The **SACI** Special-Purpose Block Cipher

Paulo Sérgio Licciardi Messeder Barreto
Gustavo Yamasaki Martins Vieira
Wilson Vicente Ruggiero
{pbarreto,gymv,wilson}@larc.usp.br

Laboratório de Arquitetura e Redes de Computadores (LARC)
Departamento de Engenharia de Computação e Sistemas Digitais (PCS)
Escola Politécnica, EP
Universidade de São Paulo, USP, Brazil

Abstract

Saci is a special-purpose iterated block cipher, targeted at certain applications with heavily constrained resources and very low traffic of encrypted messages. A typical scenario is the synchronization of tokens, an infrequent operation involving the exchange of just a few encrypted bytes under a fixed or seldom updated key. Saci is an instance of the Wide Trail family of algorithms which includes the AES cipher, and displays both *involutional structure*, in the sense that the encryption and decryption modes differ only in the key schedule, and *cyclic key schedule*, whereby the round subkeys can be computed in-place in any order.

Resumo

Saci é uma cifra de bloco iterativa de propósito especial projetada para aplicações com recursos muito escassos e com pouco tráfego de mensagens cifradas. Um cenário típico é a sincronização de tokens, uma operação esporádica envolvendo a troca de apenas alguns bytes cifrados por uma chave raramente atualizada. Saci pertence à família de algoritmos de trilha larga que incluem a cifra AES e apresentam *estrutura involutiva*, no sentido de que tanto a cifração quanto a decifração diferem apenas no escalonamento de chave, e *escalonamento cíclico de chaves*, segundo o qual as subchaves podem ser calculadas diretamente em qualquer ordem.

1 Introduction

Consider the following scenario. An online system uses a symmetric-based synchronous identification protocol (e.g. a challenge-response based on a shared secret key and a timestamp) to control the access of legitimate users. On the user side the protocol runs on a small hardware token with severely constrained storage and processing capabilities. The timestamp plays the role of a "challenge"; properly encrypting it grants access to the system. To avoid bandwidth consumption, the timestamp must be locally maintained by the token, and can eventually albeit seldom run out of synchronization with the server beyond the level of straightforward compensation (e.g. if the server clock is adjusted to daylight savings, or if the user moves to a different time zone served by a distinct central).

In such circumstances user input may be asked as the response to a challenge. However, practical considerations dictate that this input be rather short, usually less than the length of a telephone number; a typical size is 24 bits. Yet it must be encrypted with a special key used only for this purpose, and transmitted over the communications channel *without increasing the response size*: the ciphertext must fit 24 bits as well.

If the cipher key is fixed or rarely changed and it is not possible to prepend an initialization vector (IV) to the message (as is the case in the above scenario), stream ciphers are inadequate since they succumb to a simple differential attack[1], even though they are usually the method of choice when the message size must be kept unchanged. Block ciphers might be used, but all-purpose ciphers process data in blocks far larger than allowed (64 bits or more).

To tackle with the posed problem, in this and similar scenarios we assume the following *security hypothesis*, motivated by the analysis in section 4.1:

Security Hypothesis 1. *The message traffic in all situations under consideration is very low, precluding the accumulation of known data blocks and inhibiting the construction of a codebook of plaintext-ciphertext pairs. Specifically, the number of n-bit data blocks encrypted under the same key is always less than $2^{n/2}$.*

A complete codebook would allow an attacker to encrypt or decrypt any message by simple table lookup, without knowledge of the cipher key. Although massive accumulation of data blocks is a "brute force" attack, the small block size would make it feasible if message traffic were high. In the above scenario, assuming that the attacker somehow is able to obtain the plaintexts associated to all ciphertexts, even if token synchronization occurred once a day it would take over ten years to complete the codebook.

In this paper we present SACI, a special-purpose block cipher tailored to supply the scenario described above. SACI process its input in 24-bit data blocks and uses a key of size 96, 144, or 192 bits.

Because of its quite unusual defining parameters, special care had to be taken in the specification of SACI. The cipher design follows the Wide Trail strategy [1]. The most well-known member of the Wide Trail family of ciphers is RIJNDAEL, the Advanced Encryption Standard (AES). Due to the exceptionally constrained environment to which it is targeted, SACI was designed to exhibit *involutional structure* and *cyclic key schedule*.

Involutional structure is found as part of many cipher designs. All classical Feistel networks [2] have this property, as do some more general block ciphers like IDEA [3]. Self-inverse ciphers related to SACI were described in [4, 5, 6, 7, 8]. The importance of involutional structure resides not only in the advantages for implementation, but also in the equivalent security of both encryption and decryption [9].

The key schedule adopted by a block cipher is called cyclic or periodic if (1) it iterates some function on the cipher key to derive the round subkeys, and (2) that function becomes the identity after a certain number of iterations. It is important to notice that, with this setting, the subkeys need not be computed incrementally: they can be computed backwards as often needed by the decryption process, or in a random order, or even all in parallel, a useful feature if the cipher is implemented in dedicated hardware, for instance to be deployed on servers. They can also be computed in-place, contrary to schemes where all subkeys must be precomputed and stored in a large table.

The scenario above may suggest the use of a one-time-pad scheme instead of SACI due to its small block size. Nonetheless, one-time-pad demands completely random numbers in order to work properly and such requirement cannot be achieved by using pseudo-random number generators [10]. This scheme also creates difficulties to up-

[1] If a key is reused in a stream cipher, then the difference between ciphertexts is the same as the difference between plaintexts, so that a single compromised plaintext allows the attacker to recover any other message without knowing the key.

date the cryptographic key and would make the use of tamper resistant memories, when applicable, more expensive.

