FOX : a New Family of Block Ciphers*

Pascal Junod and Serge Vaudenay

École Polytechnique Fédérale de Lausanne (Switzerland)
{pascal.junod, serge.vaudenay}@epfl.ch

Abstract. In this paper, we describe the design of a new family of block ciphers based on a Lai-Massey scheme, named FOX. The main features of this design, besides a very high security level, are a large implementation flexibility on various platforms as well as high performances. In addition, we propose a new design of strong and efficient key-schedule algorithms. We provide evidence that FOX is immune to linear and differential cryptanalysis, and we discuss its security towards integral cryptanalysis, algebraic attacks, and other attacks.

Key words: Block ciphers, Lai-Massey scheme

1 Introduction

Why do we need another block cipher? First of all, industry is still requesting; second, recent advances in the cryptanalysis field motivate new designs. The AES [1] and NESSIE [27] efforts, among others, have resulted in a number of new proposals of block ciphers. It is noteworthy that there exists a clear trend in direction of lightweight and fast key-schedule algorithms, as well as substitution boxes based on purely algebraic constructions. In a parallel way, we observe that, on the one hand, several of the last published attacks against block ciphers take often advantage of exploiting "simple" key-schedule algorithms (a nice illustration is certainly Muller's attack [24] against Khazad), and, on the other hand, algebraic S-boxes are helpful to Courtois-Pieprzyk algebraic attacks [8], and lead to puzzling properties as shown by [2, 10, 25].

In this paper, we describe the design of a new family of block cipher, named FOX and designed upon the request of MediaCrypt AG [23]. The main features of this design, besides a very high security level, are a large flexibility in terms of use and of implementation on various platforms, as well as high performances. The family consists in two block ciphers, one having a 64-bit block size and the other one a 128-bit block size. Each block cipher allows a variable number of rounds and a variable key size up to 256 bits. The high-level structure is based on a Lai-Massey scheme, while the round functions consist of Substitution-Permutation Networks with no algebraic S-boxes. In addition, we propose a new design of strong and efficient key-schedule algorithms.

* Reprint from: P. Junod and S. Vaudenay. FOX: a new family of block ciphers. To appear in *Selected Areas in Cryptography 2004: Waterloo, Canada, August 9-10, 2004. Revised papers*, *Lecture Notes in Computer Science*. Springer-Verlag.

Our main motivations are the following: our first goal is to offer a serious alternative to block ciphers following present trends; we have explicitely chosen to *avoid a lightweight key-schedule and a pure algebraic construction as S-boxes.* Our second goal is to reach the highest possible flexibility, being in terms of round number, key size, block size and in terms of implementation issues. For instance, we feel that it is still useful to propose a 64-bit block size flavour for backward-compatibility reasons. Finally, our last motivation was to design a family of block ciphers which compares favourably with the performances of the fastest block ciphers on hardware, 8-bit, 32-bit, and 64-bit architectures. This paper is organized as follows: in §2, we give a formal description of the block ciphers, then we successively discuss the rationales in §3 the security foundations in §4 and several implementations aspects in §5. Test vectors are available in Appendix A. The full version of this paper is [14].

Notations: A variable x indexed by i with a length of ℓ bits is denoted $x_{i(\ell)}$. A C-like notation is used for indexing i.e. indices begin with 0.

Representation of GF $\left(2^8\right)$: Some of the internal operations used in FOX are the addition and the multiplication in the GF(2^8) finite field. Elements of the field are polynomials with coefficients in GF(2) in α, a root of the irreducible polynomial $P(\alpha) = \alpha^8 + \alpha^7 + \alpha^6 + \alpha^5 + \alpha^4 + \alpha^3 + 1$: the 8-bit binary string $s = s_{0(1)}||s_{1(1)}||s_{2(1)}||s_{3(1)}||s_{4(1)}||s_{5(1)}||s_{6(1)}||s_{7(1)}$ represents $s_{0(1)}\alpha^7 + s_{1(1)}\alpha^6 + s_{2(1)}\alpha^5 + s_{3(1)}\alpha^4 + s_{4(1)}\alpha^3 + s_{5(1)}\alpha^2 + s_{6(1)}\alpha + s_{7(1)}$.

2 Description

The different members of this block cipher family are denoted as follows:

Name	Block size	Key size	Rounds number
FOX64	64	128	16
FOX128	128	256	16
FOX64/k/r	64	k	r
FOX128/k/r	128	k	r

In FOX64/k/r and FOX128/k/r, the number r of rounds must satisfy $12 \leq r \leq 255$, while the key length k must satisfy $0 \leq k \leq 256$, with k multiple of 8. Note that a generic instance of FOX has 16 rounds.

2.1 High-Level Structure

The 64-bit version of FOX is the $(r-1)$-times iteration of a *round function* lmor64, followed by the application of a slightly modified round function called lmid64. For decryption, we replace lmor64 by lmio64. The encryption $C_{(64)}$ by FOX64/k/r of a 64-bit plaintext $P_{(64)}$ is defined as

$$C_{(64)} = \mathsf{lmid64}(\mathsf{lmor64}(\ldots(\mathsf{lmor64}(P_{(64)}, RK_{0(64)}), \ldots, RK_{r-2(64)}), RK_{r-1(64)})$$

where $RK_{(64r)} = RK_{0(64)}||RK_{1(64)}||\ldots||RK_{r-1(64)}$ is the subkey stream produced by the key schedule algorithm from the key $K_{(k)}$ (see §2.3). The decryption $P_{(64)}$ by $\mathsf{FOX64}/k/r$ of a 64-bit ciphertext $C_{(64)}$ is defined as

$$P_{(64)} = \mathsf{lmid64}(\mathsf{lmio64}(\ldots(\mathsf{lmio64}(C_{(64)}, RK_{r-1(64)}), \ldots, RK_{1(64)}), RK_{0(64)})$$

In the 128-bit version of FOX, we simply replace lmor64, lmid64, and lmio64 by elmor128, elmid128, and elmio128, respectively. lmor64, illustrated in Fig. 1(a), is built as a Lai-Massey scheme [18, 19] combined with an orthomorphism[1] or, as described in [30]. This function transforms a 64-bit input $X_{(64)}$ split in two parts $X_{(64)} = X_{0(32)}||X_{1(32)}$ and a 64-bit round key $RK_{(64)}$ in a 64-bit output $Y_{(64)} = Y_{0(32)}||Y_{1(32)}$ as $Y_{(64)} = \mathsf{or}\left(X_{0(32)} \oplus \phi\right) || \left(X_{1(32)} \oplus \phi\right)$ with $\phi = \mathsf{f32}\left(X_{0(32)} \oplus X_{1(32)}, RK_{(64)}\right)$. lmid64 and lmio64 are defined like for lmor64 but for or, which is replaced by the identity function and io (the inverse of or), respectively. elmor128, illustrated in Fig. 1(b), is built as an *Extended Lai-Massey scheme* combined with two orthomorphisms or. This function transforms a 128-bit input $X_{(128)}$ split in four parts $X_{(128)} = X_{0(32)}||X_{1(32)}||X_{2(32)}||X_{3(32)}$ and a 128-bit round key $RK_{(128)}$ in a 128-bit output $Y_{(128)}$. Let $F_{(64)} = (X_{0(32)} \oplus X_{1(32)})||(X_{2(32)} \oplus X_{3(32)})$. Then,

