FOX

–

une nouvelle famille d'algorithmes de chiffrement par bloc

Pascal Junod et Serge Vaudenay

Laboratoire de Sécurité et Cryptographie (LASEC)
École Polytechnique Fédérale de Lausanne

LASEC

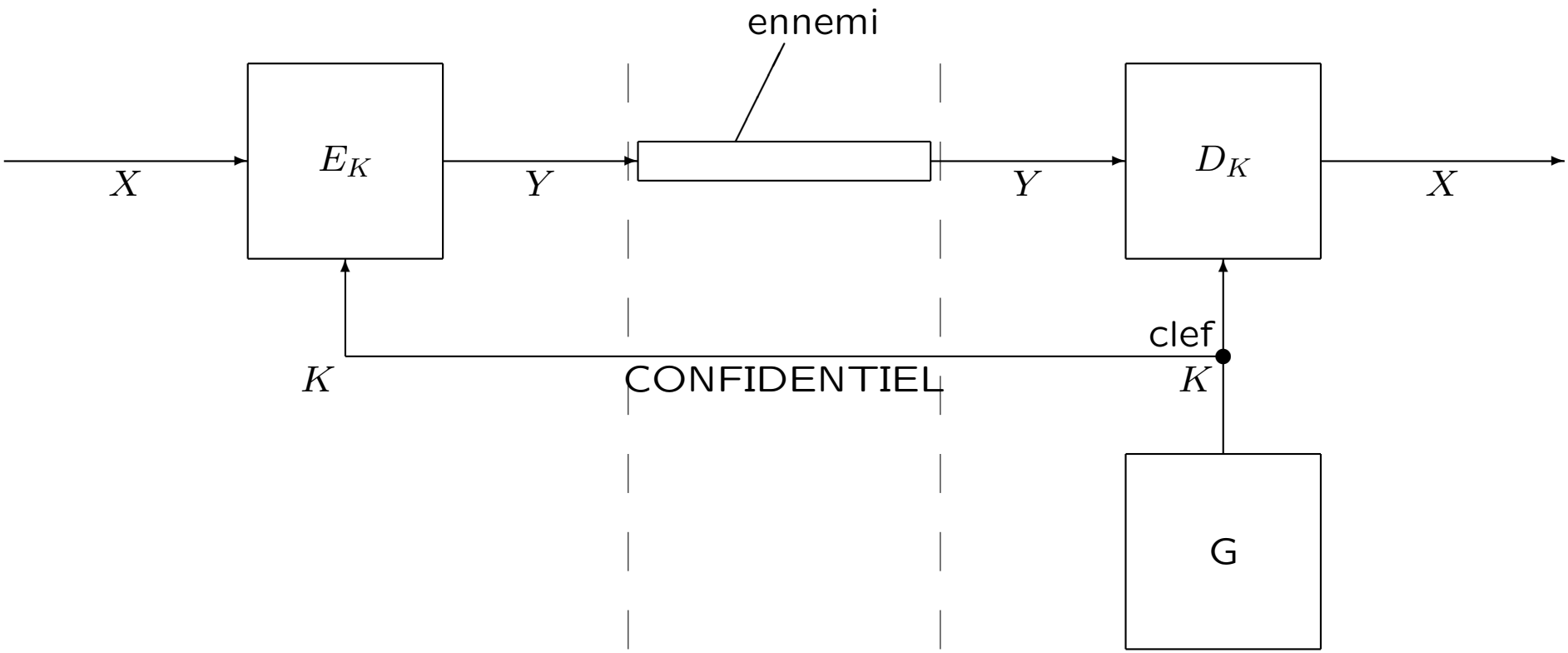
Université J. Fourier, Grenoble (24 novembre 2003)

EPFL
ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Plan de l'exposé

- → Projet FOX
- → Cahier des charges
- → Description
- → Implémentation et performances
- → Sécurité
- → Futur de FOX

Chiffrement Symétrique



Projet FOX (1)

→ Projet proposé par MediaCrypt AG (`http://www.mediacrypt.com`).
→ MediaCrypt AG est propriétaire des droits sur IDEA.
→ En 2010, le brevet protégeant IDEA perdra sa validité.
→ Nouvel algorithme (moderne, sûr, brevetable, flexible et rapide).