This document is organized as follows. We introduce basic mathematical tools in section 2. The SACI cipher is described in section 3. Security issues of the resulting design are discussed in section 4. Implementation techniques are presented in section 5, and the performance of SACI on a typical microcontroller used in identification tokens is assessed in section 6. We conclude in section 7.

2 Mathematical preliminaries and notation

2.1 Finite fields

The finite field $\mathrm{GF}(2^8)$ will be represented as $\mathrm{GF}(2)[x]/p(x)$, where $p(x) = x^8 + x^6 + x^3 + x^2 + 1$ is the only primitive pentanomial of degree 8 over $\mathrm{GF}(2)$ for which a primitive cube root of unity is represented as a quartic trinomial (namely, $c(x) = x^{85} \bmod p(x) = x^4 + x^3 + x^2$), which is the simplest form achievable. Since multiplications by the generator $g(x) = x$ of $\mathrm{GF}^*(2^8)$ and by $c(x)$ both occur in the algorithm, it is important that $c(x)$ be as simple as possible for efficiency reasons.

An element $u = u_7x^7 + u_6x^6 + u_5x^5 + u_4x^4 + u_3x^3 + u_2x^2 + u_1x + u_0$ of $\mathrm{GF}(2^8)$ where $u_i \in \mathrm{GF}(2)$ for all $i = 0, \ldots, 7$ will be denoted by the numerical value $u_7 \cdot 2^7 + u_6 \cdot 2^6 + u_5 \cdot 2^5 + u_4 \cdot 2^4 + u_3 \cdot 2^3 + u_2 \cdot 2^2 + u_1 \cdot 2 + u_0$, written in hexadecimal notation. For instance, the polynomial $c(x) = x^4 + x^3 + x^2$ is written `1C`; by extension, the reduction polynomial $p(x)$ is written `14D`.

2.2 MDS codes

We provide a few relevant definitions regarding the theory of linear codes. For a more extensive exposition on the subject we refer to [11].

The Hamming distance between two vectors u and v from the n-dimensional vector space $\mathrm{GF}(2^p)^n$ is the number of coordinates where u and v differ.

The Hamming weight $\mathrm{w_h}(a)$ of an element $a \in \mathrm{GF}(2^p)^n$ is the Hamming distance between a and the null vector of $\mathrm{GF}(2^p)^n$, i.e. the number of nonzero components of a.

A *linear* $[n,k,d]$ *code* over $\mathrm{GF}(2^p)$ is a k-dimensional subspace of the vector space $\mathrm{GF}(2^p)^n$, where the Hamming distance between any two distinct subspace vectors is at least d (and d is the largest number with this property).

A *generator matrix* G for a linear $[n,k,d]$ code $\mathcal{C}$ is a $k \times n$ matrix whose rows form a basis for $\mathcal{C}$. A generator matrix is in *echelon* or *standard* form if it has the form $G = [I_{k\times k} A_{k\times(n-k)}]$, where $I_{k\times k}$ is the identity matrix of order k. We write simply $G = [IA]$ omitting the indices wherever the matrix dimensions are irrelevant for the discussion, or clear from the context.

Linear $[n,k,d]$ codes obey the *Singleton bound*: $d \leqslant n - k + 1$. A code that meets the bound, i.e. $d = n - k + 1$, is called a *maximal distance separable* (MDS) code.

A linear $[n,k,d]$ code $\mathcal{C}$ with generator matrix $G = [I_{k\times k} A_{k\times(n-k)}]$ is MDS if, and only if, every square submatrix formed from rows and columns of A is nonsingular (cf. [11, chapter 11, § 4, theorem 8]).

2.3 Matrices over $\mathrm{GF}(2^m)$

The set of all $3 \times n$ matrices over $\mathrm{GF}(2^m)$ is denoted by $\mathbb{M}_n$, with O and I standing for the zero and identity matrices, respectively.

Consider the following matrices:

$$A = \begin{bmatrix} \texttt{01} & \texttt{01} & \texttt{01} \\ \texttt{00} & \texttt{00} & \texttt{00} \\ \texttt{01} & \texttt{01} & \texttt{01} \end{bmatrix}, B = \begin{bmatrix} \texttt{00} & \texttt{00} & \texttt{00} \\ \texttt{01} & \texttt{01} & \texttt{01} \\ \texttt{01} & \texttt{01} & \texttt{01} \end{bmatrix}, C = \begin{bmatrix} \texttt{01} & \texttt{01} \\ \texttt{01} & \texttt{01} \\ \texttt{01} & \texttt{01} \end{bmatrix}$$

A straightforward inspection reveals that A and B are nilpotent and C is idempotent over $\mathrm{GF}(2^m)$ ($A^2 = B^2 = O$, $C^2 = C$), and also that $AB = BA = O$.

As a consequence, all matrices of form $D = I + aA + bB$ are involutions, i.e. $D^2 = I$, $\forall a, b \in \mathrm{GF}(2^m)$. One can show by direct enumeration that the determinants of all square submatrices formed from rows and columns of D take one of the forms $\{a, a+1, b, b+1, a+b, a+b+1\}$, so that D is MDS iff $a \neq 0, 1$, $b \neq 0, 1$, and $b \neq a, a+1$. The simplest such matrix to display the MDS property is thus $D = I + \texttt{2}A + \texttt{4}B$.