$$Y_{(128)} = \mathsf{or}\left(X_{0(32)} \oplus \phi_L\right) \Big|\Big| \left(X_{1(32)} \oplus \phi_L\right) \Big|\Big| \mathsf{or}\left(X_{2(32)} \oplus \phi_R\right) \Big|\Big| \left(X_{3(32)} \oplus \phi_R\right)$$

where $\phi_L||\phi_R = \mathsf{f64}\left(F_{(64)}, RK_{(128)}\right)$. In elmid128, resp. elmio128, the two orthomorphisms or are replaced by two identity, resp. io functions. The orthomorphism or is a function taking a 32-bit input $X_{(32)} = X_{0(16)}||X_{1(16)}$ and returning a 32-bit output $Y_{(32)} = Y_{0(16)}||Y_{1(16)}$ which is in fact a one-round Feistel scheme with the identity function as round function; it is defined as $Y_{0(16)}||Y_{1(16)} = X_{1(16)}||\left(X_{0(16)} \oplus X_{1(16)}\right)$.

2.2 Definition of f32 and f64

The round function f32 builds the core of $\mathsf{FOX64}/k/r$. It is built of three main parts: a *substitution* part, denoted sigma4, a *diffusion* part, denoted mu4, and a *round key addition* part (see Fig. 2(a)). Formally, the f32 function takes a 32-bit input $X_{(32)}$, a 64-bit round key $RK_{(64)} = RK_{0(32)}||RK_{1(32)}$ and returns a 32-bit output $Y_{(32)} = \mathsf{sigma4}(\mathsf{mu4}(\mathsf{sigma4}(X_{(32)} \oplus RK_{0(32)})) \oplus RK_{1(32)}) \oplus RK_{0(32)}$.

The function f64, building the core of $\mathsf{FOX128}/k/r$, is very similar to f32 (see Fig. 2(b)): it takes a 64-bit input $X_{(64)}$, a 128-bit round key $RK_{(128)} = RK_{0(64)}||RK_{1(64)}$ and returns $Y_{(64)} = \mathsf{sigma8}(\mathsf{mu8}(\mathsf{sigma8}(X_{(64)} \oplus RK_{0(64)})) \oplus RK_{1(64)}) \oplus RK_{0(64)}$.

The mapping sigma4 (resp. sigma8) consists of 4 (resp. 8) parallel computations of a non-linear bijective mapping (see §3.1 for a description and the table in §B). The diffusive parts of f32 and f64, mu4 and mu8, consider an input

[1] An orthomorphism o on a group $(\mathcal{G}, +)$ is a permutation $x \mapsto \mathsf{o}(x)$ on $\mathcal{G}$ such that $x \mapsto \mathsf{o}(x) - x$ is also a permutation.

$X_{0(8)}||\ldots||X_{n(8)}$ as a vector $(X_{0(8)},\ldots,X_{n(8)})^T$ over GF $\left(2^8\right)$ and multiply it with a matrix to obtain an output vector of the same size. The two matrices are the following:

$$\mathsf{mu4}: \begin{pmatrix} 1 & 1 & 1 & \alpha \\ 1 & z & \alpha & 1 \\ z & \alpha & 1 & 1 \\ \alpha & 1 & z & 1 \end{pmatrix} \qquad \mathsf{mu8}: \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & a \\ 1 & a & b & c & d & e & f & 1 \\ a & b & c & d & e & f & 1 & 1 \\ b & c & d & e & f & 1 & a & 1 \\ c & d & e & f & 1 & a & b & 1 \\ d & e & f & 1 & a & b & c & 1 \\ e & f & 1 & a & b & c & d & 1 \\ f & 1 & a & b & c & d & e & 1 \end{pmatrix}$$

where $z = \alpha^{-1}+1 = \alpha^7+\alpha^6+\alpha^5+\alpha^4+\alpha^3+\alpha^2+1$, and where $a = \alpha+1$, $b = \alpha^7+\alpha$, $c = \alpha$, $d = \alpha^2$, $e = \alpha^7+\alpha^6+\alpha^5+\alpha^4+\alpha^3+\alpha^2$ and $f = \alpha^6+\alpha^5+\alpha^4+\alpha^3+\alpha^2+\alpha$.

2.3 Key-Schedule Algorithms

A FOX key $K_{(k)}$ must have a bit-length k such that $0 \leq k \leq 256$, and k must be a multiple of 8. Depending on the key length and the block size, a member of the FOX block cipher family may use one among three different key-schedule algorithm versions, denoted KS64, KS64h and KS128. The following table defines which variant is used, as well as a constant ek.

Cipher	Block size	Key size	Key-Schedule Version	ek
FOX64	64	$0 \leq k \leq 128$	KS64	128
FOX64	64	$136 \leq k \leq 256$	KS64h	256
FOX128	128	$0 \leq k \leq 256$	KS128	256

The three different versions of the key-schedule algorithm are constituted of four main parts: a padding part, denoted P, expanding $K_{(k)}$ into ek bits, a mixing part, denoted M, a diversification part, denoted D, whose core consists mainly in a linear feedback shift register denoted LFSR, and finally, a non-linear part, denoted NLx, which is actually the only part which differs between the different versions: we denote the three variants NL64, NL64h and NL128. When $ek = k$, the P and M parts are omitted.

Definition of P. The P-part, taking ek and k as input, is a function expanding a bit string by $\frac{ek-k}{8}$ bytes; it concatenates the key $K_{(k)}$ with the first $ek-k$ bits of a constant, pad, giving $PKEY$ as output. The constant pad is defined as being the first 256 bits of the hexadecimal development of $e-2 = \sum_{n=0}^{+\infty} \frac{1}{n!} - 2$:
pad = `0xB7E151628AED2A6ABF7158809CF4F3C762E7160F38B4DA56A784D9045190CFEF`

Definition of M. The M-part mixes the padded key $PKEY$ with the help of a Fibonacci-like recursion. It takes as input a key $PKEY$ with length ek (expressed in bits) seen as an array of $\frac{ek}{8}$ bytes $PKEY_{i(8)}, 0 \leq i \leq \frac{ek}{8}-1$, and is processed according to $MKEY_{i(8)} = PKEY_{i(8)} \oplus \left(MKEY_{i-1(8)} + MKEY_{i-2(8)} \bmod 2^8\right)$, for $0 \leq i \leq \frac{ek}{8}-1$, assuming that $\mathrm{MKEY}_{-2(8)} = \texttt{0x6A}$ and $\mathrm{MKEY}_{-1(8)} = \texttt{0x76}$.

