ICEBERG : an Involutional Cipher Efficient for Block Encryption in Reconfigurable Hardware.

Francois-Xavier Standaert, Gilles Piret, Gael Rouvroy,
Jean-Jacques Quisquater, Jean-Didier Legat

UCL Crypto Group
Laboratoire de Microelectronique
Universite Catholique de Louvain
Place du Levant, 3, B-1348 Louvain-La-Neuve, Belgium
`standaert,piret,rouvroy,quisquater,legat@dice.ucl.ac.be`

Abstract. We present a fast involutional block cipher optimized for reconfigurable hardware implementations. `ICEBERG` uses 64-bit text blocks and 128-bit keys. All components are involutional and allow very efficient combinations of encryption/decryption. Hardware implementations of `ICEBERG` allow to change the key at every clock cycle without any performance loss and its round keys are derived "on-the-fly" in encryption and decryption modes (no storage of round keys is needed). The resulting design offers better hardware efficiency than other recent 128-key-bit block ciphers. Resistance against side-channel cryptanalysis was also considered as a design criteria for `ICEBERG`.

Keywords: block cipher design, efficient implementations, reconfigurable hardware, side-channel resistance.

1 Introduction

In October 2000, NIST (National Institute of Standards and Technology) selected Rijndael as the new Advanced Encryption Standard. The selection process included performance evaluation on both software and hardware platforms. However, as implementation versatility was a criteria for the selection of the AES, it appeared that Rijndael is not optimal for reconfigurable hardware implementations. Its highly expensive substitution boxes are a typical bottleneck but the combination of encryption and decryption in hardware is probably as critical.

In general, observing the AES candidates [1, 2], one may assess that the criteria selected for their evaluation led to highly conservative designs although the context of certain cryptanalysis may be considered as very unlikely (e.g. more than 2^{100} chosen plaintexts). More recent designs of the NESSIE[1] project (e.g. Khazad [3], Misty [4]) provide an improved efficiency. They also allowed interesting comparisons between Feistel networks (e.g. Misty) and substitution-permutation networks, with respect to hardware efficiency. Although Khazad is

⋆ This work has been funded by the Wallon region (Belgium) through the research project TACTILS $http://www.dice.ucl.ac.be/crypto/TACTILS/T\_home.html$

[1] NESSIE: New European Schemes for Signatures, Integrity, and Encryption. See `http://www.cryptonessie.org`.

not a Feistel network, its structure is designed so that by choosing all components to be involutions, the inverse operation of the cipher differs from the forward operation in the key scheduling only. ICEBERG is also based on an involutional structure but allows to derive the keys more efficiently than Khazad. Moreover, the combination of encryption and decryption is improved due to a simplified diffusion layer.

Reconfigurable hardware devices usually enable high performance encryption/ decryption solutions for real-time applications of multi-Gbps data streams. Video-processing is the typical context where high throughput has to be provided at low hardware cost. Although present encryption algorithms may provide very high encryption rates, it is often at the cost of expensive designs. The main feature of ICEBERG is that is has been defined in order to allow very efficient reconfigurable hardware implementations. An additional criteria was the simplicity of the design. ICEBERG is scalable for different architectures (loop, unrolled, pipeline) and FPGA[2] technologies. As a consequence, ASIC[3] implementations are also efficient. All its components easily fit into 4-input lookup tables and its key scheduling allows to derive the round keys "on-the-fly" in encryption and decryption modes. This involves no storage requirements for the round keys. The resulting design offers better hardware efficiency than other recent 128-key-bit block ciphers. As a consequence, very low-cost hardware crypto-processors and high throughput data encryption are potential applications of ICEBERG.

Finally, resistance against side-channel cryptanalysis was also considered as a design criteria for ICEBERG. Small substitution tables are used in order to allow efficient boolean masking. Moreover, the key agility offers the opportunity to consider new encryption modes where the key is changed frequently in order to make the averaging of side-channel traces unpractical.

This paper is structured as follows. Section 2 presents the design goals of ICEBERG and section 3 gives its specifications. The security analysis of ICEBERG is in section 4 and its performance analysis in section 5. Finally, conclusions are in section 6. Some tables and proofs are given in appendixes.

2 Design goals

Present reconfigurable components like FPGAs are usually made of reconfigurable logic blocks combined with fast access memories (RAM blocks) and high speed arithmetic circuits [5, 6]. Basic logic blocks of FPGAs include a 4-input function generator (called lookup table, LUT), carry logic and a storage element.

As reconfigurable components are divided into logic elements and storage elements, an efficient implementation will be the result of a better compromise between combinatorial logic used, sequential logic used and resulting performances. These observations lead to different definitions of implementation efficiency [7–12]:

[2] FPGA : Field Programmable Gate Array.
[3] ASIC : Application Specific Integrated Circuit.

1. In terms of performances, let the efficiency of a block cipher be the ratio $Throughput\ (Mbits/s)/Area\ (LUTs,\ RAM\ blocks)$.
2. In terms of resources, the efficiency is easily tested by computing the ratio $Nbr\ of\ LUTs/Nbr\ of\ registers$: it should be close to one.

The general design goal of `ICEBERG` is to provide an efficient algorithm for reconfigurable hardware implementations meeting the usual security requirements of block ciphers. More precisely, our design goals were:

1. Good security properties: `ICEBERG` has a resistance against known attacks comparable to recently published block ciphers (AES, NESSIE).
2. Hardware implementation efficiency (as previously defined).
3. Hardware implementation versatility: `ICEBERG` is scalable for different architectures (loop, unrolled, pipeline) and FPGA technologies.
4. Resistance against side-channel cryptanalysis.

3 Specifications

3.1 Block and Key Size

Let n be the block bit-size and k be the key bit-size. The state X is represented as a n-bit vector where $X(i)$ $(0 \leq i < n)$ represents the ith bit from the right. Alternatively, X can be represented as an array of $\frac{n}{4}$ 4-bit blocks, where X_j is the jth block from the right. `ICEBERG` operates on 64-bit blocks and uses a 128-bit key. It is an involutional iterative block cipher based on the repetition of R identical key-dependent round functions.

3.2 The Non-Linear Layer γ

Function γ consists of the successive application of non-linear substitution boxes and bit permutations (i.e. wire crossings).

Substitution layers S_0, S_1: The substitution layers S_j consists of the parallel application of substitution boxes s_j to the blocks of the state.

$$S_j : \mathbb{Z}_{2^4}^{16} \rightarrow \mathbb{Z}_{2^4}^{16} : x \rightarrow y = S_j(x) \Leftrightarrow y_i = s_j(x_i) \quad 0 \leq i \leq 15 \tag{1}$$

Tables of S-boxes s_0, s_1 are given in appendix B.

