www.schneier.com/blowfish-bug.txt
- On the Weak Keys of Blowfish. (1996)- Serge Vaudenay
- Procs of the 3rd Intl Wkshp on Fast Soft Encrpt 1996 ISBN 3-540-60865-6
- http ://citeseerx.ist.psu.edu/viewdoc/summary ?doi=10.1.1.31.7942
- Systematic Generation of Cryptographically Robust S-Boxes, Procs 1st ACM Conf on Comp and Comm Security, 1993, Fairfax, VA, USA. ACM, pp. 171-182 - Jennifer Seberry, Xian-Mo Zhang, Yuliang Zheng
- http ://citeseerx.ist.psu.edu/viewdoc/summary ?doi=10.1.1.158.3013
- Cryptography Tutorials - Herong's Tutorial Examples http ://www.herongyang.com/Cryptography
- Blowfish Source Code : http ://www.schneier.com/blowfish-download.html
- Fcts Blowfish (CryptoSys API) : http ://www.cryptosys.net/blowfish.html
- Les chiffres hexa de π, en 4 s_boxes & 1 p_array : http ://www.schneier.com/code/constants.txt
- Test Vectors : http ://www.schneier.com/code/vectors.txt

Bibliographie

- Practical Cryptology and Web Security (2005)- P.K. Yuen
- Applied Cryptography (1996) - Bruce Schneier
- Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish) (1993)- Bruce Schneier
- http ://www.schneier.com/paper-blowfish-fse.html, http ://www.schneier.com/blowfish-bug.txt
- On the Weak Keys of Blowfish. (1996)- Serge Vaudenay
- Procs of the 3rd Intl Wkshp on Fast Soft Encrpt 1996 ISBN 3-540-60865-6
- http ://citeseerx.ist.psu.edu/viewdoc/summary ?doi=10.1.1.31.7942
- Systematic Generation of Cryptographically Robust S-Boxes, Procs 1st ACM Conf on Comp and Comm Security, 1993, Fairfax, VA, USA. ACM, pp. 171-182 - Jennifer Seberry, Xian-Mo Zhang, Yuliang Zheng
- http ://citeseerx.ist.psu.edu/viewdoc/summary ?doi=10.1.1.158.3013
- Cryptography Tutorials - Herong's Tutorial Examples http ://www.herongyang.com/Cryptography
- Blowfish Source Code : http ://www.schneier.com/blowfish-download.html
- Fcts Blowfish (CryptoSys API) : http ://www.cryptosys.net/blowfish.html
- Les chiffres hexa de π, en 4 s_boxes & 1 p_array : http ://www.schneier.com/code/constants.txt
- Test Vectors : http ://www.schneier.com/code/vectors.txt

Bibliographie

- Practical Cryptology and Web Security (2005)- P.K. Yuen
- Applied Cryptography (1996) - Bruce Schneier
- Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish) (1993)- Bruce Schneier
- http ://www.schneier.com/paper-blowfish-fse.html, http ://www.schneier.com/blowfish-bug.txt
- On the Weak Keys of Blowfish. (1996)- Serge Vaudenay
- Procs of the 3rd Intl Wkshp on Fast Soft Encrpt 1996 ISBN 3-540-60865-6
- http ://citeseerx.ist.psu.edu/viewdoc/summary ?doi=10.1.1.31.7942
- Systematic Generation of Cryptographically Robust S-Boxes, Procs 1st ACM Conf on Comp and Comm Security, 1993, Fairfax, VA, USA. ACM, pp. 171-182 - Jennifer Seberry, Xian-Mo Zhang, Yuliang Zheng
- http ://citeseerx.ist.psu.edu/viewdoc/summary ?doi=10.1.1.158.3013
- Cryptography Tutorials - Herong's Tutorial Examples http ://www.herongyang.com/Cryptography
- Blowfish Source Code : http ://www.schneier.com/blowfish-download.html
- Fcts Blowfish (CryptoSys API) : http ://www.cryptosys.net/blowfish.html
- Les chiffres hexa de π, en 4 s_boxes & 1 p_array : http ://www.schneier.com/code/constants.txt
- Test Vectors : http ://www.schneier.com/code/vectors.txt

Bibliographie

- Practical Cryptology and Web Security (2005)- P.K. Yuen
- Applied Cryptography (1996) - Bruce Schneier
- Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish) (1993)- Bruce Schneier
- http ://www.schneier.com/paper-blowfish-fse.html, http ://www.schneier.com/blowfish-bug.txt
- On the Weak Keys of Blowfish. (1996)- Serge Vaudenay
- Procs of the 3rd Intl Wkshp on Fast Soft Encrpt 1996 ISBN 3-540-60865-6
- http ://citeseerx.ist.psu.edu/viewdoc/summary ?doi=10.1.1.31.7942
- Systematic Generation of Cryptographically Robust S-Boxes, Procs 1st ACM Conf on Comp and Comm Security, 1993, Fairfax, VA, USA. ACM, pp. 171-182 - Jennifer Seberry, Xian-Mo Zhang, Yuliang Zheng
- http ://citeseerx.ist.psu.edu/viewdoc/summary ?doi=10.1.1.158.3013
- Cryptography Tutorials - Herong's Tutorial Examples http ://www.herongyang.com/Cryptography
- Blowfish Source Code : http ://www.schneier.com/blowfish-download.html
- Fcts Blowfish (CryptoSys API) : http ://www.cryptosys.net/blowfish.html
- Les chiffres hexa de π, en 4 s_boxes & 1 p_array : http ://www.schneier.com/code/constants.txt
- Test Vectors : http ://www.schneier.com/code/vectors.txt