Definition of D *and* LFSR. The D-part takes a key $MKEY$ having a length in bits equal to ek, the total round number r, and the current round number i, with $1 \leq i \leq r$; it modifies $MKEY$ with the help of the output of a 24-bit Linear Shift Feedback Register (LFSR) denoted LFSR. More precisely, $MKEY$ is seen as an array of $\lfloor \frac{ek}{24} \rfloor$ 24-bit values $MKEY_{j(24)}$, with $0 \leq j \leq \lfloor \frac{ek}{24} \rfloor - 1$ concatenated with one residue byte $MKEYRB_{(8)}$ (if $ek = 128$) or two residue bytes $MKEYRB_{(16)}$ (if $ek = 256$), and is modified according to, for $0 \leq j \leq \lfloor \frac{ek}{24} \rfloor - 1$,

$$DKEY_{j(24)} = MKEY_{j(24)} \oplus \mathsf{LFSR}\left((i-1) \cdot \left\lceil \frac{ek}{24} \right\rceil + j, r\right)$$

and the $DKEYRB_{(8)}$ value ($DKEYRB_{(16)}$) is obtained by XORing the most 8 (16) significant bits of $\mathsf{LFSR}((i-1) \cdot \lceil \frac{ek}{24} \rceil + \lfloor \frac{ek}{24} \rfloor, r)$ with $MKEYRB_{(8)}$ ($MKEYRB_{(16)}$), respectively. The remaining 16 (8) bits of the LFSR routine output are discarded. The stream of pseudo-random values is generated by a 24-bit linear feedback shift register, denoted LFSR. It takes two inputs: the total number of rounds r and the number of preliminary clockings. It is based on the following primitive polynomial of degree 24 over GF(2): $\xi^{24} + \xi^4 + \xi^3 + \xi + 1$. The register is initially seeded with the value $\mathtt{0x6A} || r_{(8)} || \overline{r_{(8)}}$, where $r_{(8)}$ is expressed as an 8-bit value.

Definition of NL64*,* NL128*, and* NL64h. We describe here the NL64 and NL128 processes, respectively. Basically, the $DKEY$ value passes through a substitution layer, made of four parallel sigma4 (sigma8) functions, a diffusion layer, made of four parallel mu4 (mu8) functions and a mixing layer called mix64 (mix128), respectively. Then, the constant $\mathsf{pad}_{[0\ldots127]}$ ($\mathsf{pad}_{[0\ldots255]}$) is XORed and the result is flipped if and only if $k = ek$. The result passes through a second substitution layer, it is hashed down to 64 (128) bits using two exclusive-or operations and the resulting value is encrypted first with a lmor64 (elmor128) round function, where the subkey is the left half of the $DKEY$ value and second by a lmid64 (elmid128) function, where the subkey is the right half of $DKEY$. The resulting value is defined to be the 64-bit (128-bit) round key, respectively. Detailed descriptions may be found in Fig. 3(a) and Fig. 3(b), respectively. In the case of NL64h, the process is very similar than for NL128; the difference is that the sigma8 (mu8) functions are replaced by two concatenated sigma4 (mu4) functions, respectively, that mix128 is replaced by mix64h, and that one uses *three* lmor64 round functions, where the respective subkeys are the three left quarters of the $DKEY$ value and a lmid64 function, where the subkey is the rightmost quarter of $DKEY$. The resulting value is defined to be the 64-bit round key. Fig. 3(c) illustrates the NL64h process whose construction is similar to those of NL64 and of NL128.

Definition of mix64*,* mix64h *and* mix128. Given an input vector of four 32-bit values, denoted $X = X_{0(32)} || X_{1(32)} || X_{2(32)} || X_{3(32)}$, the mix64 function consists in processing it by the following relations, resulting in an output vector denoted $Y = Y_{0(32)} || Y_{1(32)} || Y_{2(32)} || Y_{3(32)}$. More formally, mix64 is defined as $Y_{i(32)} =$

$\bigoplus_{j \neq i} X_{j(32)}$ for $0 \leq i, j \leq 3$. The $\mathsf{mix64h}$ and $\mathsf{mix128}$ functions use identical relations operating on 64-bit values.

3 Rationales

3.1 sbox Transformation and Linear Multipermutations

As outlined in the introduction, our primary goal was to avoid a purely algebraic construction for the S-box; a secondary goal was the possibility to implement it in a very efficient way on hardware using ASIC or FPGA technologies. The sbox function is a bijective non-linear mapping on 8-bit values. It consists of a Lai-Massey scheme with 3 rounds taking three different substitution boxes as round function; these "small" S-boxes are denoted S_1, S_2 and S_3, and their content is given in §B. The orthomorphism[2] $\mathsf{or4}$ used in the Lai-Massey scheme is a single round of a 4-bit Feistel scheme with the identity function as round function. We describe now the generation process of the sbox transformation. First a set of three different candidates for small substitution boxes, each having a $\mathrm{LP}_{\max}$ and a $\mathrm{DP}_{\max}$ (with the common notations[3] [22]) smaller than 2^{-2} were pseudo-randomly chosen. Then, the candidate sbox mapping was evaluated and tested regarding its $\mathrm{LP}_{\max}$ and $\mathrm{DP}_{\max}$ values until a good candidate was found. The chosen sbox satisfy $\mathrm{DP}_{\max}^{\mathsf{sbox}} = \mathrm{LP}_{\max}^{\mathsf{sbox}} = 2^{-4}$ and its algebraic degree is equal to 6.

Both $\mathsf{mu4}$ and $\mathsf{mu8}$ are *linear multipermutations.* This kind of construction was early recognized as being optimal for which regards its diffusion properties [28, 29]. A linear application defined by a matrix A is a multipermutation if and only if $\det(A) \neq 0$ and if the determinant of each submatrix of A is different of zero as well. It is well-known that linear multipermutations are equivalent to MDS linear codes (i.e. Maximum Distance Separable codes). Not all constructions are very efficient to implement, especially on low-end smartcard, which have usually very few available memory and computational power (see [15]). In order to be efficiently implementable, the elements of the matrix, which are elements of $\mathrm{GF}(2^8)$, should be efficient to multiply to[4].

3.2 Key-Schedule Algorithms

The FOX key-schedule algorithms were designed with several rationales in mind: first, the function, which depends on the block size, taking a key K and the round number r in output and returning r subkeys should be a cryptographic

[2] The orthormorphism of the third round is omitted.