Projet FOX (2)

- → Phase 1 : état de l'art (6 mois)
- → Phase 2 : mise au point d'un cahier des charges (3 mois)
- → Phase 3 : développement d'une version α (6 mois)
- → Phase 4 : analyse de la version α (3 mois)
- → Phase 5 : développement d'une version β (3 mois)
- → Phase 6 : expertise externe (3 mois)
- → Phase 7 : réponse à l'expertise (3 mois)
- → Phase 8 : implémentation et finalisation (3 mois)

Cahier des charges (1)

FOX ...

→ ... ne devra pas utiliser de multiplications opérant sur des nombres de plus de 8 bits.

→ ... devra nécessiter moins de 256 octets de mémoire vive sur une smartcard low-cost.

→ ... devra nécessiter moins de 2048 octets de ROM sur une smartcard low-cost.

→ ... devra nécessiter moins de 8000 cycles d'horloge sur un 8051 et moins de 6000 sur une architecture ARM pour chiffrer un bloc de données.

FOX ...

→ ... devra nécessiter moins de 500 cycles d'horloge sur un Intel Pentium III pour chiffrer un bloc.

→ ... ne devra pas être pénalisé par une architecture 64 bits.

Cahier des charges (3)

FOX ...

→ ... devra accepter n'importe quelle taille de clef située entre 0 et 256 bits.

→ ... devra être capable de chiffrer des blocs de 64 bits.

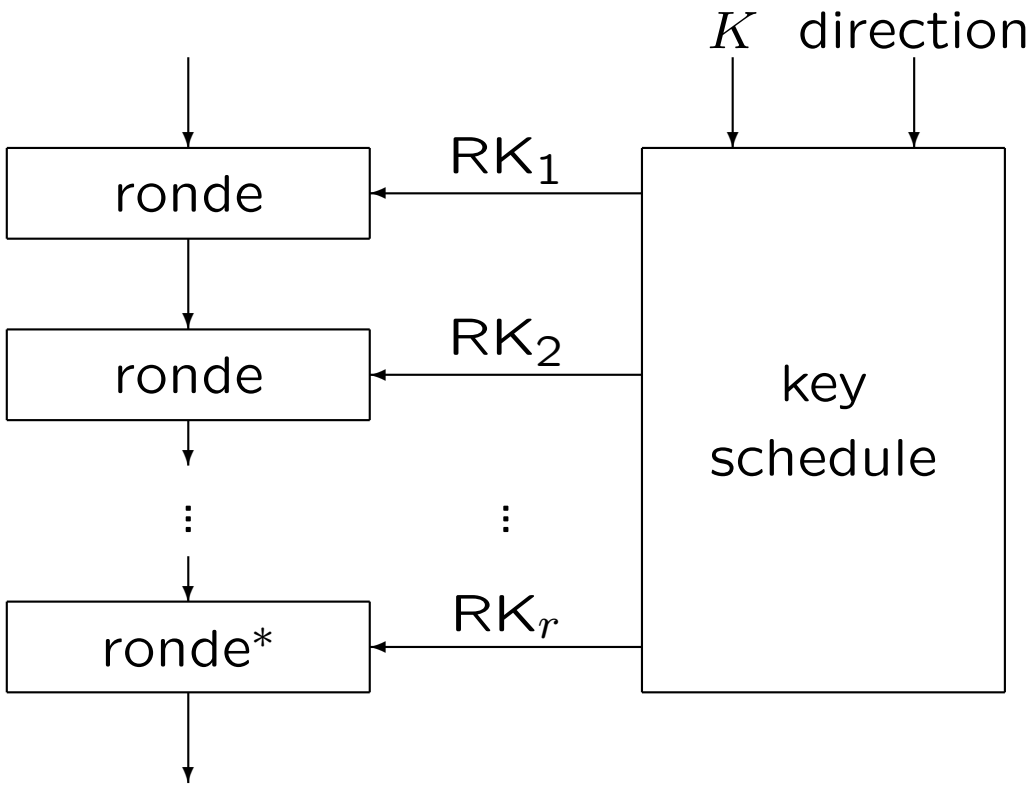
→ ... devra être capable de chiffrer des blocs de 128 bits.