Bit permutation layer $P8$: The permutation layer $P8$ consists of the parallel application of 8 permutations $p8$ to the state, where $p8$ consists of bit permutations on 8-bit blocks of data. Table of $p8$ is given in appendix B.

$$P8 : \mathbb{Z}_{2^8}^{8} \rightarrow \mathbb{Z}_{2^8}^{8} : x \rightarrow y = P8(x) \Leftrightarrow y(8i+j) = x(8i + p8(j)) \quad 0 \leq i \leq 7,\ 0 \leq j \leq 7 \tag{2}$$

Based on previous descriptions, the non-linear layer γ can be expressed as:

$$\gamma : \mathbb{Z}_2^{64} \rightarrow \mathbb{Z}_2^{64} : \gamma \equiv S_0 \circ P8 \circ S_1 \circ P8 \circ S_0 \tag{3}$$

For cryptanalytic and software implementation purposes, γ may also be viewed as a unique layer consisting of the application of 8 identical 8×8 S-boxes for which the table is given in appendix B.

3.3 The Key Addition Layer $\boldsymbol{\sigma_K}$

The affine key addition σ_K consists of the bitwise exclusive or (XOR, $\oplus$) of a key vector K.

$$\sigma_K : \mathbb{Z}_2^{64} \to \mathbb{Z}_2^{64} : x \to y = \sigma_K(x) \;\Leftrightarrow\; y(i) = x(i) \oplus K(i) \quad 0 \le i \le 63 \tag{4}$$

3.4 The Linear Layer $\boldsymbol{\epsilon_K}$

Function ϵ_K consists of the successive application of binary matrix multiplications and wire crossing layers, combined with the key addition layer for efficiency purposes. We describe it as:

$$\epsilon_K : \mathbb{Z}_2^{64} \to \mathbb{Z}_2^{64} : \epsilon_K \equiv P64 \circ P4 \circ \sigma_K \circ M \circ P64 \tag{5}$$

where M, $P64$, and $P4$ are defined as follows:

Matrix multiplication layer $\boldsymbol{M}$: The matrix multiplication layer M is based on the parallel application of a simple involutional matrix multiplication.

Let $V \in \mathbb{Z}_2^{4\times 4}$ be a binary involutional (i.e. such that $V^2 = I_n$) matrix:

$$V = \begin{bmatrix} 0\,1\,1\,1 \\ 1\,0\,1\,1 \\ 1\,1\,0\,1 \\ 1\,1\,1\,0 \end{bmatrix}$$

M is then defined as:

$$M : \mathbb{Z}_{2^4}^{16} \to \mathbb{Z}_{2^4}^{16} : x \to y = M(x) \;\Leftrightarrow\; y_i = V \cdot x_i \quad 0 \le i \le 15 \tag{6}$$

We define diffusion boxes D as performing multiplication by V. Table of D is given in appendix B.

Bit permutation layer $\boldsymbol{P64}$: Permutation $P64$ performs bit permutations on 64-bit blocks of data.

$$P64 : \mathbb{Z}_2^{64} \to \mathbb{Z}_2^{64} : x \to y = P64(x) \;\Leftrightarrow\; y(i) = x(P64(i)) \quad 0 \le i \le 63 \tag{7}$$

Table of permutation $P64$ is given in appendix B.

Bit permutation layer $\boldsymbol{P4}$: The permutation layer $P4$ consists of the parallel application of 16 permutations $p4$ to the state. $p4$ consists of bit permutations on 4-bit blocks of data. Table of $p4$ is given in appendix B.

$$\begin{aligned} P4 : \mathbb{Z}_{2^4}^{16} \to \mathbb{Z}_{2^4}^{16} : x \to y = P4(x) \;\Leftrightarrow\; y_i(j) = x_i(p4(j)) \\ 0 \le i \le 15,\; 0 \le j \le 3 \end{aligned} \tag{8}$$

The purpose of permutation $P4$ is to efficiently distinguish encryption from decryption. It will become clearer in section 3.8 and appendix A.

3.5 The Round Function ρ_K

Finally, the whole round function can be expressed as:

$$\rho_K : \mathbb{Z}_2^{64} \rightarrow \mathbb{Z}_2^{64} : \rho_K \equiv \epsilon_K \circ \gamma \tag{9}$$

It is illustrated in Figure 1.

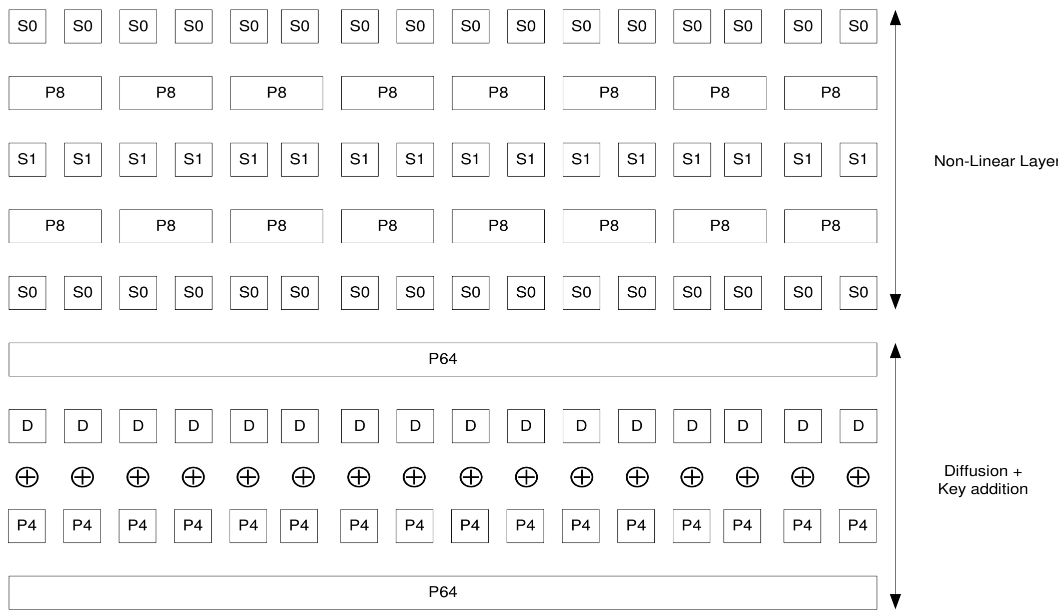

Fig. 1. The round function ρ_K.

3.6 The Key Schedule

The key scheduling process consists of key expansion and key selection.