[3] Where $\mathrm{DP}^{\mathsf{sbox}}(a, b) = \Pr[\mathsf{sbox}(X \oplus a) = \mathsf{sbox}(X) \oplus b]$ and where $\mathrm{LP}^{\mathsf{sbox}}(a, b) = (2 \cdot \Pr[a \cdot X = b \cdot \mathsf{sbox}(X)] - 1)^2$ with $\cdot$ being the inner dot-product on $\mathrm{GF}(2)^n$, $\mathrm{DP}_{\max}^{\mathsf{sbox}} = \max_{a \neq 0, b} \mathrm{DP}^{\mathsf{sbox}}(a, b)$, and $\mathrm{LP}_{\max}^{\mathsf{sbox}} = \max_{a, b \neq 0} \mathrm{LP}^{\mathsf{sbox}}(a, b)$

[4] The only really efficient operations are the addition, the multiplication by α and the division by α. Note that $\alpha^7 + \alpha = \alpha^{-1} + \alpha^{-2}$, $\alpha^7 + \alpha^6 + \alpha^5 + \alpha^4 + \alpha^3 + \alpha^2 = \alpha^{-1}$, and that $\alpha^6 + \alpha^5 + \alpha^4 + \alpha^3 + \alpha^2 + \alpha = \alpha^{-2}$.

pseudo-random, collision resistant and one-way function. Second, the sequence of subkeys should be generated in any direction without any complexity penalty. Third, all the bytes of $MKEY$ should be randomized even when the key size is strictly smaller than ek. Finally, the key-schedule algorithm should resist *related-cipher attacks* as described by Wu in [33].

We are convinced that "strong" key-schedule algorithms have significant advantages in terms of security, even if the price to pay is a smaller key agility; in the case of FOX, we believe that the time needed to compute the subkeys (about equal to the time needed to encrypt 6 blocks[5] of data) remains acceptable. The second central property of FOX key-schedule algorithms is ensured by the LFSR construction. The third property is ensured by our "Fibonacci-like" construction (which is a bijective mapping). Furthermore, $MKEY$ is expanded by XORing constants depending on r and ek with *no overlap* on these constants sequences (this was checked experimentally). Finally, the fourth property is ensured by the dependency of the subkey sequence to the actual round number of the algorithm instance for which the sequence will be used.

4 Security Foundations

4.1 Luby-Rackoff-like Security

Although less popular than the Feistel scheme or SPN structures, the Lai-Massey scheme offers similar (super-) pseudorandomness and decorrelation inheritance properties, as was demonstrated by Vaudenay [30]. Note that we will indifferently use the term "Lai-Massey scheme" to denote both versions, as we can see the Extended Lai-Massey scheme as a Lai-Massey scheme[6]. From this point, we will make use of the following notation: given an orthomorphism o on a group $(\mathcal{G}, +)$ and given r functions $\mathsf{f}_1, \mathsf{f}_2, \ldots, \mathsf{f}_r$ on $\mathcal{G}$, we note a r-rounds Lai-Massey scheme using the r functions and the orthomorphism by $\Lambda^{\mathsf{o}}(\mathsf{f}_1, \ldots, \mathsf{f}_r)$. Then the following results are two Luby-Rackoff-like [21] results on the Lai-Massey scheme. We refer to [30, 31] for proofs thereof.

Theorem 1 *1. Let f_1^*, f_2^* and f_3^* be three independent random functions uniformly distributed on a group $(\mathcal{G}, +)$. Let o be an orthomorphism on $\mathcal{G}$. For any distinguisher[7] limited to d chosen plaintexts, where $g = |\mathcal{G}|$ denotes the cardinality of the group, between $\Lambda^{\mathsf{o}}(\mathsf{f}_1^*, \mathsf{f}_2^*, \mathsf{f}_3^*)$ and a uniformly distributed random permutation c^*, we have* $\mathrm{Adv}(\Lambda^{\mathsf{o}}(\mathsf{f}_1^*, \mathsf{f}_2^*, \mathsf{f}_3^*), \mathsf{c}^*) \leq d(d-1)(g^{-1}+g^{-2})$.

[5] In the case of FOX64 with keys strictly larger than 128 bit, it takes the time to encrypt 12 blocks of data.

[6] We can prove this by swapping the two inner inputs and noting that the function $(x, y) \mapsto \mathsf{or32}(x)||\mathsf{or32}(y)$ builds an orthomorphism

[7] A distinguisher $\mathcal{A}$ is a probabilistic Turing machine with unlimited computational power. It has access to an oracle $\mathcal{O}$ and can send it a *limited* number of queries. At the end, the distinguisher must output "0" or "1". The advantage for distinguishing a random function f from a random function g is defined by $\mathrm{Adv}(\mathsf{f}, \mathsf{g}) = \left|\Pr\left[\mathcal{A}^{\mathcal{O}=\mathsf{f}} = 1\right] - \Pr\left[\mathcal{A}^{\mathcal{O}=\mathsf{g}} = 1\right]\right|$.

2. If $\mathsf{f}_1, \ldots, \mathsf{f}_r$ *are* $r \geq 3$ *independent random functions on a group* $(\mathcal{G}, +)$ *of order* g *such that* $\mathrm{Adv}(\mathsf{f}_i, \mathsf{f}_i^*) \leq \frac{\epsilon}{2}$ *for any adaptive distinguisher between* f_i *and* f_i^* *limited to* d *queries for* $1 \leq i \leq r$ *and if* o *is an orthomorphism on* $\mathcal{G}$*, we have* $\mathrm{Adv}(\Lambda^{\mathsf{o}}(\mathsf{f}_1, \ldots, \mathsf{f}_r), \mathsf{c}^*) \leq \frac{1}{2}(3\epsilon + d(d-1)(2g^{-1} + g^{-2}))^{\lfloor \frac{r}{3} \rfloor}$.

Basically, the first result proves that the Lai-Massey scheme provides pseudorandomness on three rounds unless the f_i's are weak , like for the Feistel scheme [9]. Super-pseudorandomness corresponds to cases where a distinguisher can query chosen ciphertexts as well; in this scenario, the previous result holds when we consider $\Lambda^{\mathsf{o}}(\mathsf{f}_1^*, \ldots, \mathsf{f}_4^*)$ with a fourth round. The second result proves that the decorrelation bias of the round functions of a Lai-Massey scheme is inherited by the whole structure: provided the f_i's are strong, so is the Lai-Massey scheme[8]; in other words, a potential cryptanalysis will not be able to exploit the Lai-Massey's scheme only, but it will have to take advantage of weaknesses of the round functions' internal structure.

4.2 Linear and Differential Cryptanalysis

It is possible to prove some important results about the security of both f32 and f64 functions towards linear and differential cryptanalysis, too. As these functions may be viewed as classical *Substitution-Permutation Network* constructions, we will refer to some well-known results on their resistance towards linear and differential cryptanalysis proved in [12] by Hong *et al.* As the mu4 (mu8) mapping is a $(4, 4)$-multipermutation ($(8, 8)$-multipermutation), one is ensured that at least $n_d = 5$ ($n_d = 9$) S-boxes before and after mu4 will be active, respectively. Then, by Theorem 1 of [12], we have $\mathrm{DP}_{\max}^{\mathsf{f32}} \leq (\mathrm{DP}_{\max}^{\mathsf{sbox}})^4$ and $\mathrm{DP}_{\max}^{\mathsf{f64}} \leq (\mathrm{DP}_{\max}^{\mathsf{sbox}})^8$. Similar results can be obtained with respect to linear cryptanalysis. By taking into account the fact that in a Lai-Massey scheme, any differential or linear characteristic on two rounds must involve *at least one round function*, we obtain the following result; its complete proof can be found in [14].