Furthermore, a simple calculation shows that a matrix of form $E = I + cC$ is invertible iff $c \neq 1$, in which case $E^{-1} = I + \left(\frac{c}{c+1}\right)C$. The determinants of all square submatrices formed from rows and columns of E take one of the forms $\{1, c, c+1\}$, so that D is MDS iff $c \neq 0, 1$. If c is a primitive cube root of unity, i.e. if $c^2 + c + 1 = 0$ then, on the one hand $c^3 = 1 \Rightarrow c^2 = 1/c$, and on the other hand $c^2 = c + 1 \Rightarrow 1/(c+1) = 1/c^2 = c \Rightarrow c/(c+1) = c^2 = c + 1$. Thus E^{-1}

assumes a particularly simple form, namely, $E^{-1} = I + (c+1)C$.

Matrices D and E play important roles in SACI (see sections 3.2 and 3.6).

2.4 Cryptographic properties

A product of m distinct Boolean variables is called an m-th order product of the variables. Every Boolean function $f : \mathrm{GF}(2)^n \to \mathrm{GF}(2)$ can be written as a sum over $\mathrm{GF}(2)$ of distinct m-order products of its arguments, $0 \leqslant m \leqslant n$; this is called the algebraic normal form of f. The *nonlinear order* of f, denoted $\nu(f)$, is the maximum order of the terms appearing in its algebraic normal form.

A *linear* Boolean function is a Boolean function of nonlinear order 1, i.e. its algebraic normal form only involves isolated arguments. Given $\alpha \in \mathrm{GF}(2)^n$, we denote by $\ell_\alpha : \mathrm{GF}(2)^n \to GF(2)$ the linear Boolean function consisting of the sum of the argument bits selected by the bits of α:

$$\ell_\alpha(x) = \bigoplus_{i=0}^{n-1} \alpha_i \cdot x_i.$$

A mapping $S : \mathrm{GF}(2^n) \to \mathrm{GF}(2^n), x \mapsto S[x]$, is called a *substitution box*, or S-box for short. An S-box can also be viewed as a mapping $S : \mathrm{GF}(2)^n \to \mathrm{GF}(2)^n$ and therefore described in terms of its component Boolean functions $s_i : \mathrm{GF}(2)^n \to \mathrm{GF}(2), 0 \leqslant i \leqslant n-1$, i.e. $S[x] = (s_0(x), \ldots, s_{n-1}(x))$.

The *nonlinear order* of an S-box S, denoted ν_S, is the minimum nonlinear order over all linear combinations of the components of S:

$$\nu_S = \min_{\alpha \in \mathrm{GF}(2)^n} \{\nu(\ell_\alpha \circ S)\}.$$

The *difference table* of an S-box S is defined as

$$e_S(a,b) = \#\{c \in \mathrm{GF}(2^n) \mid S[c \oplus a] \oplus S[c] = b\}.$$

The δ-*parameter* of an S-box S is defined as

$$\delta_S = 2^{-n} \cdot \max_{a \neq 0,\, b} e_S(a,b).$$

The product $\delta_S \cdot 2^n$ is called the *differential uniformity* of S.

The *correlation* $c(f,g)$ between two Boolean functions f and g can be calculated as follows:

$$c(f,g) = 2^{1-n} \cdot \#\{x \mid f(x) = g(x)\} - 1.$$

The λ-*parameter* of an S-box S is defined as the maximal value for the correlation between linear functions of input bits and linear functions of output bits of S:

$$\lambda_S = \max_{(i,j) \neq (0,0)} c(\ell_i, \ell_j \circ S).$$

The *branch number* $\mathcal{B}$ of a linear mapping $\theta : \mathrm{GF}(2^n) \to \mathrm{GF}(2^n)$ is defined as

$$\mathcal{B}(\theta) = \min_{a \neq 0} \{\mathrm{w_h}(a) + \mathrm{w_h}(\theta(a))\}.$$

2.5 Miscellaneous notation

Given a sequence of functions $f_m, f_{m+1}, \ldots, f_{n-1}, f_n$, $m \leqslant n$, we adopt the notation

$$\bigcirc_{r=m}^{n} f_r \equiv f_n \circ f_{n-1} \circ \cdots \circ f_{m+1} \circ f_m$$

and

$$\bigcirc_{m}^{r=n} f_r \equiv f_m \circ f_{m+1} \circ \cdots \circ f_{n-1} \circ f_n.$$

If $m > n$, both expressions stand for the identity mapping.

3 Description of the SACI primitive

The SACI cipher is an iterated block cipher that operates on a 24-bit *cipher state*. It uses a $48t$-bit *cipher key* $(2 \leqslant t \leqslant 4)$ organized as a $3 \times 2t$ matrix of $\mathrm{GF}(2^8)$ elements. The cipher key K is subjected to an in-place *key evolution* process, whereby a linear transform ω of period $m = 6t$ and a suitable *schedule constant* are repeatedly applied to produce *key stages* from which the round subkeys are computed (two per stage). After m key evolution steps the key stage only differs from the original cipher key by a constant; an extra addition of this constant entirely recovers K and resets the cipher for the next operation.

An R-round iterated cipher needs $R+1$ subkeys. Since an evolution of order $6t$ gives rise to $12t$ 24-bit subkeys, the number of rounds is bound by $R \leqslant 12t - 1$, more than enough the security needs of SACI. The actual choice is $R = 12t - 2$ rounds.

In the following we will individually define the component mappings and constants that build up SACI, then specify the complete cipher in terms of these components.