FOX doit être sûr !

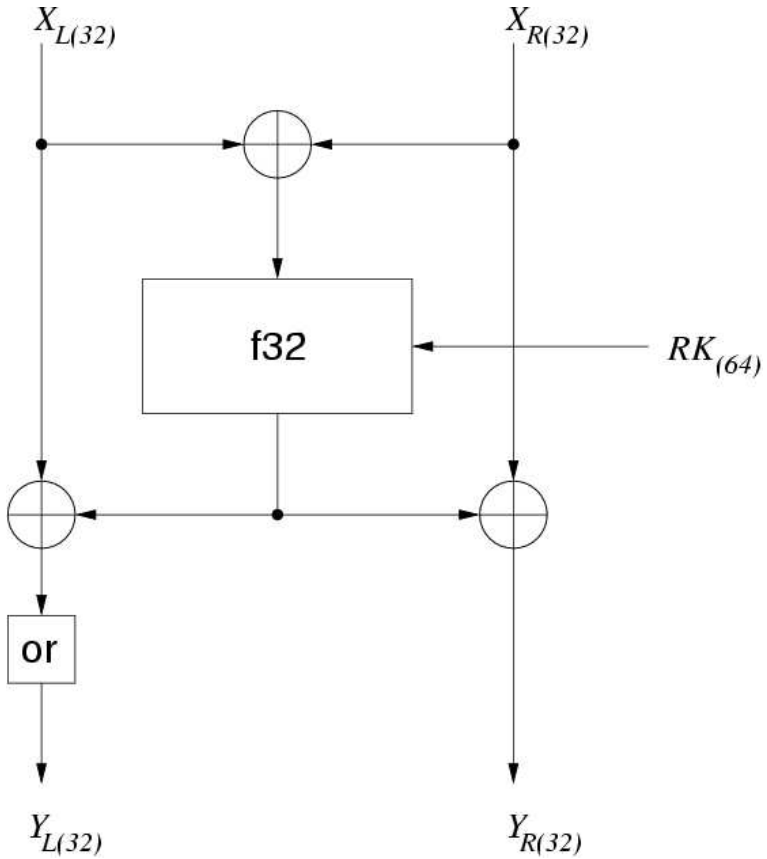
Description de FOX (1)

→ FOX est une famille d'algorithmes de chiffrement par bloc.
→ FOX64 / FOX128
→ Clefs de 0 à 256 bits (par multiple de 8)
→ Nombre de rondes variable (de 12 à 255)

Description de FOX (2)



Description de FOX64 (1)



→ FOX64 est basé sur un *schéma de Lai-Massey* épaulé par un *orthomorphisme*.

→ L'orthomorphisme est un schéma de Feistel à une ronde avec l'identité comme fonction d'étage.