The key expansion: This process expands the cipher key $K \in \mathbb{Z}_2^{128}$ into a sequence of keys $K^0, K^1, ..., K^R$ also $\in \mathbb{Z}_2^{128}$. We set the initial key $K^0 = K$. Then we expand K^0 by a simple key round function β_C so that:

$$K^{i+1} = \beta_C(K^i) \tag{10}$$

Where $0 \leq i \leq R$ and $C \in \mathbb{Z}_2$ is a round constant discussed in section 3.9.

The **key round** β_C is pictured in Figure 2. It consists of the application of non-linear substitution boxes, shift operations and bit permutations:

$$\beta_C : \mathbb{Z}_2^{128} \rightarrow \mathbb{Z}_2^{128} : \beta_C \equiv \tau_C \circ P128 \circ S' \circ P128 \circ \tau_C \tag{11}$$

where τ_C, S', and $P128$ are defined as follows:

– **Shift layer τ_C:** The shift layer τ_C consists of the application of a variable shift operator to the bytes of the key : shift left if $C = 1$, shift right if $C = 0$.

$$\begin{aligned} \tau_C : \mathbb{Z}_2^{128} \to \mathbb{Z}_2^{128} :\ x \to y = \tau_C(x) \Leftrightarrow \\ \text{if } C = 0 : y(i) = x((i+8) \mod 128) \quad 0 \le i \le 127 \\ \text{if } C = 1 : y(i) = x((i-8) \mod 128) \quad 0 \le i \le 127 \end{aligned} \tag{12}$$

– **Substitution layer S':** The substitution layer S' consists of the parallel application of substitution boxes s_0 to the blocks of the key.

$$S' : \mathbb{Z}_{2^4}^{32} \to \mathbb{Z}_{2^4}^{32} : x \to y = S'(x) \ \Leftrightarrow\ y_i = s_0'(x_i) \quad 0 \le i \le 31 \tag{13}$$

Table of S-box s_0 is given in appendix B.

– **Bit permutations layer $P128$:** $P128$ performs bit permutation on 128-bit blocks of data.

$$\begin{aligned} P128 : \mathbb{Z}_2^{128} \to \mathbb{Z}_2^{128} : x \to y = P128(x) \ \Leftrightarrow\ y(i) = x(P128(i)) \\ 0 \le i \le 127 \end{aligned} \tag{14}$$

Table of $P128$ is given in appendix B.

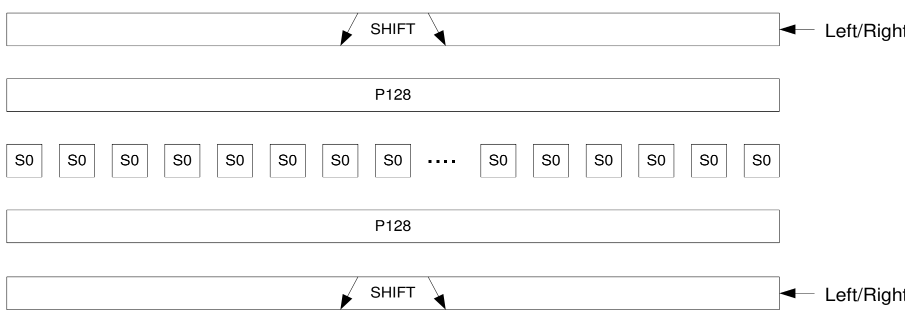

Fig. 2. The key round β_C.

The key selection: From every 128-bit vector K^i, we first apply a simple compression function E that selects 64 bits corresponding to key bytes of K^i having odd indices. We denote the resulting key as $K64^i$. Then we apply a **key selection layer (ϕ)** that consists of the parallel application of a selection function X to the blocks of the key.

$$\begin{aligned} \phi_{sel} : \mathbb{Z}_{2^4}^{16} \to \mathbb{Z}_{2^4}^{16} : K64^i \to RK_{sel}^i = \phi_{sel}(K64^i) \ \Leftrightarrow \\ RK_{sel,j}^i = X_{sel}(K64_j^i) \quad 0 \le j \le 15 \end{aligned} \tag{15}$$

The **selection function** X takes 4-bit inputs and a selection bit sel:

$$\begin{aligned} X_{sel} : \mathbb{Z}_{2^4} \to \mathbb{Z}_{2^4} : x \to y = X_{sel}(x) \Leftrightarrow \\ \begin{cases} y(0) = (x(0) \oplus x(1) \oplus x(2)) \cdot sel \vee (x(0) \oplus x(1)) \cdot \overline{sel} \\ y(1) = (x(1) \oplus x(2)) \cdot sel \vee x(1) \cdot \overline{sel} \\ y(2) = (x(2) \oplus x(3) \oplus x(0)) \cdot sel \vee (x(2) \oplus x(3)) \cdot \overline{sel} \\ y(3) = (x(3) \oplus x(0)) \cdot sel \vee x(3) \cdot \overline{sel} \end{cases} \end{aligned} \tag{16}$$

This selection process is represented in Figure 3. As a result, we obtain a 64-bit round key denoted by RK_1^i if $sel = 1$ and RK_0^i if $sel = 0$.

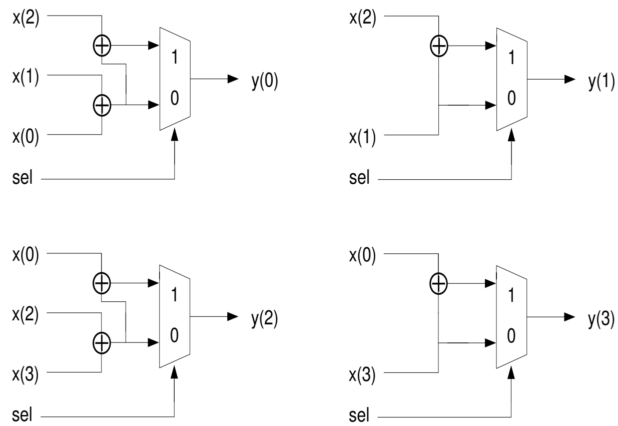

Fig. 3. The key selection function X_{sel}.

3.7 Encryption Process

`ICEBERG` is defined for the cipher key K, as the transformation $\texttt{ICEBERG}[K] = \alpha_R[RK_1^0, RK_1^1,.. .,RK_0^R]$ applied to the plaintext where:

$$\alpha_R[RK_1^0, RK_1^1, ..., RK_0^R] = \sigma_{RK_0^R} \circ \gamma \circ (\bigcirc_{r=1}^{R-1} \rho_{RK_1^r}) \circ \sigma_{RK_1^0} \tag{17}$$

The standard number of rounds is $R = 16$.

3.8 Decryption Process

We now show that `ICEBERG` is an involutional cipher in the sense that the only difference between encryption and decryption is in the key schedule. We will need the following theorem, proven in appendix A:

Theorem 1. For any $K64 \in \mathbb{Z}_{2^4}^{16}$: $\epsilon_{RK_0}^{-1} = \epsilon_{RK_1}$, where $RK_0 = \phi_0(K64)$ and $RK_1 = \phi_1(K64)$.