3.1 The nonlinear layer γ

Function $\gamma : \mathbb{M}_n \to \mathbb{M}_n$ consists of the parallel application of a nonlinear S-box $S : \mathrm{GF}(2^8) \to \mathrm{GF}(2^8)$ to all bytes of the argument individually:

$$\gamma(a) = b \Leftrightarrow b_{i,j} = S[a_{i,j}],\ 0 \leqslant i < 3,\ 0 \leqslant j < n.$$

The S-box in SACI is the same defined for the block ciphers ANUBIS and KHAZAD [4, 5], designed to display $\delta_S = 2^{-5}$, $\lambda_S = 2^{-2}$, and $\nu_S = 7$ and hence to thwart differential, linear, and interpolation attacks. It was also constructed so that $S[S[x]] = x$, $\forall x \in \mathrm{GF}(2^8)$. Therefore, γ is an involution.

The actual S-box can be computed on demand with algorithm 1 from two mini-boxes (see table 3) that fit in just 16 bytes. Alternatively and more commonly, if space is available it can be precomputed and stored in 256 bytes.

Algorithm 1 Computing $S[u]$ from the mini-boxes P and Q

```
INPUT: $u(x) \in \mathrm{GF}(2^8)$, represented as a byte $u$.
OUTPUT: $S[u]$.
1: $u_h \leftarrow P[(u \gg 4)\,\&\,\mathtt{F}],\ u_l \leftarrow Q[u\,\&\,\mathtt{F}]$
2: $u'_h \leftarrow Q[(u_h\,\&\,\mathtt{C}) \oplus ((u_l \gg 2)\,\&\,\mathtt{3})],\ u'_l \leftarrow P[((u_h \ll 2)\,\&\,\mathtt{C}) \oplus (u_l\,\&\,\mathtt{3})]$
3: $u_h \leftarrow P[(u'_h\,\&\,\mathtt{C}) \oplus ((u'_l \gg 2)\,\&\,\mathtt{3})],\ u_l \leftarrow Q[((u'_h \ll 2)\,\&\,\mathtt{C}) \oplus (u'_l\,\&\,\mathtt{3})]$
4: return $(u_h \ll 4) \oplus u_l$
```

3.2 The linear diffusion layer θ

The diffusion layer $\theta : \mathbb{M}_n \to \mathbb{M}_n$ is a linear mapping based on the $[6,3,4]$ MDS code with generator matrix $G_D = [I\,D]$ where $D = I + \mathtt{2}A + \mathtt{4}B$, i.e.

$$D = \begin{bmatrix} \mathtt{03} & \mathtt{02} & \mathtt{02} \\ \mathtt{04} & \mathtt{05} & \mathtt{04} \\ \mathtt{06} & \mathtt{06} & \mathtt{07} \end{bmatrix},$$

so that

$$\theta(a) = b \Leftrightarrow b = D \cdot a.$$

Since D is self-inverse, θ is an involution.

Circulant matrices [12, 13] are not suitable for the diffusion layer because no circulant matrix can be involutional. Cauchy matrices [7, 8] might have been an option, but the resulting coefficients are in general very complex, impairing efficient implementation.

The actual choice involves the simplest possible coefficients, in the sense of minimum polynomial degree and minimum Hamming weight.

3.3 The key addition $\sigma[k]$

The affine key addition $\sigma[k] : \mathbb{M}_n \to \mathbb{M}_n$ consists of the bitwise addition of a key matrix $k \in \mathbb{M}_n$, i.e.

$$\sigma[k](a) = b \Leftrightarrow b_{i,j} = a_{i,j} \oplus k_{i,j},\ 0 \leqslant i < 3,\ 0 \leqslant j < n.$$

Of course $n = 1$ when this mapping is applied to the cipher state. However, we define it in general because it is also used to introduce schedule constants in the key schedule. The key addition so defined is obviously an involution.

3.4 Key representation

A $48t$-bit user key $\mathcal{K}$, $2 \leqslant t \leqslant 4$, externally stored as a byte array of length $6t$, is internally represented as a matrix $K \in \mathbb{M}_{2t}$ such that

$$K_{i,j} = \mathcal{K}[i + 3j],\ 0 \leqslant i < 3,\ 0 \leqslant j < 2t.$$

In other words, the user key is mapped to the cipher key by columns (not by rows).

3.5 Schedule constants

The *schedule constants* $q^{(s)}$ are constant matrices defined as $q^{(0)} = 0$ and, for $s > 0$ and $0 \leqslant j < 2t$,

$$q^{(s)}_{i,j} = \begin{cases} S[2t(s-1) + j], & \text{if } i = 0, \\ 0, & \text{otherwise.} \end{cases}$$

Good schedule constants should not be equal for all bytes in a state, and also not equal for all bit positions in a byte. They should also be different in each round. The actual choice meets these constraints. Taking values from the S-box itself avoids the need for any extra storage. In fact, exactly t S-box entries are used per round.

It is convenient to define also the accumulated schedule constant $Q^{(s)}$ as

$$Q^{(s)} = \sum_{i=0}^{s} \omega^i(q^{(i)}).$$

3.6 The key evolution ψ_s

The cipher key is updated during the cipher operation by a reversible transform defined by two linear operations as follows.

Let $\pi : \mathbb{M}_{2t} \to \mathbb{M}_{2t}$ be the linear transform such that

$$\pi(a) = b \Leftrightarrow \begin{cases} b_{0,j} &=& a_{0,j}, \\ b_{1,j} &=& a_{1,(j+1) \bmod 2t}, \\ b_{2,j} &=& a_{2,(j-1) \bmod 2t}, \end{cases}$$

i.e. keeps the first row of its argument unchanged, rotates the second row one position to the left, and rotates the third row one position to the right.

Let $\mu : \mathbb{M}_n \to \mathbb{M}_n$ be the linear transform such that

$$\mu(a) = E \cdot a,$$

where $E = I + c(x)C$ with $c(x) = x^4 + x^3 + x^2$. Its inverse is $\mu^{-1}(a) = E^{-1} \cdot a$ where $E^{-1} = I + (c(x)+1)C$.