→ On a besoin d'une fonction d'étage

$$\mathrm{f32} : \{0,1\}^{32} \times \{0,1\}^{64} \rightarrow \{0,1\}^{32}$$

Résultat dans le modèle de Luby-Rackoff :

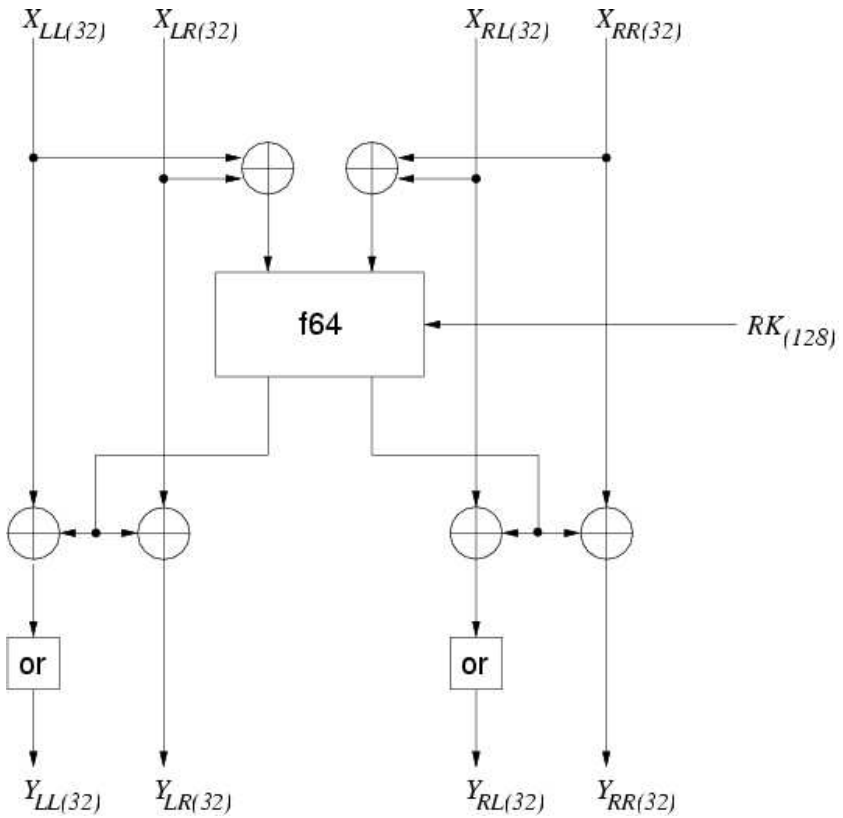
Théorème 1 (Vau99)
Si or est un orthomorphisme, le schéma de Lai-Massey possède les mêmes propriétés d'aléa (après trois rondes) et de super-aléa (après quatre rondes) qu'un schéma de Feistel.

Description de FOX64 (4)

→ Le chiffrement d'un bloc consiste à itérer le schéma de Lai-Massey $r-1$ fois, et de terminer par une ronde où l'orthomorphisme est remplacé par la fonction identité.

→ Le déchiffrement consiste à itérer $r-1$ fois le schéma de Lai-Massey, avec *l'inverse* de l'orthomorphisme, et de terminer par une ronde où l'orthomorphisme est remplacé par la fonction identité (avec les sous-clefs dans l'ordre inverse).

Description de FOX128 (1)



Description de FOX128 (2)

→ FOX128 est basé sur un *schéma de Lai-Massey étendu* épaulé du même orthomorphisme.

→ On a besoin d'une fonction d'étage

$$\mathsf{f64} : \{0,1\}^{64} \times \{0,1\}^{128} \longrightarrow \{0,1\}^{64}$$