Theorem 2 *The differential (resp. linear) probability of any single-path characteristic in* $\mathsf{FOX64}/k/r$ *is upper bounded by* $(\mathrm{DP}_{\max}^{\mathsf{sbox}})^{2r}$ *(resp.* $(\mathrm{LP}_{\max}^{\mathsf{sbox}})^{2r}$*). Similarly, the bounds are* $(\mathrm{DP}_{\max}^{\mathsf{sbox}})^{4r}$ *(resp.* $(\mathrm{LP}_{\max}^{\mathsf{sbox}})^{4r}$*) for* $\mathsf{FOX128}/k/r$.

Since $\mathrm{DP}_{\max}^{\mathsf{sbox}} = \mathrm{LP}_{\max}^{\mathsf{sbox}} = 2^{-4}$, we conclude that it is impossible to find any useful differential or linear characteristic after 8 rounds for both FOX64 and FOX128. Hence, a minimal number of 12 rounds provides a minimal safety margin.

4.3 Integral Attacks

Integral attacks [17] apply to ciphers operating on well-aligned data, like SPN structures. As the round functions of FOX are SPNs, one can wonder whether it

[8] One should not misinterpret these results in terms of the overall block cipher security: FOX's round functions are far to be indistinguishable from random functions, as it is the case of DES round functions, for instance: the fact that DES is vulnerable to linear and differential cryptanalysis does not contradict Luby-Rackoff results.

is possible to find an integral distinguisher on the whole structure. We consider now the case of FOX64: let us denote the input bytes by $X_{i(8)}$ with $0 \leq i \leq 7$. Let $X_{3(8)} = a$, $X_{7(8)} = a \oplus c$, and $X_{i(8)} = c$ for $i = 0, 1, 2, 4, 5, 6$, where c is a constant. We consider plaintext structures $x^{(j)}$ for $1 \leq j \leq 256$ where a takes all 256 possibles byte values. Let us denote the output of the third round lmid64 by $Y_{i(8)}$ with $0 \leq i \leq 7$. Then, $\bigoplus_{j=1}^{256} Y_{i(8)}^{(j)} = \bigoplus_{j=1}^{256} Y_{i+4(8)}^{(j)}$ for $0 \leq i \leq 3$. Note that we have still two such equalities if we replace the last round by a lmor64 round. This integral distinguisher[9] can be used to break (four, five) six rounds of FOX64 (by guessing the one, two, or three last round keys and testing the integral criterion for each subkey candidate on a few structures of plaintexts) with a complexity of about (2^{64}, 2^{128}) 2^{192} operations. A similar property may be used to break up to 4 rounds of FOX128 (by guessing the last round key) with a complexity of about 2^{128} operations.

4.4 Other Attacks

Statistical Attacks. Due to the very high diffusion properties of FOX's round functions, the high algebraic degree of the sbox mapping, and the high number of rounds, we are strongly convinced that FOX will resist to known variants of linear and differential cryptanalysis (like differential-linear cryptanalysis [4, 20], boomerang [32] and rectangle attacks [5]), as well as generalizations thereof, like Knudsen's truncated and higher-order differentials [16], impossible differentials [3], and Harpes' partitioning cryptanalysis [11], for instance.

Slide and Related-Key Attacks. Slide attacks [6, 7] exploit periodic key-schedule algorithms, which is not a property of FOX's key-schedule algorithms. Furthermore, due to very good diffusion and the high non-linearity of the key-schedule, related-key attacks are very unlikely to be effective against FOX.

Interpolation and Algebraic Attacks. Interpolation attacks [13] take advantage of S-boxes exhibiting a simple algebraic structure. Since FOX's non-linear mapping sbox does not possess any simple relation over GF(2) or GF(2^8), such attacks are certainly not effective. One of our main concerns was to avoid a pure algebraic construction for the sbox mapping, as it is the case for a large number of modern designs of block ciphers. Although such S-boxes have many interesting non-linear properties, they probably form the best conditions to express a block cipher as a system of sparse, over-defined low-degree multivariate polynomial equations over GF(2) or GF(2^8); this fact may lead to effective attacks, as argued by Courtois and Pieprzyk in [8]. Not choosing an algebraic construction for sbox does not necessarily ensure security towards algebraic attacks. Note that we base our non-linear mapping on "small" permutations, mapping 4 bits to 4 bits, and that, according to [8], *any* such mapping can always be written as an overdefined system of *at least* 21 quadratic equations. Indeed, we checked that S_1, S_2, and

[9] Note that one could extend it to four rounds using large precomputed tables, and thus reduce the overall complexity by a factor of 2^{64}.

S_3 cannot be described by a system with more than 21 quadratic equations over GF (2); furthermore, we are not aware of any quadratic relation over GF (2^8) for sbox. Following the very same methodology than [8], it appears that XSL attacks *would* break members of the FOX family within a complexity[10] of 2^{171} to 2^{192}, depending on the block size and the round numbers. However, one should interpret these figures with an extreme care: on the one hand, the real complexity of XSL attacks is by no means clear at the time of writing and is the subject of much controversy [26]; on the other hand, we feel that the advantages of a small hardware footprint overcome such a (possible) security decrease.

5 Implementation Issues

Hardware. The size of the small S-boxes allows to implement FOX very efficiently on hardware using ASIC or FPGA technologies (which can usually implement any 4-bit to 4-bit mapping very efficiently). Projects are currently in process. We expect that FOX results in very high performances on hardware.

8-bit Platforms. Obviously, the most intensive computations are related to the evaluation of the sbox mapping and of mu4 and mu8. Different strategies may be applied: when extremely few memory is available, one computes on-the-fly the sbox mapping, as it is described in §3.1, and all the operations in GF (2^8). The sole needed constants are the small substitution boxes S_1, S_2 and S_3 (see §B) and the constants needed by the key-schedule algorithm. A significant speed gain can be obtained if one precomputes the sbox mapping, the finite field operations being all computed dynamically. A third possibility is to precompute two more mappings: multiplication in GF (2^8) by α and by α^{-1}. Finally, in the case of FOX128, a further speed gain may be obtained by tabulating two more mappings: multiplication by α^2 and by α^{-2}.

32/64-bit Platforms. The f32 and f64 functions can be implemented very efficiently using a classical combinations of table-lookups and XORs. For a fully precomputed implementation, one needs 8'192 bytes of memory space for FOX64, as well as 32'768 bytes for FOX128. Depending on the target processor, the nearest cache (i.e. the fastest memory) size may be smaller than 32 kB. In this case, one can spare half of the space (at the cost of a few masking operations) by noting that the S-boxes are "embedded" in the tables combining the S-box and the diffusion layer; this allows to reduce the fast memory needs to 4'096 and 16'384 bytes, respectively.