Then, the decryption process can be obtained as follows. We start from the encryption process:

$$\alpha_R[RK_1^0, RK_1^1, ..., RK_0^R] = \sigma_{RK_0^R} \circ \gamma \circ (\bigcirc_{r=1}^{R-1} \epsilon_{RK_1^r} \circ \gamma) \circ \sigma_{RK_1^0}$$

Then we have for decryption:

$$\alpha_R^{-1}[RK_1^0, RK_1^1, ..., RK_0^R] = \sigma_{RK_1^0} \circ (\bigcirc_{r=R-1}^{1} \gamma \circ \epsilon_{RK_1^r}^{-1}) \circ \gamma \circ \sigma_{RK_0^R}$$

$$\Leftrightarrow \alpha_R^{-1}[RK_1^0, RK_1^1, ..., RK_0^R] = \sigma_{RK_1^0} \circ \gamma \circ (\bigcirc_{r=R-1}^{1} \epsilon_{RK_1^r}^{-1} \circ \gamma) \circ \sigma_{RK_0^R} \tag{18}$$

Finally the above theorem leads to:

$$\alpha_R^{-1}[RK_1^0, RK_0^1, ..., RK_0^R] = \sigma_{RK_1^0} \circ \gamma \circ (\bigcirc_{r=R-1}^{1} \epsilon_{RK_0^r} \circ \gamma) \circ \sigma_{RK_0^R}$$

3.9 Round Constants

`ICEBERG` is an involutional cipher in the sense that the only difference between encryption and decryption is in the key schedule. Moreover, if properly chosen, the round constants allow to compute keys "on-the-fly" in encryption and decryption modes. Basically, we would like round keys to satisfy:

$$\begin{aligned} K^0 &= K^R \\ K^1 &= K^{R-1} \\ K^2 &= K^{R-2} \\ &\ldots \end{aligned} \tag{19}$$

This involves that R is even. Then, if the first half of round constants (i.e. until round 8) is 0 (shift left) and the second half is 1 (shift right), the resulting round keys will satisfy Equation (19).

As a consequence, the only difference between encryption and decryption is the selection function ϕ of the key bits, as $\epsilon_{RK_1}^{-1} \equiv \epsilon_{RK_0}$.

4 Security Analysis

4.1 Design Properties of the Components

S-Boxes: The non-linear layer γ may be viewed as made out of the parallel application of 8 copies of the same 8×8 S-box. We designed this S-box such that it has the following properties:

- It is an involution.
- Its δ-parameter[4] is 2^{-5}.
- Its λ-parameter[5] is 2^{-2}.
- Its nonlinear order ν is maximum, namely 7.

For efficiency purposes, the s-box was generated from a fixed permutation $p8$ and small 4×4 s-boxes s_0 and s_1 that perfectly fits into 4-input LUTs. The generation of the s-boxes is detailed in Appendix C.

The bit permutations: $P64$ and $P128$ were designed such as to disturb as much as possible the bit alignment inside bytes, in order to provide resistance against some attacks. A remarkable property of $P64$ and $P128$ is that 2 bits from the same byte are always mapped to 2 bits belonging to different bytes. $p8$ is involutional and allows to generate good substitution boxes. Finally, $p4$ allows the selection function X_{sel} to be implemented in one LUT.

[4] δ equals the probability of the best differential approximation.

[5] We define the *bias* of a linear approximation that holds with probability p as $\epsilon = |p - 1/2|$. The λ-*parameter* of a S-box is equal to 2 times the bias of its best linear approximation.

The Diffusion Layer: Due to the fact that we attached much importance to hardware implementation aspects in the design of the diffusion layer, it is not optimal. More precisely, it is easy to see that its byte branch number is 4, as the bit branch number of $p4 \circ \sigma_K \circ D$ is 4, and because of the remarkable property of $P64$ we have just mentioned. The diffusion boxes D were designed so that their combination with the key addition layer σ_K can be done inside one LUT.

The key round: The key round has been chosen for its efficiency properties, as well as in order to provide resistance against key schedule cryptanalysis and slide attacks:

1. Non periodicity is provided by the shift operation τ_C.
2. Non linearity is provided by non-linear S-boxes.
3. Good diffusion properties are provided by the combination of shifts, S-boxes and bit permutations.

Moreover, the shift layer τ_C is used in order to allow the property (19) to be respected. The selection function X_{sel} is necessary to prove the property of Appendix A and is designed such that it fits into a single LUT.

4.2 Strength Against Known Attacks

Linear and Differential Cryptanalysis: From the properties of the S-box and the diffusion layer, we can compute that a differential characteristic [13] over 2 rounds of `ICEBERG` has probability at most $(2^{-5})^4 = 2^{-20}$. Also, a linear characteristic [14] over these 2 rounds has input-output correlation at most $(2^{-2})^4 = 2^{-8}$. Therefore loose bounds can be computed for the full cipher (16 rounds):

- The probability of the best differential characteristic is smaller than 2^{-160}.
- The input-output correlation of the best linear characteristic is smaller than 2^{-64}.

The security margin is very likely big enough to prevent variants of differential and linear attacks, such as boomerang [15] and rectangle [16] attacks, multiple linear cryptanalysis [17], non-linear approximations of outer rounds [18],... Note also that the security margin of `ICEBERG` against linear and differential cryptanalysis is comparable to the one of Khazad. This is probably more than it is necessary, as resistance against structural attacks [19, 20] was probably more determinant in the choice of the number of rounds of Khazad (8), than security margins against linear and differential cryptanalysis.

Truncated and Impossible Differentials: Truncated differentials were introduced in [21], and impossible differentials in [22, 23]. They typically apply to ciphers operating on well-aligned data blocks (often bytes), such as Khazad or the AES (and many others). However our cipher does not enter in this category because of the $P64$ layer, which makes it very difficult to attack this way. Therefore such an attack on the full 16-round cipher seems very unlikely.

Square Attacks: Like truncated differential attacks, square attacks [19] generally apply to ciphers operating on well-aligned data blocks. Therefore the $P64$ layer should prevent them efficiently, at least on more than a few rounds. More precisely, consider a batch of $256^m (m \in \{1...7\})$ plaintexts, such that $8 - m$ bytes remain constant for all of them (these bytes are said *passive*), while the concatenation of the m other bytes takes every possible value (it is said *active*). This property is preserved by the γ layer. On the contrary, the $P64$ layer makes all bytes garbled (i.e. not active nor passive), which prevents pushing a basic "square characteristic" further. The same type of argument could be applied to truncated differential attacks.