Define $\omega \equiv \mu \circ \pi$, and let ω^m denote the composition of ω with itself m times. Let $a \in \mathbb{M}_{2t}$. The following theorem holds:

Theorem 1. *The period of ω over $\mathbb{M}_{2t}$ is $m = 6t$ for $2 \leqslant t \leqslant 4$. In other words, $m = 6t$ is the smallest positive integer such that $\omega^m(a) = a$, $\forall a \in \mathbb{M}_{2t}$.*

Proof. By direct inspection. The idea is to compute ω^m on a basis of $\mathbb{M}_{2t}$, e.g. $\{e^{(kl)} \mid e^{(kl)}_{ij} = \delta_{ki}\delta_{lj}\}$, and verifying that $\omega^m(e^{(kl)}) \neq e^{(kl)}$ for $1 \leqslant m < 6t$ but $\omega^m(e^{(kl)}) \neq e^{(kl)}$ for $m = 6t$. This process is lengthy and tedious but straightforward; it is omitted here for the sake of brevity. □

Let $K \in \mathbb{M}_n$ be the cipher key. Define the initial key stage to be $K^{(0)} \equiv K$. The *key evolution function* $\psi_s : \mathbb{M}_n \to \mathbb{M}_n$ computes key stage $K^{(s)}$ from key stage $K^{(s-1)}$. It is defined as $\psi_s \equiv \omega \circ \sigma[q^{(s)}]$, i.e.

$$K^{(s)} = \psi_s(K^{(s-1)}) = \left(\bigcirc_{i=1}^{s} \omega \circ \sigma[q^{(i)}] \right)(K) = \omega^s(K) + Q^{(s)}.$$

The key schedule derives subkeys from key stages $K^{(0)}$ through $K^{(m-1)}$. It is clear that an extra application of ω and subsequent addition of $Q^{(m-1)}$ recovers the original cipher key, i.e. $K = \omega(K^{(m-1)}) + Q^{(m-1)}$. Only $Q^{(m-1)}$ needs to be stored for sequential processing; in all scheduling steps but this finalization the simpler constants $q^{(s)}$ can be used. This *cyclic schedule* behavior resets the cipher; therefore, the key evolution process can be conducted in-place; no extra storage is needed for the key stages, which can always be computed on-the-fly at low computational cost.

3.7 The key selection ϕ_r

The effective round subkeys $\kappa^{(r)}$ needed by the cipher are computed (either directly from the cipher key K or indirectly from some key stage $K^{(s)}$) via the *key selection function*, $\phi_r : \mathbb{M}_n \to \mathbb{M}_1$.

Let $\zeta : \mathbb{M}_n \to \mathbb{M}_2$ be the linear transform such that

$$\zeta(a) = b \Leftrightarrow b = a \cdot V^{(n)},$$

where $V^{(n)}$ is the $n \times 2$ MDS Vandermonde matrix given by

$$\left.\begin{array}{rcl} V^{(n)}_{i,0} &=& 1 \\ V^{(n)}_{i,1} &=& x^i \end{array}\right\} 0 \leqslant i < n.$$

The composition of the nonlinear layer γ and the ζ transform to $K^{(s)}$ produces a pair of round subkeys $(\kappa^{(2s)}, \kappa^{(2s+1)}) \equiv (\zeta \circ \gamma)(K^{(s)})$. In other words, the key selection function is such that:

$$\kappa^{(r)} = \phi_r(K) \Leftrightarrow \kappa^{(r)}_i = (\zeta \circ \gamma)(K^{(\lfloor r/2 \rfloor)})_{i, r \bmod 2},\ 0 \leqslant i < 3.$$

3.8 The complete cipher

SACI is defined for the cipher key $K \in \mathbb{M}_n$ as the mapping SACI$[K] : \mathbb{M}_1 \to \mathbb{M}_1$ given by

$$\text{SACI}[K] \equiv \sigma[\kappa^{(R)}] \circ \gamma \circ \left(\bigcirc_{r=1}^{R-1} \sigma[\kappa^{(r)}] \circ \theta \circ \gamma \right) \circ \sigma[\kappa^{(0)}].$$

The initial key addition $\sigma[\kappa^{(0)}]$ is called *whitening*. The composite mapping $\rho[\kappa^{(r)}] \equiv \sigma[\kappa^{(r)}] \circ \theta \circ \gamma$ is called the *round function* for the r-th round; by convenience, the related mapping $\rho'[\kappa^{(R)}] \equiv \sigma[\kappa^{(R)}] \circ \gamma$ is called the *last round function*, although it is not the same as the round function.

Figure 1 helps to make clear the steps described by SACI[K], showing the cipher structure and how each layer fits in each step.

3.9 The inverse cipher

We now show that SACI is an involutional cipher, in the sense that the only difference between the cipher and its inverse is in the key schedule.

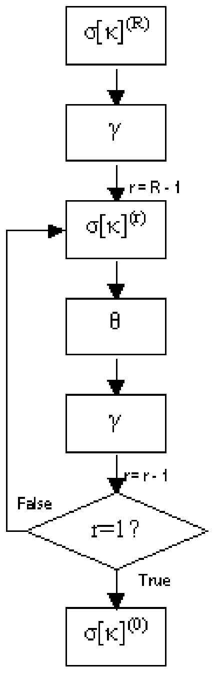

Figure 1: Cipher Structure

Lemma 1. $\theta \circ \sigma[\kappa^{(r)}] = \sigma[\theta(\kappa^{(r)})] \circ \theta$.