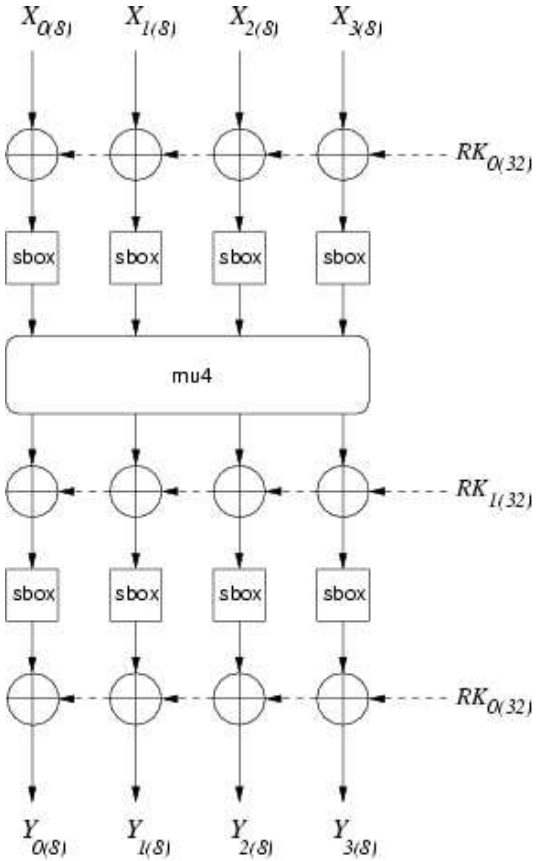
→ Les résultats obtenus dans le modèle de Luby-Rackoff sont également valables.

Description de FOX128 (3)

→ Le chiffrement d'un bloc consiste à itérer le schéma de Lai-Massey étendu $r-1$ une fois, et de terminer par une ronde où l'orthomorphisme est remplacé par la fonction identité.

→ Le déchiffrement consiste à itérer $r-1$ fois le schéma de Lai-Massey étendu, avec *l'inverse* de l'orthomorphisme, et de terminer par une ronde où l'orthomorphisme est remplacé par la fonction identité (avec les sous-clefs dans l'ordre inverse).

Description de f32 (1)



Description de f32 (2)

f32 utilise trois opérations :

→ une opération de substitution (sbox)
→ une opération de diffusion (mu4)
→ une opération d'addition de clef (XOR)

→ sbox est une boîte de substitution générée pseudo-aléatoirement.

→ Elle ne possède pas de description algébrique simple.

→ sbox est un schéma de Lai-Massey utilisant 3 boîtes de substitution sur 4 bits comme fonctions de ronde.

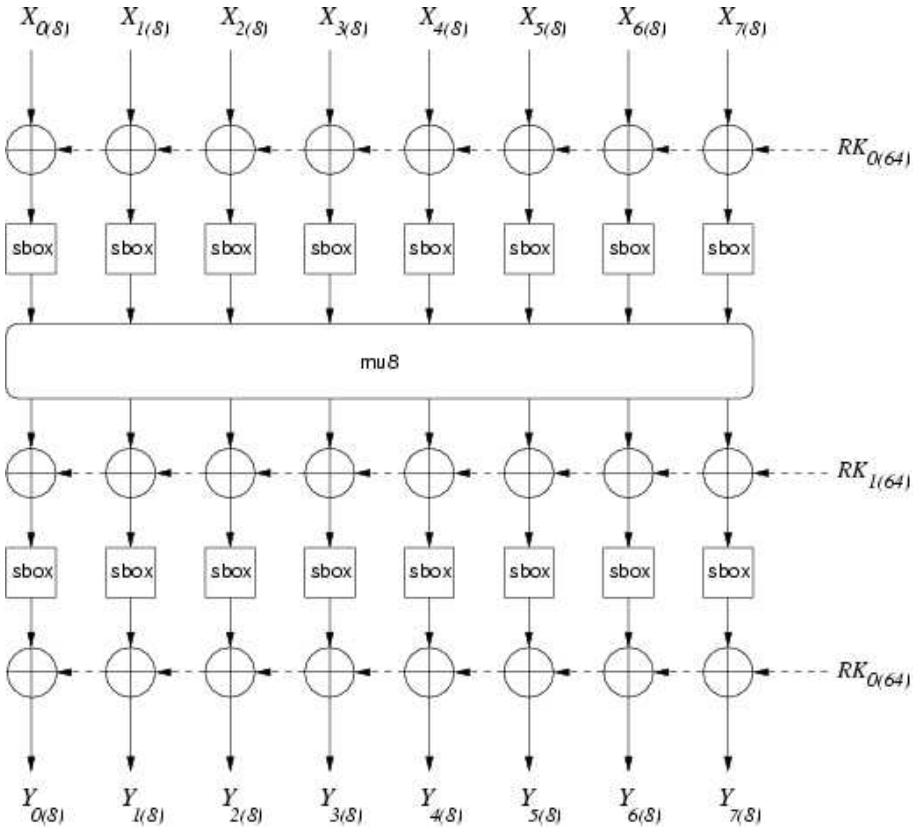
→ $\mathrm{DP}_{\max}^{\mathrm{sbox}} = \mathrm{LP}_{\max}^{\mathrm{sbox}} = 2^{-4}$. Son degré algébrique est égal à 6.