Interpolation Attacks: Interpolation attacks [24] are made possible when the S-box has a simple algebraic structure, allowing to express the cipher as a sufficiently simple polynomial or rational expression. The diffusion layer also has a role with this respect. As the S-box of `ICEBERG` has no simple algebraic expression, it prevents interpolation attacks for more than a few rounds of our cipher.

Higher Order Differential Cryptanalysis: It was introduced by Knudsen in [21], and relies on finding high order differentials being a constant for the whole cipher. But as the nonlinear order of the S-box we use is maximal, namely 7, we can expect that the maximal value of 63 for the non-linear order of the cipher is reached after a few rounds of `ICEBERG`.

Slide Attacks: Slide attacks [25, 26] work against ciphers using a periodic key schedule. Although the sequence of subkeys produced by the key schedule of `ICEBERG` is not periodic, it has a particular structure, namely:

$$(K^0, K^1, ..., K^7, K^8, K^7, ..., K^0)$$

The key schedule of the GOST cipher has some similarities with the one of `ICEBERG`. Vulnerability of some variants and reduced-round versions of GOST against slide attacks is examined in [26]. However none of the attacks presented there seems to be applicable to our cipher.

Related-Key Attacks: The first related-key attack has been described in [27], and is the related-key counterpart of the slide attack. Let us examine a slightly simplified version of `ICEBERG`, where the initial key addition $\sigma_{RK_1^0}$ is replaced by a normal round $\rho_{RK_1^0}$, and the final $\sigma_{RK_0^R} \circ \gamma$ is also replaced by a normal round $\rho_{RK_0^R}$. Then if 2 keys K and K^* are such that $K^1 = K^{*0}$, and 2 plaintexts P and P^* are such that $P^* = \rho_{RK_1^0}(P)$, encryption of P under K and of P^* under K^* will process the same way (with a difference of 1 round) during 8 rounds. However then round keys, and hence computation, will differ; therefore such a related key attack does not work against our key schedule. Forgetting the simplification we made on the first and last round of `ICEBERG`, a related-key attack becomes even more difficult. Differential related-key attacks [28] are also very unlikely to be applicable to `ICEBERG`, due to the good diffusion and nonlinearity of its key schedule.

Weak keys: The design properties of the key round prevent `ICEBERG` from having weak keys. The only remarkable property of the key round is in the selection function X_{sel} where some symbols are independent of the selection bit. Namely, hexadecimal input symbols $0, 2, 8, A$ become $0, C, 3, F$ regardless of $sel = 0$ or $sel = 1$. However, this point is very unlikely to be an exploitable weakness.

Biryukov's observations on Involutional Ciphers: Observations of Biryukov on Khazad and Anubis [29] remain valid for `ICEBERG`. However this study could at best threaten 5 rounds of our cipher, while it is made out of 16 rounds.

Side-channel cryptanalysis: Although cryptosystem designers frequently assume that secret parameters will be manipulated in closed reliable computing environments, Kocher et *al.* stressed in 1998 [30] that actual computers and microchips leak information correlated to the data handled. Side-channel attacks based on time, power and electromagnetic measurements were successfully applied to smart card implementations of block ciphers. Protecting implementations against side-channel attacks is usually difficult and expensive. Masking all the data with random boolean values is suggested in several papers [31, 32] and the use of small substitution tables allows to implement this efficiently, although it is still an expensive solution.

The key agility provided by `ICEBERG` (changing the key at every plaintext block is for free) also offers interesting opportunities to prevent most side-channel attacks by defining new encryption modes where the key is changed sufficiently often. As most side-channel attacks need to collect several leakage traces to remove the noise from useful information, changing the key frequently, even in a well chosen deterministic way (e.g. LFSR-based), would make most attacks somewhat unpractical. Actually, only template attacks [33] allow to extract information from a single sample but the context is also more specific as they require that an adversary has access to an experimental device (identical to the device attacked) that he can program to his choosing.

5 Performance analysis

`ICEBERG` has been designed in order to allow very efficient reconfigurable hardware implementations, as defined in section 2. For this purpose, we applied the following design rules:

1. All components easily fit into 4-input LUTs. Practically, `ICEBERG` is made of the parallel application of 4-input-bit transforms combined with bit permutations or shifts.
2. All components are involutional so that encryption and decryption can be made with the same hardware. The only difference between encryption and decryption is in the selection bit of ϕ_{sel}.
3. The key expansion allows to derive round keys "on-the-fly" in encryption and decryption modes. There is no need to store the round keys and the key can be changed in one clock cycle.

4. The scheduling of the algorithm is balanced so that the round and key round can be made in the same number of clock cycles.
5. The non-linear layer can be efficiently implemented into the RAM blocks available in most modern FPGAs.

5.1 Hardware implementations

As all components easily fit into 4-input LUTs, we can directly evaluate the hardware cost of `ICEBERG`:

Component	HW cost (LUTs)	Component	HW cost (LUTs)
S_0, S_1	64	τ_C	128
γ	192	S'	128
ϵ_K	64	Key round β_C	384
Round ρ_K	256	ϕ_{sel}	64
Complete round + key round			704

As a comparison, Khazad needs 576 LUTs for its round and 768 LUTs for its key round [11], with a more expensive encryption/decryption structure. AES Rijndael is even more critical as its round needs 2608 LUTs and its key round 768 LUTs [12]. Although comparisons between hardware implementations are made difficult by their high dependency on the design methodology, we may reasonably expect to have `ICEBERG` encryption/decryption for the cost of Khazad encryption only and half the cost of Rijndael encryption, with comparable encryption rates.

Moreover, the parallel nature of `ICEBERG` allows to implement every possible throughput/area tradeoff as well as very efficient pipeline. The maximum pipeline can be obtained by inserting registers after every LUT which allows to reach an optimal ratio $Nbr\ of\ LUTs/Nbr\ of\ registers = 1$. Loop and unrolled architectures are easily implementable with LUT-based or RAM-based substitution boxes. As a consequence, `ICEBERG` offers various and efficient implementation opportunities. Its regular structure makes the classical design optimizations easily reachable. Software efficiency is not a design goal of `ICEBERG`. It is briefly discussed in Appendix D.

6 Conclusion

This paper presented the platform-specific encryption algorithm `ICEBERG` and the rationale behind its design. `ICEBERG` is based on a fast involutional structure in order to provide very efficient hardware implementation opportunities. We showed the specificity of this type of platform in block cipher design. We also underlined that the overall structure of a cipher is important for efficiency purposes (for example, in designing rounds and key rounds that can be made in the same number of clock cycles, or in allowing "on the fly" key derivation in both encryption and decryption modes). We believe `ICEBERG` to be as secure as AES and NESSIE candidates and much more efficient for reconfigurable hardware implementations. `ICEBERG` also offers free opportunities to defeat most side-channel attacks by using adequate encryption modes.