Proof. It suffices to notice that $(\theta \circ \sigma[\kappa^{(r)}])(a) = \theta(\kappa^{(r)} + a) = \theta(\kappa^{(r)}) + \theta(a) = (\sigma[\theta(\kappa^{(r)})] \circ \theta)(a)$, for any input a. $\square$

Theorem 2. *Let $\alpha[\kappa^{(0)}, \ldots, \kappa^{(R)}]$ stand for* SACI *encryption under the sequence of round keys $\kappa^{(0)}, \ldots, \kappa^{(R)}$, and let the decryption keys be defined as $\bar{\kappa}^{(0)} \equiv \kappa^{(R)}$, $\bar{\kappa}^{(R)} \equiv \kappa^{(0)}$, and $\bar{\kappa}^{(r)} \equiv \theta(\kappa^{(R-r)})$ for $0 < r < R$. Then $\alpha^{-1}[\kappa^{(0)}, \ldots, \kappa^{(R)}] = \alpha[\bar{\kappa}^{(0)}, \ldots, \bar{\kappa}^{(R)}]$.*

Proof. We start from the definition of $\alpha[\kappa^{(0)}, \ldots, \kappa^{(R)}]$:

$$\alpha[\kappa^{(0)}, \ldots, \kappa^{(R)}] = \\ \sigma[\kappa^{(R)}] \circ \gamma \circ \left(\bigcirc_{r=1}^{R-1} \sigma[\kappa^{(r)}] \circ \theta \circ \gamma \right) \circ \sigma[\kappa^{(0)}].$$

Since the component functions are involutions, the inverse transform is obtained by applying them in reverse order:

$$\alpha^{-1}[\kappa^{(0)}, \ldots, \kappa^{(R)}] = \\ \sigma[\kappa^{(0)}] \circ \left(\bigcirc_{1}^{r=R-1} \gamma \circ \theta \circ \sigma[\kappa^{(r)}] \right) \circ \gamma \circ \sigma[\kappa^{(R)}].$$

Lemma 1 leads to:

$$\alpha^{-1}[\kappa^{(0)}, \ldots, \kappa^{(R)}] = \\ \sigma[\kappa^{(0)}] \circ \left(\bigcirc_{1}^{r=R-1} \gamma \circ \sigma[\theta(\kappa^{(r)})] \circ \theta \right) \circ \gamma \circ \sigma[\kappa^{(R)}].$$

The associativity of function composition allows for slightly changing the grouping of operations:

$$\alpha^{-1}[\kappa^{(0)}, \ldots, \kappa^{(R)}] = \\ \sigma[\kappa^{(0)}] \circ \gamma \circ \left(\bigcirc_{1}^{r=R-1} \sigma[\theta(\kappa^{(r)})] \circ \theta \circ \gamma \right) \circ \sigma[\kappa^{(R)}].$$

Finally, by substituting $\bar{\kappa}^{(r)}$ in the above equation we arrive at:

$$\alpha^{-1}[\kappa^{(0)}, \ldots, \kappa^{(R)}] = \\ \sigma[\bar{\kappa}^{(R)}] \circ \gamma \circ \left(\bigcirc_{r=1}^{R-1} \sigma[\bar{\kappa}^{(r)}] \circ \theta \circ \gamma \right) \circ \sigma[\bar{\kappa}^{(0)}].$$

That is, $\alpha^{-1}[\kappa^{(0)}, \ldots, \kappa^{(R)}] = \alpha[\bar{\kappa}^{(0)}, \ldots, \bar{\kappa}^{(R)}]$. $\square$

Corollary 1. *The* SACI *cipher has involutional structure, in the sense that the only difference between the cipher and its inverse is in the key schedule.*

3.10 The inverse schedule

If the round subkeys are to be generated sequentially and in-place, it is advantageous to start from the cipher key K and compute $K^{(s)}$ backwards. In other words, compute $K^{(m-1)} = \omega^{-1}(K + Q^{(m-1)})$ and use the relation $K^{(s)} = \psi_s^{-1}(K^{(s+1)}) = \sigma[q^{(s)}] \circ \omega^{-1}(K^{(s+1)})$.

3.11 Extensions

The range of the t parameters that defines the key sizes and the corresponding number of rounds was chosen so that the period of ω is $6t$. Although one would hardly need larger keys, it is straightforward to extend SACI to accept them, as long as ω continues to display period $6t$. This corresponds to taking t of form 2×2^n or 3×2^n, for any $n \geqslant 0$. However, we advise against this unless a different method of selecting schedule constants is chosen: the key evolution process takes $2t$ distinct S-box entries for each group of $5t$ evolution steps, hence to avoid repetitions the total number of entries must satisfy $2t \times 5t \leqslant 256$, or $t \leqslant 5$.

4 Analysis

4.1 Block collisions

If an n-bit block cipher encrypts about $2^{n/2}$ plaintexts under the same key, then with probability $1/2$ one can find two ciphertexts c_i and c_j corresponding to plaintexts p_i and p_j such that $p_i \oplus p_j = c_i \oplus c_j$. If one of these plaintexts is known, the other can be easily recovered. Although this behavior, pointed out by L. Knudsen in his thesis [14], is not much of a threat, it suggests the quantitative limit for the security hipothesis 1.

4.2 Differential and linear cryptanalysis

Because the branch number of θ is $\mathcal{B} = 4$ (cf. [15], proposition 1), no differential characteristic over two rounds has probability larger than $\delta^{\mathcal{B}} = (2^{-5})^4 = 2^{-20}$, and no linear approximation over two rounds has input-output correlation larger than $\lambda^{\mathcal{B}} = (2^{-2})^4 = 2^{-8}$. This makes classical differential or linear attacks, which need characteristics with probability larger than 2^{-23} or input-output correlations larger than 2^{-11} over *all* rounds, as well as some advanced variants like differential-linear attacks, very unlikely to succeed for the full cipher, given its conservative number of rounds.