Performance Results The following table summarizes the results obtained so far by our optimized implementations of the FOX family (in clock cycles to encrypt one block, with precomputed subkeys):

[10] Under the unchecked hypothesis that XSL can use Gaussian elimination within a complexity equal to $n^{2.376}$.

Cipher	Architecture	Implementation	$r = 12$	$r = 16$
FOX64/k/r	Intel Pentium 3	C (gcc)	316	406
FOX64/k/r	Intel Pentium 3	ASM	220	295
FOX64/k/r	Intel Pentium 4	C (gcc)	388	564
FOX64/k/r	AMD Athlon-XP	C (gcc)	306	390
FOX64/k/r	Alpha 21264	C (Compaq cc)	360	480
FOX128/k/r	Intel Pentium 3	C (gcc)	636	840
FOX128/k/r	AMD Athlon-XP	C (gcc)	544	748
FOX128/k/r	Alpha 21264	C (Compaq cc)	440	588

We note that FOX64 is extremely fast on 32-bit architectures, while FOX128 is competitive on 64-bit architectures. Namely, according to the Nessie project [27], FOX64/12 is the fourth fastest 64-bit block cipher on Pentium 3 behind Nimbus, CAST-128 and RC5. It is 19% faster than Misty1 (NESSIE's choice), 39% faster than IDEA, 57% faster than DES and about three times faster than TDES. The generic version of FOX64 (with 16 rounds) is still 8% faster than IDEA. On Alpha 21264, a 64-bit architecture, FOX128/12 is the third fastest block cipher behind Nush and AES, according to [27], while FOX128 (16 rounds) with 256-bit keys is still 30% faster than Camellia, which is one of NESSIE's choices.

Finally, we have an implementation of FOX64/12 (resp. FOX64/16) on 8051, a typical low-cost 8-bit architecture, needing 16 bytes of RAM, 896 bytes of ROM (precomputed data and precomputed subkeys) and 575 bytes of code size encrypting one block in 2958 (resp. 3950) clock cycles.

6 Conclusion

Obviously, proposing a new block cipher family leads to new open problems. We *strongly* encourage the development of attacks against full or reduced versions of any member of the FOX family.

Another very interesting open problem is the definition of new linear multipermutations which can be implemented efficiently on low-cost 8-bit smartcards. Some proposals have been done in connection with the design of block ciphers based on SPNs, where the inverse multipermutation also has to be implemented; using them in a self-inverting structure, *e.g.* a Feistel or a Lai-Massey scheme, allows to relax this condition. Hence, the linear mapping can be optimized

Acknowledgments

We would like to thank MediaCrypt AG, for having motivated and supported this work, as well as Jacques Stern and David Wagner for their review of a preliminary version of FOX.

References

1. AES Homepage. http://csrc.nist.gov/encryption/aes/.

2. E. Barkan and E. Biham. In how many ways can you write Rijndael? In Y. Zheng, editor, *Advances in Cryptology – ASIACRYPT'02*, volume 2501 of *Lecture Notes in Computer Science*, pages 160–175. Springer-Verlag, 2002.
3. E. Biham, A. Biryukov, and A. Shamir. Cryptanalysis of Skipjack reduced to 31 rounds using impossible differentials. In J. Stern, editor, *Advances in Cryptology - EUROCRYPT'99*, volume 1592 of *Lecture Notes in Computer Science*, pages 12–23. Springer-Verlag, 1999.
4. E. Biham, O. Dunkelman, and N. Keller. Enhancing differential-linear cryptanalysis. In Y. Zheng, editor, *Advances in Cryptology – ASIACRYPT'02*, volume 2501 of *Lecture Notes in Computer Science*, pages 254–266. Springer-Verlag.
5. E. Biham, O. Dunkelman, and N. Keller. The rectangle attack - rectangling the Serpent. In B. Pfitzmann, editor, *Advances in Cryptology – EUROCRYPT'01*, volume 2045 of *Lecture Notes in Computer Science*, pages 340–357. Springer-Verlag, 2001.
6. A. Biryukov and D. Wagner. Slide attacks. In L. Knudsen, editor, *Fast Software Encryption: 6th International Workshop, FSE'99*, volume 1636 of *Lecture Notes in Computer Science*, pages 245–259. Springer-Verlag, 1999.
7. A. Biryukov and D. Wagner. Advanced slide attacks. In B. Preneel, editor, *Advances in Cryptology - EUROCRYPT'00*, volume 1807 of *Lecture Notes in Computer Science*, pages 589–606. Springer-Verlag, 2000.
8. N. Courtois and J. Pieprzyk. Cryptanalysis of block ciphers with overdefined systems of equations. In Y. Zheng, editor, *Advances in Cryptology – ASIACRYPT'02*, volume 2501 of *Lecture Notes in Computer Science*, pages 267–287. Springer-Verlag, 2002.
9. H. Feistel. Cryptography and data security. *Scientific American*, 228(5):15–23, 1973.
10. N. Ferguson, R. Schroeppel, and D. Whiting. A simple algebraic representation of Rijndael. In S. Vaudenay and A. Youssef, editors, *Selected Areas in Cryptography: SAC 2001*, volume 2259 of *Lecture Notes in Computer Science*, pages 103–111. Springer-Verlag, 2001.
11. C. Harpes and J. Massey. Partitioning cryptanalysis. In E. Biham, editor, *Fast Software Encryption: 4th International Workshop, FSE'97*, volume 1267 of *Lecture Notes in Computer Science*, pages 13–27. Springer-Verlag.
12. S. Hong, S. Lee, J. Lim, J. Sung, D. Cheon, and I. Cho. Provable security against differential and linear cryptanalysis for the SPN structure. In B. Schneier, editor, *Fast Software Encryption: 7th International Workshop, FSE 2000*, volume 1978 of *Lecture Notes in Computer Science*, pages 273–283. Springer-Verlag, 2001.
13. T. Jakobsen and L. Knudsen. The interpolation attack against block ciphers. In E. Biham, editor, *Fast Software Encryption: 4th International Workshop, FSE'97*, volume 1267 of *Lecture Notes in Computer Science*, pages 28–40. Springer-Verlag, 1997.
14. P. Junod and S. Vaudenay. *FOX specifications version 1.1.* Technical Report EPFL/IC/2004/75, École Polytechnique Fédérale, Lausanne, Switzerland, 2004.
15. P. Junod and S. Vaudenay. Perfect diffusion primitives for block ciphers – building efficient MDS matrices. In *Proceedings of SAC'04*. Springer-Verlag, 2004.
16. L. Knudsen. Truncated and higher order differentials. In B. Preneel, editor, *Fast Software Encryption: Second International Workshop*, volume 1008 of *Lecture Notes in Computer Science*, pages 196–211. Springer-Verlag, 1995.
17. L. Knudsen and D. Wagner. Integral cryptanalysis (extended abstract). In J. Daemen and V. Rijmen, editors, *Fast Software Encryption: 9th International Work-*

shop, FSE 2002, volume 2365 of *Lecture Notes in Computer Science*, pages 112–127. Springer-Verlag, 2002.