Acknowledgements

The authors are grateful to Paulo Barreto for providing valuable comments and help during the design of `ICEBERG`.

References

1. NIST Home page, http://csrc.nist.gov/CryptoToolkit/aes/.
2. J.Daemen, V.Rijmen, *The Block Cipher Rijndael*, Smart Card Research and Applications, pp 288-296, Springer-Verlag, LNCS 1820, 2000.
3. P.Barreto, V.Rijmen, *The KHAZAD Legacy-Level Block Cipher*, Submission to NESSIE project, available from http://www.cosic.esat.kuleuven.ac.be/nessie/
4. M.Matsui, *Supporting Document of MISTY1*, , Submission to NESSIE project, available from http://www.cosic.esat.kuleuven.ac.be/nessie/
5. Xilinx: *Virtex 2 FPGAs Data Sheet*, http://www.xilinx.com.
6. Altera: *Stratix 1.5V FPGAs Data Sheet*, http://www.altera.com.
7. M.McLoone and J.V.McCanny, *High Performance Single Ship FPGA Rijndael Algorithm Implementations*, in the proceedings of CHES 2001: The Third International CHES Workshop, Lecture Notes In Computer Science, LNCS2162, pp 65-76, Springer-Verlag.
8. V.Fischer and M.Drutarovsky, *Two Methods of Rijndael Implementation in Reconfigurable Hardware*, in the proceedings of CHES 2001: The Third International CHES Workshop, Lecture Notes In Computer Science, LNCS2162, pp 65-76, Springer-Verlag.
9. A.Satoh et al, *A Compact Rijndael Hardware Architecture with S-Box Optimization*, Advances in Cryptology - ASIACRYPT 2001, LNCS 2248, pp239-254, Springer-Verlag.
10. Helion Technology, *High Performance AES (Rijndael) Cores for XILINX FPGA*, *http : //www.heliontech.com*.
11. F.X.Standaert,G.Rouvroy,J.J.Quisquater,J.D.Legat, *Efficient FPGA Implementations of Block Ciphers KHAZAD and MISTY1*, in the proceedings of the Third NESSIE Workshop, November 6-7 2002, Munich, Germany.
12. F.X.Standaert,G.Rouvroy,J.J.Quisquater,J.D.Legat, *A Methodology to Implement Block Ciphers in Reconfigurable Hardware and its Application to Fast and Compact AES Rijndael*, in the proceedings of FPGA 2003: the Field Programmable Logic Array Conference, February 23-25 2003, Monterey, California.
13. E.Biham, A.Shamir, *Differential cryptanalysis of DES-like cryptosystems (Extended abstract)*, Proceedings of Crypto 90, pp 2-21, Springer-Verlag, LNCS 537, 1990.
14. M.Matsui, *Linear cryptanalysis method for DES cipher*, Proceedings of EuroCrypt 93, pp 386-397, Springer-Verlag, LNCS 765, 1993.
15. D.Wagner, *The Boomerang Attack*, Proceedings of FSE 99, pp 156-170, Springer-Verlag, LNCS 1636, 1999.
16. E.Biham, O.Dunkelman, N.Keller, *The rectangle Attack - Rectangling the Serpent*, Proceedings of Eurocrypt 2001, pp 340-357, Springer-Verlag, LNCS 2045, 2001.
17. B.S.Kaliski, M.J.B.Robshaw, *Linear Cryptanalysis using Multiple Approximations*, Proceedings of Crypto 94, pp.26-39, Springer-Verlag, LNCS 0839, 1994.
18. L.Knudsen, M.J.B.Robshaw, *Non-Linear Approximations in Linear Cryptanalysis*, Proceedings of Eurocrypt 96, pp 224-236, Springer-Verlag, LNCS 1070, 1996.

19. J.Daemen, L.Knudsen, V.Rijmen, *The Block Cipher SQUARE*, Proceedings of FSE 1997, pp 149-165, Springer-Verlag, LNCS 1267, 1999.
20. N.Ferguson, J.Kelsey, S.Lucks, and al., *Improved Cryptanalysis of Rijndael*, Proceedings of FSE 2000, pp 213-230, Springer-Verlag, LNCS 1978, 2000.
21. L.Knudsen, *Truncated and Higher Order Differentials*, Proceedings of FSE 94, pp 196-211, Springer-Verlag, LNCS 1008, 1995.
22. E.Biham, A.Biryukov and A.Shamir, *Cryptanalysis of Skipjack Reduced to 31 Rounds Using Impossible Differentials*, Proceedings of Eurocrypt 99, pp 12-23, Springer-Verlag, LNCS 1592, 1999.
23. E.Biham, A.Biryukov and A.Shamir, *Miss in the Middle Attacks on IDEA, Khufu, and Khafre*, Proceedings of FSE 99, pp 124-138, Springer-Verlag, LNCS 1636, 1999.
24. T.Jakobsen and L.Knudsen, *The Interpolation Attack on Block Ciphers*, Proceedings of FSE 97, pp 28-40, Springer-Verlag, LNCS 1267, 1997.
25. A.Biryukov, D.Wagner, *Slide Attacks*, Proceedings of FSE'99, pp 245-259, Springer Verlag, LNCS 1636, 1999.
26. A.Biryukov, D.Wagner, *Advanced Slide Attacks*, Proceedings of Eurocrypt 00, pp 589-606, Springer Verlag, LNCS 1807, 2000.
27. E.Biham, *New Type of Cryptanalytic Attacks Using Related Key*, Proceedings of Eurocrypt 93, pp 229-246, Springer-Verlag, LNCS 765, 1994.
28. J.Kelsey, B.Schneier, D.Wagner, *Related-Key Cryptanalysis of 3-WAY, Biham-DES, CAST, DES-X, NewDES, RC2, and TEA*, Proceedings of AusCrypt'92, pp 196-208, Springer-Verlag, LNCS 718, 1993.
29. A.Biryukov, *Analysis of Involutional Ciphers: Khazad and Anubis*, Proceedings of FSE 2003, Springer-Verlag, to appear.
30. P.Kocher, J.Jaffe, B.Jun, *Differential Power Analysis*, in the proceedings of CRYPTO 99, Lecture Notes in Computer Science 1666, pp 398-412, Springer-Verlag.
31. L.Goubin,J.Patarin, *DES and Differential Power Analysis: The Duplication Method*, in the proceedings of CHES 1999, Lecture Notes in Computer Science 1717, pp 158-172, Springer-Verlag.
32. S.Chari et *al.*, *Towards Sound Approaches to Counteract Power-Analysis Attacks*, in the proceedings of CRYPTO 1999, Lecture Notes in Computer Science 1666, pp 398-412, Springer-Verlag.
33. S.Chari, J.Rao, P.Rohatgi, *Template Attacks*, in the proceedings of CHES 2002, Lecture Notes in Computer Science 2523, pp 13-28, Springer-Verlag.
34. A.Pfitzmann,R.Aβmann, *More Efficient Software Implementations of (Generalized) DES*, Institut fur Rechnerent und Fehlertoleranz, Univ.Karlsruhe, Interner Bericht 18/90.
35. E.Biham, *A Fast New DES Implementation in Software*, Technion - Computer Science Department, Technical Report CS0891 - 1997.
36. A.M.Youssef, S.E.Tavares, H.Heys, *A New Class of Substitution-Permutation Networks*, Proceedings of Selected Areas in Cryptography (SAC 96), pp 132-147, 1996.
37. H.M. Heys, S.E. Tavares, *Known Plaintext Cryptanalysis of Tree-Structured Block Ciphers*, Electronics Letters, Vol. 31, pp 784-785, May 1995.
38. L.Knudsen, *Block Ciphers - Analysis, Design and Applications*, Doctoral Dissertation, DAIMI PB 485, Aarhus University, Denmark, 1994.
39. J.Daemen, *Cipher and Hash Function Design*, Doctoral Dissertation, March 1995, KULeuven.
40. V.Rijmen, *Cryptanalysis and Design of Iterated Block Ciphers*, Doctoral Dissertation, October 1997, KULeuven.