4.3 Truncated differentials

The concept of truncated differentials was introduced in [16], and typically applies to ciphers in which all transformations operate on well aligned data blocks, as is the case for SACI where all transformations operate on bytes rather than individual bits. However, the fact that all submatrices of D are nonsingular makes a truncated differential attack against more than a few rounds of SACI impossible, because the S/N ratio of an attack becomes too low. For 4 rounds or more, no truncated differential attacks can be mounted.

4.4 Interpolation attacks

Interpolation attacks [17] generally depend on the cipher components (particularly the S-box) having simple algebraic structures that can be combined to give polynomial or rational expressions with manageable complexity. The involved expression of the S-box in $\mathrm{GF}(2^8)$, combined with the effect of the diffusion layer, makes this type of attack infeasible for more than a few rounds.

4.5 Weak keys

The weak keys we discuss are keys that result in a block cipher mapping with detectable weaknesses. The best known case of such weak keys are those of IDEA [1]. Typically, this occurs for ciphers where the nonlinear operations depend on the actual key value. This is not the case for SACI, where keys are applied using exclusive-or and all nonlinearity is in the fixed S-box. In SACI, there is no restriction on key selection.

4.6 Related-key cryptanalysis

Related-key attacks generally rely upon slow diffusion and/or symmetry in the key schedule. The SACI key schedule was itself designed according to the Wide Trail strategy and features fast, nonlinear diffusion of cipher key differences to the round keys. This makes related-key attacks exceedingly unlikely.

4.7 Saturation attacks

The saturation attack described in [18] against the SHARK cipher works against SACI reduced to 3 rounds. This attack recovers one byte of the last round key at a time, and for each one requires 2^9 chosen plaintexts. However, almost all wrong key values can be eliminated after processing a single set of 2^8 plaintexts. The workload to recover one key byte is 2^8 key guesses for each of 2^8 chosen plaintexts, or 2^{16} S-box lookups, and three times as much to recover the last subkey completely.

4.8 A general extension attack

Any n-round attack can be extended against $n+1$ or more rounds for long keys by simply guessing the whole $\kappa^{(n+1)}$ round key and proceeding with the n-round attack [19]. Each extra round increases the complexity by a factor 2^{24} S-box lookups. The saturation attack against 3 rounds of SACI has complexity about 3×2^{16} S-box lookups; hence a 6-round extension costs 3×2^{88} S-box lookups and recovers 96-bit keys, an 8-round extension costs 3×2^{136} S-box lookups and recovers 144-bit keys, and a 10-round extension costs 3×2^{184} S-box lookups and recovers 192-bit keys. This improvement is ineffective against the full cipher, which consists of many more rounds ($12t - 2$ rather than $2t + 2$).

4.9 Other attacks

For completeness, we mention other lines of attack that, although less effective than those described above, are of theoretical interest nonetheless.

An extension of the Biham-Keller impossible differential attack on RIJNDAEL reduced to 5 rounds [20] can be applied to SACI reduced to 3 rounds. The attack requires about 2^{13} chosen plaintexts and an effort of 2^{24} encryptions, worse than the saturation attack.

The boomerang attack [21] benefits from ciphers whose diffusion is slow or incomplete resulting on different strengths on encryption and decryption; this is hardly the case for SACI, due to its involutional structure.

The Gilbert-Minier attack [22] requires the cipher to feature a two-level diffusion structure; there is no obvious way to mount it against the one-level diffusion scheme of SACI (represented by θ on SACI).

Muller's attack against KHAZAD [23] is based on the details of the key schedule adopted for that cipher, and shows no hope of working against SACI.

Attacks based on linear cryptanalysis can sometimes be improved by using nonlinear approximations [24]. However, with the current state of the art the application of nonlinear approximations seems limited to the first and/or the last round of a linear approximation. This seems to be even more so for ciphers using strongly nonlinear S-boxes, like SACI.

There is no consensus regarding the effectiveness of algebraic attacks like XSL [25]. Because of the heuristic nature of such attacks, theoretical complexity analyses remain controversial, while experimental evidence tends to deny their viability. At the time of this writing no such attack has ever been demonstrated, not even against simplified versions of AES, whose S-box has a far simpler algebraic structure than the S-box used in SACI. It is pointed out, however, that this very S-box in a different context has already been subjected to third-party scrutiny, with the explicit goal of assessing its resistance against algebraic attacks [26], and no evidence of weaknesses was found.

In summary, provided the traffic of messages is low enough to preclude plain accumulation of data blocks (see the security hypothesis in section 1), no method could be found to attack the cipher faster than exhaustive key search. SACI is expected, for all key lengths defined, to behave as good as can be expected from a block cipher with the given block and key lengths.

5 Implementation issues

On an 8-bit processor with a limited amount of RAM, e.g. a typical smart card processor, the nonlinear substitution layer is performed bytewise, combined with the $\sigma[k]$ transformation. For θ and ψ_s, it is necessary to implement matrix multiplication.

Multiplication by the polynomial $g(x) = x$ in $\mathrm{GF}(2^8)$ is of central importance. The most efficient way to implement it uses a 256-byte table X such that $X[u] \equiv x \cdot u$. It can also be implemented with one shift and one conditional XOR. Algorithm 2 calculates $x \cdot u(x)$. For simplicity of exposition, from now on we assume that table X is available.