18. X. Lai. *On the design and security of block ciphers*, volume 1 of *ETH Series in Information Processing*. Hartung-Gorre Verlag, 1992.
19. X. Lai and J. Massey. A proposal for a new block encryption standard. In I. Damgård, editor, *Advances in Cryptology - EUROCRYPT'90*, volume 473 of *Lecture Notes in Computer Science*, pages 389–404. Springer-Verlag, 1991.
20. K. Langford and E. Hellman. Differential-linear cryptanalysis. In Y. Desmedt, editor, *Advances in Cryptology - CRYPTO'94*, volume 839 of *Lecture Notes in Computer Science*, pages 17–25. Springer-Verlag, 1994.
21. M. Luby and C. Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM Journal on Computing*, 17(2):373–386, 1988.
22. M. Matsui. New block encryption algorithm MISTY. In E. Biham, editor, *Fast Software Encryption: 4th International Workshop, FSE'97*, volume 1267 of *Lecture Notes in Computer Science*, pages 53–67. Springer-Verlag.
23. MediaCrypt AG. Website http://www.mediacrypt.com.
24. F. Muller. A new attack against Khazad. In C. Laih, editor, *Advances in Cryptology - ASIACRYPT'03*, volume 2894 of *Lecture Notes in Computer Science*, pages 347 – 358. Springer-Verlag, 2003.
25. S. Murphy and M. Robshaw. Essential algebraic structure within the AES. In M. Yung, editor, *Advances in Cryptology – CRYPTO'02*, volume 2442 of *Lecture Notes in Computer Science*, pages 1–16. Springer-Verlag, 2002.
26. S. Murphy and M. Robshaw. Comments on the security of the AES and the XSL technique. *Electronic Letters*, 39(1):36–38. 2003.
27. NESSIE Homepage. https://www.cryptonessie.org.
28. C. Schnorr and S. Vaudenay. Black box cryptanalysis of hash networks based on multipermutations. In A. De Santis, editor, *Advances in Cryptology - EUROCRYPT'94*, volume 950 of *Lecture Notes in Computer Science*, pages 47–57. Springer-Verlag, 1995.
29. S. Vaudenay. On the need for multipermutations: cryptanalysis of MD4 and SAFER. In B. Preneel, editor, *Fast Software Encryption: Second International Workshop*, volume 1008 of *Lecture Notes in Computer Science*, pages 286–297. Springer-Verlag, 1995.
30. S. Vaudenay. On the Lai-Massey scheme. In K. Lam, T. Okamoto, and C. Xing, editors, *Advances in Cryptology - ASIACRYPT'99*, volume 1716 of *Lecture Notes in Computer Science*, pages 8–19. Springer-Verlag, 2000.
31. S. Vaudenay. Decorrelation: a theory for block cipher security. *Journal of Cryptology*, 16(4):249–286, 2003.
32. D. Wagner. The boomerang attack. In L. Knudsen, editor, *Fast Software Encryption: 6th International Workshop, FSE'99*, volume 1636 of *Lecture Notes in Computer Science*, pages 156–170. Springer-Verlag, 1999.
33. H. Wu. Related-cipher attacks. In R. Deng, S. Qing, F. Bao, and J. Zhou, editors, *Information and Communications Security: 4th International Conference, ICICS 2002*, volume 2513 of *Lecture Notes in Computer Science*, pages 447–455. Springer-Verlag, 2002.

A Test Vectors

An implementation of FOX can be validated using the following test vectors. The ciphertexts corresponding to the plaintext `0x0123456789ABCDEF`, respectively

0x0123456789ABCDEFFEDCBA9876543210 are given for two different key lengths, for FOX64 and FOX128, respectively.

```
-----------------------------------------------------------------------------------------
FOX64/16/128 K  : 00112233 44556677 8899AABB CCDDEEFF
FOX64/16/128 C  : B85D6B76 6DCE952E
-----------------------------------------------------------------------------------------
FOX64/16/256 K  : 00112233 44556677 8899AABB CCDDEEFF FFEEDDCC BBAA9988 77665544 33221100
FOX64/16/256 C  : BB654D30 11DB367E
-----------------------------------------------------------------------------------------
FOX128/16/128 K : 00112233 44556677 8899AABB CCDDEEFF
FOX128/16/128 C : 849E0F06 82F50CD5 88AE0730 06A10BEE
-----------------------------------------------------------------------------------------
FOX128/16/256 K : 00112233 44556677 8899AABB CCDDEEFF FFEEDDCC BBAA9988 77665544 33221100
FOX128/16/256 C : 45CCB103 0F67B768 247F5302 66BC4996
-----------------------------------------------------------------------------------------
```

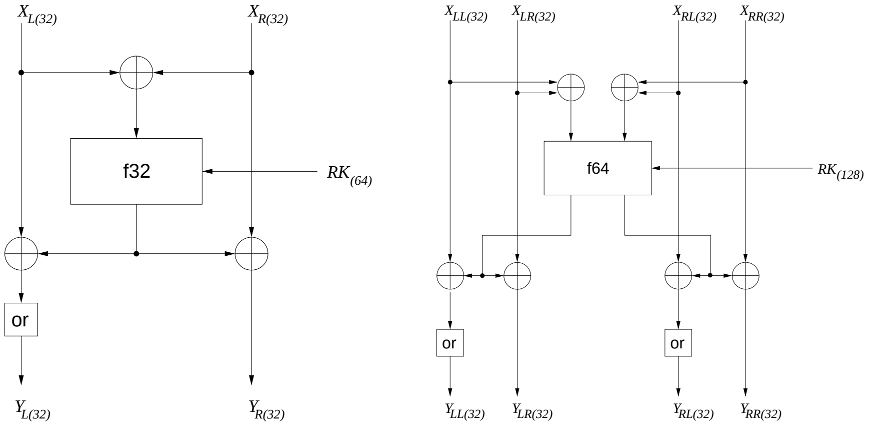
B sbox Definition

The three small S-boxes S_1, S_2, and S_3, as well as the full S-box, are defined in the following tables:

x	0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7	0x8	0x9	0xA	0xB	0xC	0xD	0xE	0xF
$\mathsf{S}_1(x)$	0x2	0x5	0x1	0x9	0xE	0xA	0xC	0x8	0x6	0x4	0x7	0xF	0xD	0xB	0x0	0x3
$\mathsf{S}_2(x)$	0xB	0x4	0x1	0xF	0x0	0x3	0xE	0xD	0xA	0x8	0x7	0x5	0xC	0x2	0x9	0x6
$\mathsf{S}_3(x)$	0xD	0xA	0xB	0x1	0x4	0x3	0x8	0x9	0x5	0x7	0x2	0xC	0xF	0x0	0x6	0xE

One should read the next table in that way: to compute sbox(0x4C), one selects first the row named 4. (i.e. the fifth row), and then one selects the column named .C (i.e. the thirteenth column) and we get finally sbox(0x4C) = 0x15.