A Proof of theorem 1.

We have to prove that $P4 \circ \sigma_{RK_1} \circ M \equiv M \circ \sigma_{RK_0} \circ P4$. Inputs and outputs of every transform are represented in Figure 4. We simply write down relations between them. First we encrypt with RK_1:

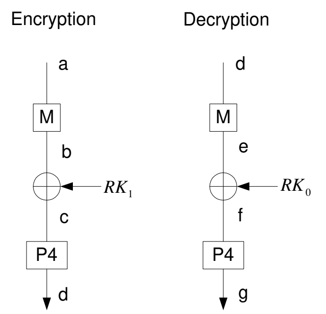

Fig. 4. Theorem 1.

$$\begin{aligned}
b_0 &= a_1 \oplus a_2 \oplus a_3 \\
b_1 &= a_0 \oplus a_2 \oplus a_3 \\
b_2 &= a_0 \oplus a_1 \oplus a_3 \\
b_3 &= a_0 \oplus a_1 \oplus a_2
\end{aligned}$$

$$\begin{aligned}
c_0 &= a_1 \oplus a_2 \oplus a_3 \oplus k_0 \oplus k_1 \oplus k_2 \\
c_1 &= a_0 \oplus a_2 \oplus a_3 \oplus k_1 \oplus k_2 \\
c_2 &= a_0 \oplus a_1 \oplus a_3 \oplus k_2 \oplus k_3 \oplus k_0 \\
c_3 &= a_0 \oplus a_1 \oplus a_2 \oplus k_3 \oplus k_0
\end{aligned}$$

$$\begin{aligned}
d_0 &= a_0 \oplus a_2 \oplus a_3 \oplus k_1 \oplus k_2 \\
d_1 &= a_1 \oplus a_2 \oplus a_3 \oplus k_0 \oplus k_1 \oplus k_2 \\
d_2 &= a_0 \oplus a_1 \oplus a_2 \oplus k_3 \oplus k_0 \\
d_3 &= a_0 \oplus a_1 \oplus a_3 \oplus k_2 \oplus k_3 \oplus k_0
\end{aligned}$$

Then, when we decrypt with RK_0:

$$\begin{aligned}
e_0 &= d_1 \oplus d_2 \oplus d_3 = a_1 \oplus k_0 \oplus k_1 \\
e_1 &= d_0 \oplus d_2 \oplus d_3 = a_0 \oplus k_1 \\
e_2 &= d_0 \oplus d_1 \oplus d_3 = a_3 \oplus k_2 \oplus k_3 \\
e_3 &= d_0 \oplus d_1 \oplus d_2 = a_2 \oplus k_3
\end{aligned}$$

From (16), we have:

$$\begin{aligned}
f_0 &= a_1 \\
f_1 &= a_0 \\
f_2 &= a_3 \\
f_3 &= a_2
\end{aligned}$$

And finally, permutation $P4$ finishes the decryption:

$$\begin{aligned}
g_0 &= a_0 \\
g_1 &= a_1 \\
g_2 &= a_2 \\
g_3 &= a_3
\end{aligned}$$

Remark that permutation $P4$ allows the selection function X_{sel} to be efficiently implemented in LUTs as it has at most 4 inputs: 3 key bits and a selection bit.

B Tables.

0	1	2	3
1	0	3	2

Table 1. p4.

0	1	2	3	4	5	6	7
0	1	4	5	2	3	6	7

Table 2. p8.

0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
d	7	3	2	9	a	c	1	f	4	5	e	6	0	b	8

Table 3. s_0.

0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
4	a	f	c	0	d	9	b	e	6	1	7	3	5	8	2

Table 4. s_1.

0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	e	d	3	b	5	6	8	7	9	a	4	c	2	1	f

Table 5. D.

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	24	c1	38	30	e7	57	df	20	3e	99	1a	34	ca	d6	52	fd
10	40	6c	d3	3d	4a	59	f8	77	fb	61	0a	56	b9	d2	fc	f1
20	07	f5	93	cd	00	b6	62	a7	63	fe	44	bd	5f	92	6b	68
30	03	4e	a2	97	0b	60	83	a3	02	e5	45	67	f4	13	08	8b
40	10	ce	be	b4	2a	3a	96	84	c8	9f	14	c0	c4	6f	31	d9
50	ab	ae	0e	64	7c	da	1b	05	a8	15	a5	90	94	85	71	2c
60	35	19	26	28	53	e2	7f	3b	2f	a9	cc	2e	11	76	ed	4d
70	87	5e	c2	c7	80	b0	6d	17	b2	ff	e4	b7	54	9d	b8	66
80	74	9c	db	36	47	5d	de	70	d5	91	aa	3f	c9	d8	f3	f2
90	5b	89	2d	22	5c	e1	46	33	e6	09	bc	e8	81	7d	e9	49
a0	e0	b1	32	37	ea	5a	f6	27	58	69	8a	50	ba	dd	51	f9
b0	75	a1	78	d0	43	f7	25	7b	7e	1c	ac	d4	9a	2b	42	e3
c0	4b	01	72	d7	4c	fa	eb	73	48	8c	0c	f0	6a	23	41	ec
d0	b3	ef	1d	12	bb	88	0d	c3	8d	4f	55	82	ee	ad	86	06
e0	a0	95	65	bf	7a	39	98	04	9b	9e	a4	c6	cf	6e	dc	d1
f0	cb	1f	8f	8e	3c	21	a6	b5	16	af	c5	18	1e	0f	29	79