→ La fonction mu4 est une $(4,4)$-multipermutation linéaire. Elle possède des propriétés optimales de diffusion.

→ Pour appliquer mu4, on considère les 32 bits d'entrée comme un vecteur de 4 éléments sur GF $\left(2^8\right)$ et on le multiplie par

$$\text{mu4} \triangleq \begin{pmatrix} \texttt{0x01} & \texttt{0x01} & \texttt{0x01} & \texttt{0x02} \\ \texttt{0x01} & \texttt{0xFD} & \texttt{0x02} & \texttt{0x01} \\ \texttt{0xFD} & \texttt{0x02} & \texttt{0x01} & \texttt{0x01} \\ \texttt{0x02} & \texttt{0x01} & \texttt{0xFD} & \texttt{0x01} \end{pmatrix}$$

Description de f64 (1)



Description de f64 (2)

→ La boîte de substitution est identique à celle utilisée dans f32.

→ La fonction mu8 est une $(8, 8)$-multipermutation linéaire. Elle possède des propriétés optimales de diffusion.

→ Pour appliquer mu8, on considère les 64 bits d'entrée comme un vecteur de 8 éléments sur GF $\left(2^8\right)$ et on le multiplie par

$$\text{mu8} \triangleq \begin{pmatrix} \texttt{0x01} & \texttt{0x01} & \texttt{0x01} & \texttt{0x01} & \texttt{0x01} & \texttt{0x01} & \texttt{0x01} & \texttt{0x03} \\ \texttt{0x01} & \texttt{0x03} & \texttt{0x82} & \texttt{0x02} & \texttt{0x04} & \texttt{0xFC} & \texttt{0x7E} & \texttt{0x01} \\ \texttt{0x03} & \texttt{0x82} & \texttt{0x02} & \texttt{0x04} & \texttt{0xFC} & \texttt{0x7E} & \texttt{0x01} & \texttt{0x01} \\ \texttt{0x82} & \texttt{0x02} & \texttt{0x04} & \texttt{0xFC} & \texttt{0x7E} & \texttt{0x01} & \texttt{0x03} & \texttt{0x01} \\ \texttt{0x02} & \texttt{0x04} & \texttt{0xFC} & \texttt{0x7E} & \texttt{0x01} & \texttt{0x03} & \texttt{0x82} & \texttt{0x01} \\ \texttt{0x04} & \texttt{0xFC} & \texttt{0x7E} & \texttt{0x01} & \texttt{0x03} & \texttt{0x82} & \texttt{0x02} & \texttt{0x01} \\ \texttt{0xFC} & \texttt{0x7E} & \texttt{0x01} & \texttt{0x03} & \texttt{0x82} & \texttt{0x02} & \texttt{0x04} & \texttt{0x01} \\ \texttt{0x7E} & \texttt{0x01} & \texttt{0x03} & \texttt{0x82} & \texttt{0x02} & \texttt{0x04} & \texttt{0xFC} & \texttt{0x01} \end{pmatrix}$$

Algorithmes de key-schedule (1)

Algorithmes de key-schedule (2)

→ Trois algorithmes de key-schedule différents : KS64, KS64h, KS128.