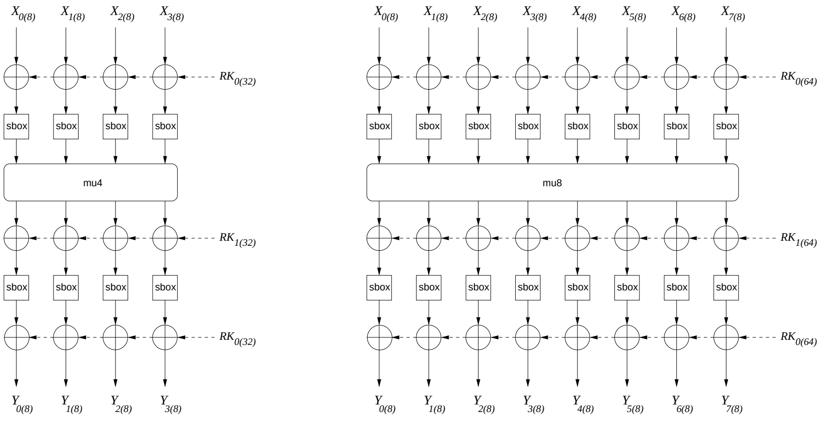
sbox	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.A	.B	.C	.D	.E	.F
0.	5D	DE	00	B7	D3	CA	3C	0D	C3	F8	CB	8D	76	89	AA	12
1.	88	22	4F	DB	6D	47	E4	4C	78	9A	49	93	C4	C0	86	13
2.	A9	20	53	1C	4E	CF	35	39	B4	A1	54	64	03	C7	85	5C
3.	5B	CD	D8	72	96	42	B8	E1	A2	60	EF	BD	02	AF	8C	73
4.	7C	7F	5E	F9	65	E6	EB	AD	5A	A5	79	8E	15	30	EC	A4
5.	C2	3E	E0	74	51	FB	2D	6E	94	4D	55	34	AE	52	7E	9D
6.	4A	F7	80	F0	D0	90	A7	E8	9F	50	D5	D1	98	CC	A0	17
7.	F4	B6	C1	28	5F	26	01	AB	25	38	82	7D	48	FC	1B	CE
8.	3F	6B	E2	67	66	43	59	19	84	3D	F5	2F	C9	BC	D9	95
9.	29	41	DA	1A	B0	E9	69	D2	7B	D7	11	9B	33	8A	23	09
A.	D4	71	44	68	6F	F2	0E	DF	87	DC	83	18	6A	EE	99	81
B.	62	36	2E	7A	FE	45	9C	75	91	0C	0F	E7	F6	14	63	1D
C.	0B	8B	B3	F3	B2	3B	08	4B	10	A6	32	B9	A8	92	F1	56
D.	DD	21	BF	04	BE	D6	FD	77	EA	3A	C8	8F	57	1E	FA	2B
E.	58	C5	27	AC	E3	ED	97	BB	46	05	40	31	E5	37	2C	9E
F.	0A	B1	B5	06	6C	1F	A3	2A	70	FF	BA	07	24	16	C6	61



(a) Round lmor64

(b) Round elmor128

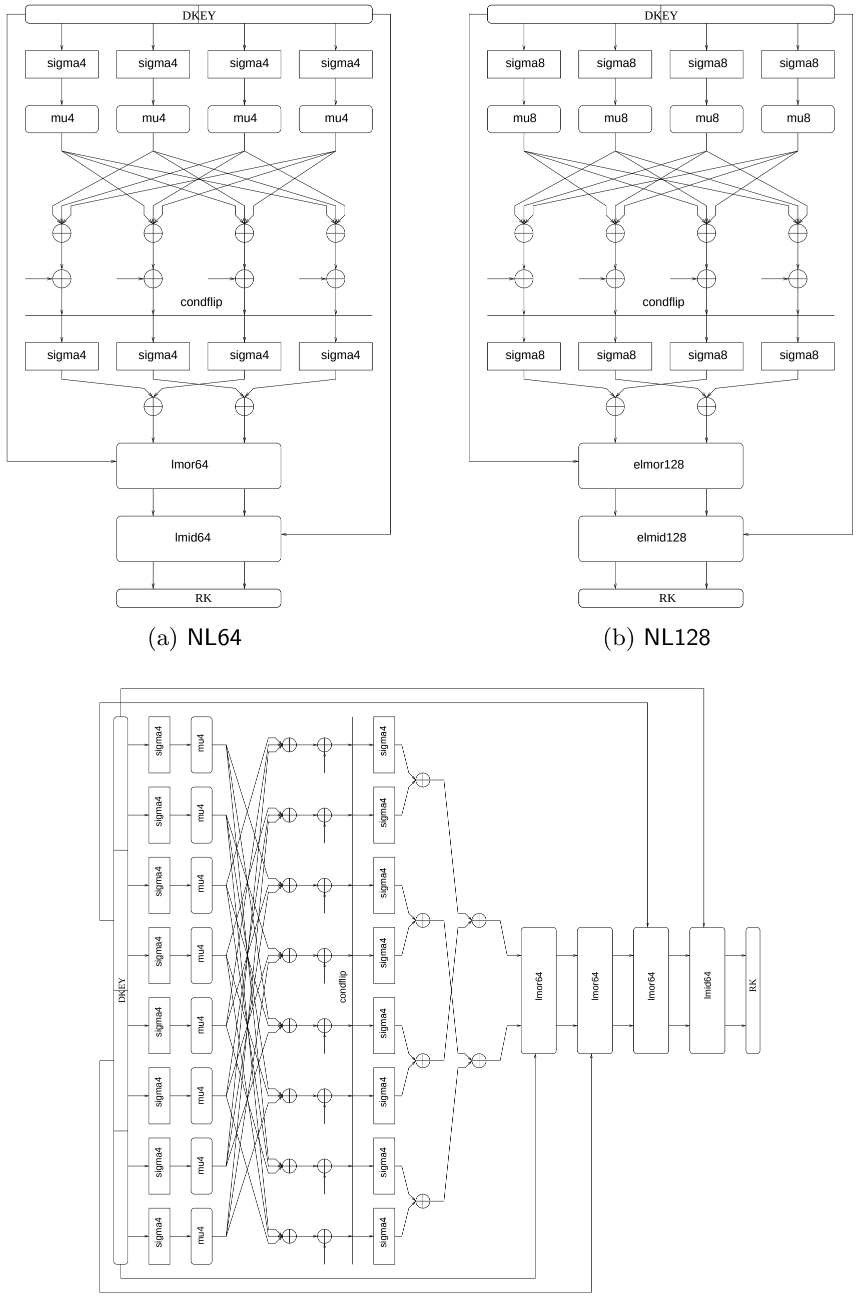
Fig. 1. Round Functions



(a) Round function f32

(b) Round function f64

Fig. 2. Functions f32 and f64



(a) NL64

(b) NL128

(c) NL64h

Fig. 3. NL64, NL64h, and NL128 Functions