Bibliographie

- Practical Cryptology and Web Security (2005)- P.K. Yuen
- Applied Cryptography (1996) - Bruce Schneier
- Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish) (1993)- Bruce Schneier
- http ://www.schneier.com/paper-blowfish-fse.html, http ://www.schneier.com/blowfish-bug.txt
- On the Weak Keys of Blowfish. (1996)- Serge Vaudenay
- Procs of the 3rd Intl Wkshp on Fast Soft Encrpt 1996 ISBN 3-540-60865-6
- http ://citeseerx.ist.psu.edu/viewdoc/summary ?doi=10.1.1.31.7942
- Systematic Generation of Cryptographically Robust S-Boxes, Procs 1st ACM Conf on Comp and Comm Security, 1993, Fairfax, VA, USA. ACM, pp. 171-182 - Jennifer Seberry, Xian-Mo Zhang, Yuliang Zheng
- http ://citeseerx.ist.psu.edu/viewdoc/summary ?doi=10.1.1.158.3013
- Cryptography Tutorials - Herong's Tutorial Examples http ://www.herongyang.com/Cryptography
- Blowfish Source Code : http ://www.schneier.com/blowfish-download.html
- Fcts Blowfish (CryptoSys API) : http ://www.cryptosys.net/blowfish.html
- Les chiffres hexa de π, en 4 s_boxes & 1 p_array : http ://www.schneier.com/code/constants.txt
- Test Vectors : http ://www.schneier.com/code/vectors.txt

Bibliographie

- Practical Cryptology and Web Security (2005)- P.K. Yuen
- Applied Cryptography (1996) - Bruce Schneier
- Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish) (1993)- Bruce Schneier
- http ://www.schneier.com/paper-blowfish-fse.html, http ://www.schneier.com/blowfish-bug.txt
- On the Weak Keys of Blowfish. (1996)- Serge Vaudenay
- Procs of the 3rd Intl Wkshp on Fast Soft Encrpt 1996 ISBN 3-540-60865-6
- http ://citeseerx.ist.psu.edu/viewdoc/summary ?doi=10.1.1.31.7942
- Systematic Generation of Cryptographically Robust S-Boxes, Procs 1st ACM Conf on Comp and Comm Security, 1993, Fairfax, VA, USA. ACM, pp. 171-182 - Jennifer Seberry, Xian-Mo Zhang, Yuliang Zheng
- http ://citeseerx.ist.psu.edu/viewdoc/summary ?doi=10.1.1.158.3013
- Cryptography Tutorials - Herong's Tutorial Examples http ://www.herongyang.com/Cryptography
- Blowfish Source Code : http ://www.schneier.com/blowfish-download.html
- Fcts Blowfish (CryptoSys API) : http ://www.cryptosys.net/blowfish.html
- Les chiffres hexa de π, en 4 s_boxes & 1 p_array : http ://www.schneier.com/code/constants.txt
- Test Vectors : http ://www.schneier.com/code/vectors.txt

Bibliographie

- Practical Cryptology and Web Security (2005)- P.K. Yuen
- Applied Cryptography (1996) - Bruce Schneier
- Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish) (1993)- Bruce Schneier
- http ://www.schneier.com/paper-blowfish-fse.html, http ://www.schneier.com/blowfish-bug.txt
- On the Weak Keys of Blowfish. (1996)- Serge Vaudenay
- Procs of the 3rd Intl Wkshp on Fast Soft Encrpt 1996 ISBN 3-540-60865-6
- http ://citeseerx.ist.psu.edu/viewdoc/summary ?doi=10.1.1.31.7942
- Systematic Generation of Cryptographically Robust S-Boxes, Procs 1st ACM Conf on Comp and Comm Security, 1993, Fairfax, VA, USA. ACM, pp. 171-182 - Jennifer Seberry, Xian-Mo Zhang, Yuliang Zheng
- http ://citeseerx.ist.psu.edu/viewdoc/summary ?doi=10.1.1.158.3013
- Cryptography Tutorials - Herong's Tutorial Examples http ://www.herongyang.com/Cryptography
- Blowfish Source Code : http ://www.schneier.com/blowfish-download.html
- Fcts Blowfish (CryptoSys API) : http ://www.cryptosys.net/blowfish.html
- Les chiffres hexa de π, en 4 s_boxes & 1 p_array : http ://www.schneier.com/code/constants.txt
- Test Vectors : http ://www.schneier.com/code/vectors.txt

Fin

There are two kinds of cryptography in this world : cryptography that will stop your kid sister from reading your files, and cryptography that will stop major governments from reading your files. This book is about the latter.

Preface de l'ouvrage *Applied Cryptography* par Bruce Schneier

Il y a deux types de cryptographie dans ce monde : la cryptographie qui va empêcher votre petite sœur de lire vos fichiers, et la cryptographie qui va empêcher les gouvernements des grandes puissances de lire vos fichiers. Ce livre parle de cette dernière.

Fin

There are two kinds of cryptography in this world : cryptography that will stop your kid sister from reading your files, and cryptography that will stop major governments from reading your files. This book is about the latter.
Preface de l'ouvrage *Applied Cryptography* par Bruce Schneier

Il y a deux types de cryptographie dans ce monde : la cryptographie qui va empêcher votre petite sœur de lire vos fichiers, et la cryptographie qui va empêcher les gouvernements des grandes puissances de lire vos fichiers. Ce livre parle de cette dernière.

Fin

There are two kinds of cryptography in this world : cryptography that will stop your kid sister from reading your files, and cryptography that will stop major governments from reading your files. This book is about the latter.
Preface de l'ouvrage *Applied Cryptography* par Bruce Schneier
Il y a deux types de cryptographie dans ce monde : la cryptographie qui va empêcher votre petite sœur de lire vos fichiers, et la cryptographie qui va empêcher les gouvernements des grandes puissances de lire vos fichiers. Ce livre parle de cette dernière.