→ Le flux de sous-clefs doit être un flux pseudo-aléatoire de qualité cryptographique.

→ Temps pour calculer toutes les sous-clefs : entre 6 et 12 fois le temps de chiffrer un bloc.

→ Temps identique pour produire les sous-clefs pour le chiffrement et le déchiffrement (calcul à la volée).

→ Recycle les composants de f32 et f64.

Implémentation et Performances (1)

Implémentation	FOX64/12	FOX64/16
Implémentation optimisée en C (gcc)	316	406
Implémentation optimisée en ASM	220	295

Unités : cycles d'horloge pour chiffrer un bloc (Intel Pentium III)

→ Selon NESSIE, FOX64-12 est le quatrième algorithme (chiffrant 64 bits) le plus rapide sur Pentium III, derrière Nimbus, CAST-128, et RC5. FOX64-12 est 19 % plus rapide que Misty1 (choix de NESSIE), 39% plus rapide qu'IDEA, 57% plus rapide que DES et 289 % plus rapide que T-DES.

→ FOX64-16 est 8% plus rapide qu'IDEA.

Implémentation et Performances (2)

Implémentation	FOX128/12	FOX128/16
Implémentation optimisée en C (cc)	440	588

Unités : cycles d'horloge pour chiffrer un bloc (Alpha 21264)

→ Selon NESSIE, FOX128-12 est le troisième algorithme (chiffrant 128 bits) sur Alpha, derrière Nush et AES. FOX128-16 (avec des clefs de 256 bits) est 30 % plus rapide que Camellia (choix de NESSIE).

→ Implémentation sur 8051 de FOX64-12 : 16 octets de RAM, 896 octets de ROM (sous-clefs et données précalculées) et 575 octets de code.

→ Résultat : chiffrement d'un bloc en 2958 cycles d'horloge (3950 pour 16 rondes).

→ Résultats dans le modèle de Luby-Rackoff.

→ Résultats concernant la cryptanalyse linéaire et différentielle.

Théorème 2

La probabilité de n'importe quelle caractéristique (single-path) différentielle (resp. linéaire) dans FOX64/k/r *est bornée par* $\left(\mathrm{DP}_{\max}^{\mathrm{sbox}}\right)^{2r}$ *(resp.* $\left(\mathrm{LP}_{\max}^{\mathrm{sbox}}\right)^{2r}$*). D'une manière similaire, la probabilité de n'importe quelle caractéristique (single-path) différentielle (resp. linéaire) dans* FOX128/k/r *est bornée par* $\left(\mathrm{DP}_{\max}^{\mathrm{sbox}}\right)^{4r}$ *(resp.* $\left(\mathrm{LP}_{\max}^{\mathrm{sbox}}\right)^{4r}$*).*

→ Nombre de ronde minimal : $8 + 2 + 2 = 12$.

Sécurité (2)

→ Autres attaques statistiques (différentielles tronquées, impossibles, d'ordre supérieur, diverses généralisations de cryptanalyse linéaire, attaques intégrales, attaques par interpolation)

→ Attaques algébriques (Courtois-Pieprzyk)

→ *[...] The conclusion of my investigations is that FOX appears to be competently designed. I was unable to find any weaknesses in FOX when used for encryption, and the analysis provides evidence that FOX resist known attacks with a healthy safety margin. [...]* David Wagner, UC Berkeley

→ *[...] The algorithms [...] are supported by a convincing analysis showing their resistance againt linear and differential cryptanalysis. Furthermore, we haven't found any attack on the design of the algorithms. [...]* Jacques Stern, ENS Paris

→ Période d'évaluation (académique / industrielle)

→ Commercialisation

Questions ?

Pour plus de détails :

→ `http://lasecwww.epfl.ch`

→ `http://crypto.junod.info`

→ `http://www.mediacrypt.com`

Merci !