Table 6. 8 x 8 substitution box.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	12	23	25	38	42	53	59	22	9	26	32	1	47	51	61
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
24	37	18	41	55	58	8	2	16	3	10	27	33	46	48	62
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
11	28	60	49	36	17	4	43	50	19	5	39	56	45	29	13
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
30	35	40	14	57	6	54	20	44	52	21	7	34	15	31	63

Table 7. P64.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
76	110	83	127	67	114	92	97	98	65	121	106	78	112	91	82
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
71	101	89	126	72	107	81	118	90	124	73	88	64	104	100	85
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
109	87	75	113	120	66	103	115	122	108	95	69	74	116	80	102
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
84	96	125	68	93	105	119	79	123	86	70	117	111	77	99	94
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
28	9	37	4	51	43	58	16	20	26	44	34	0	61	12	55
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
46	22	15	2	48	31	57	33	27	18	24	14	6	52	63	42
96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
49	7	8	62	30	17	47	38	29	53	11	21	41	32	1	60
112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
13	35	5	39	45	59	23	54	36	10	40	56	25	50	19	3

Table 8. P128.

C Generation of the ICEBERG S-box [3]

As $P8$ is fixed, the only part of the ICEBERG S-box structure still unspecified consists of the s_0 and s_1 involutions, which are generated pseudo-randomly in a verifiable way.

The searching algorithm starts with two copies of a simple involution without fixed points (namely, the negation mapping $u \mapsto \bar{u} = u \oplus \texttt{0xF}$), and pseudo-randomly derives from each of them a sequence of 4×4 substitution boxes ("mini-boxes") with the optimal values $\delta = 1/4$, $\lambda = 1/2$, and $\nu = 3$. At each step, in alternation, only one of the sequences is extended with a new mini-box. The most recently generated mini-box from each sequence is taken, and the pair is combined according to the ICEBERG S-box shuffle structure; finally, the resulting 8×8 S-box, if free of fixed points, is tested for the design criteria regarding δ, λ, and ν.

Given a mini-box at any point during the search, a new one is derived from it by choosing two pairs of mutually inverse values and swapping them, keeping the result an involution without fixed points; this is repeated until the running mini-box has optimal values of δ, λ, and ν.

The pseudo-random number generator is implemented using the AES cipher Rijndael in counter mode, with a fixed key consisting of 128 zero bits and an initial counter value consisting of 128 zero bits.

The following pseudo-code fragment illustrates the computation of the chains of mini-boxes and the resulting S-box:

procedure ShuffleStructure(s_0, s_1)
  for $w \leftarrow 0$ **to** 255 **do**
    $u_0 \leftarrow s_0[w \gg 4]; \quad v_0 \leftarrow s_0[w \,\&\, \texttt{0x0F}];$
    $u_1 \leftarrow (u_0 \,\&\, \texttt{0xC}) \mid ((v_0 \,\&\, \texttt{0xC}) \gg 2); \quad v_1 \leftarrow (v_0 \,\&\, \texttt{0x3}) \mid ((u_0 \,\&\, \texttt{0x3}) \ll 2);$
    $u_0 \leftarrow s_1[u_1]; \quad v_0 \leftarrow s_1[v_1];$
    $u_1 \leftarrow (u_0 \,\&\, \texttt{0xC}) \mid ((v_0 \,\&\, \texttt{0xC}) \gg 2); \quad v_1 \leftarrow (v_0 \,\&\, \texttt{0x3}) \mid ((u_0 \,\&\, \texttt{0x3}) \ll 2);$
    $S[w] \leftarrow (s_0[u_1] \ll 4) \mid s_0[v_1];$
  end for
  return S;
end procedure

procedure SearchRandomSBox()
  // initialize mini-boxes to the negation involution:
  for $u \leftarrow 0$ **to** 255 **do**
    $s_0[u] \leftarrow \bar{u};\ s_1[u] \leftarrow \bar{u};$
  end for
  // look for S-box conforming to the design criteria:
  repeat
    // swap mini-boxes (update the "older" one only)
    $s_0 \leftrightarrow s_1;$
    // randomly generate a "good" $GF(2^4)$ involution free of fixed points:

repeat
  repeat
    // randomly select x and y such that
    // $x \neq y$ and $s_1[x] \neq y$ (this implies $s_1[y] \neq x$):
    $z \leftarrow$ RandomByte(); $x \leftarrow z \gg 4$; $y \leftarrow z \,\&\, \texttt{0x0F}$;
  until $x \neq y \;\wedge\; s_1[x] \neq y$;
  // swap entries:
  $u \leftarrow s_1[x]$; $v \leftarrow s_1[y]$;
  $s_1[x] \leftarrow v$; $s_1[u] \leftarrow y$;
  $s_1[y] \leftarrow u$; $s_1[v] \leftarrow x$;
until $\delta(s_1) = 1/4 \;\wedge\; \lambda(s_1) = 1/2 \;\wedge\; \nu(s_1) = 3$;
// build S-box from the mini-boxes:
$S \leftarrow$ ShuffleStructure(s_0, s_1);
// test the design criteria:
until #FixedPoints(S) $= 0 \;\vee\; \delta(S) \leqslant 2^{-5} \;\wedge\; \lambda(S) \leqslant 2^{-2} \;\wedge\; \nu(S) = 7$;
return S;
end procedure

D Software implementations

Software efficiency is not a design goal of `ICEBERG`. Nevertheless, its round function may be implemented using a table lookup approach as it is suggested in [34] and is therefore comparable to the one of Khazad. The key expansion of `ICEBERG` is actually its most critical part as a lookup table implementation requires separate tables for transforms $\tau_C \circ P128$ and transforms $S' \circ P128 \circ \tau_C$. Note that `ICEBERG` is also susceptible to be implemented in bitslice mode as suggested in [35].

If a software-efficient key schedule is wanted, an alternative key round based on a small Feistel structure can be used, illustrated in Figure 5. We just use a

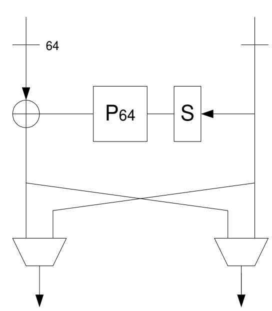

Fig. 5. A modified ρ_K.

conditional switch of the two 64-bit vectors so that we can encrypt during half the rounds and decrypt afterwards in order to satisfy Equation (19). This will only slightly affect hardware performances (an additional multiplexor is necessary to select the round keys).