Algorithm 2 Computing $x \cdot u(x)$

```
INPUT: $u(x) \in \mathrm{GF}(2^8)$, represented as a byte $u$.
OUTPUT: $x \cdot u(x)$.
1: $v \leftarrow u \ll 1$
2: if $v \geqslant 256$ then
3:     $v \leftarrow v \oplus \mathtt{14D}$
4: end if
5: return $v$
```

Multiplication by the polynomial $c(x) = x^4 + x^3 + x^2$ in $\mathrm{GF}(2^8)$ is equally important. The most efficient way to implement it uses a 256-byte table Z such that $Z[u] \equiv c(x) \cdot u$. It can also be implemented using the X table, using 2 XORs and 4 table lookups. Algorithm 3 calculates $c(x) \cdot u(x)$.

Algorithm 3 Computing $c(x) \cdot u(x)$

```
INPUT: $u(x) \in \mathrm{GF}(2^8)$, represented as a byte $u$.
OUTPUT: $c(x) \cdot u(x)$.
1: $v \leftarrow X[X[X[X[u] \oplus u] \oplus u]]$
2: return $v$
```

Algorithm 4 calculates $D \cdot a$ for a vector $a \in \mathbb{M}_1$. This implementation requires 6 XORs and 2 table lookups.

Algorithm 4 Computing $D \cdot a$

```
INPUT: $a \in \mathbb{M}_1$.
OUTPUT: $D \cdot a$.
1: $v \leftarrow X[a_0 \oplus a_1 \oplus a_2],\ w \leftarrow X[v]$
2: $b_0 \leftarrow a_0 \oplus v,\ b_1 \leftarrow a_1 \oplus w,\ b_2 \leftarrow a_2 \oplus v \oplus w$
3: return $b$
```

Algorithm 5 calculates $E \cdot a$ for a vector $a \in \mathbb{M}_1$ (or a column of a matrix in $\mathbb{M}_n$) at the cost of 5 XORs and 1 table lookup if Z is available, or 7 XORs and 4 table lookups if only X is available. It also calculates $E^{-1} \cdot a$ at the cost of one extra XOR.

Algorithm 6 calculates $a \cdot V^{(n)}$ for a vector $a \in \mathbb{M}_n$. This implementation requires $2(n-1)$ XORs and $n-1$ table lookups.

Algorithm 5 Computing $E \cdot a$ or $E^{-1} \cdot a$

```
INPUT: $a \in \mathbb{M}_1$.
INPUT: $e$, a Boolean flag signaling whether $E \cdot a$ (**true**) or $E^{-1} \cdot a$ (**false**) is
    to be computed.
1: $v \leftarrow a_0 \oplus a_1 \oplus a_2$
2: if $e$ then
3:     $v \leftarrow Z[v]$
4: else
5:     $v \leftarrow Z[v] \oplus v$
6: end if
7: $b_0 \leftarrow a_0 \oplus v,\ b_1 \leftarrow a_1 \oplus v,\ b_2 \leftarrow a_2 \oplus v$
8: return $b$
```

Algorithm 6 Computing $a \cdot V^{(n)}$

```
INPUT: $a \in \mathbb{M}_n$.
OUTPUT: $a \cdot V^{(n)} \in \mathbb{M}_2$.
1: for $i \leftarrow 0 \ldots 2$ do
2:     $b_{i,0} \leftarrow a_{i,n-1},\ b_{i,1} \leftarrow a_{i,n-1}$
3:     for $j \leftarrow n-2 \ldots 0$ do
4:         $b_{i,0} \leftarrow b_{i,0} \oplus a_{i,j},\ b_{i,1} \leftarrow X[b_{i,1}] \oplus a_{i,n-1}$
5:     end for
6: end for
7: return $b$
```

6 Performance evaluation

In order to evaluate its performance on a typical environment, SACI was implemented on an 8-bit microcontroller and compared against an AES implementation. Although AES is not suitable for the target scenario because of its large block size, it suitably plays the role of a standard gauge for performance comparisons. The microcontroller chosen for the analysis is a PIC18F6490 from Microchip. It has important support devices, like an LCD controller and an oscillator, that make the token circuitry much simpler. Both ciphers were written in C and compiled using the Microchip C18 compiler 2.3. All optimizations were enabled.

The AES implementation is restricted to, and optimized for, 128-bits keys. The SACI implementation, on the contrary, accepts all allowed key sizes.

Table 1 summarizes the storage requirements for each cipher.

For running time assessment we set the microcontroller at 1 MHz. Each cipher was invoked 1000 times under the same key, the initial plaintext being an array of null bytes and the input of each subsequent invocation being the ciphertext output by the preceding encryption. Table 2 summarizes our results.

Table 1: Storage requirements

cipher	Storage (bytes)
AES (encryption only)	1012
SACI (encryption only)	1296
AES (full)	1780
SACI (full)	1524

Table 2: Running times (in ms)

cipher	key schedule + encryption	encryption only
AES-128	173 ± 3	125 ± 3
SACI-96	115 ± 2	35 ± 2
SACI-144	232 ± 3	54 ± 5
SACI-192	393 ± 4	73 ± 2

Plotting graphics for the running times (figures 2 and 3) we see that both algorithms show equivalent performance when performing both key scheduling and encryption in each run of the algorithm. When just the encryption is performed we can notice that SACI presents a better performance than AES. It is important to keep in mind that the number of rounds for SACI is much more conservative, and hence higher, than it is for AES.

From table 2 we can see that SACI running time is adequate for use in a token scenario because the user would have to wait less than 1 second in order to get a result from the system (393 ± 4ms in worst case scenario). If the device employed on the authentication scheme has enough memory to hold the whole key scheduling the user of the system would notice a even better performance from the system.

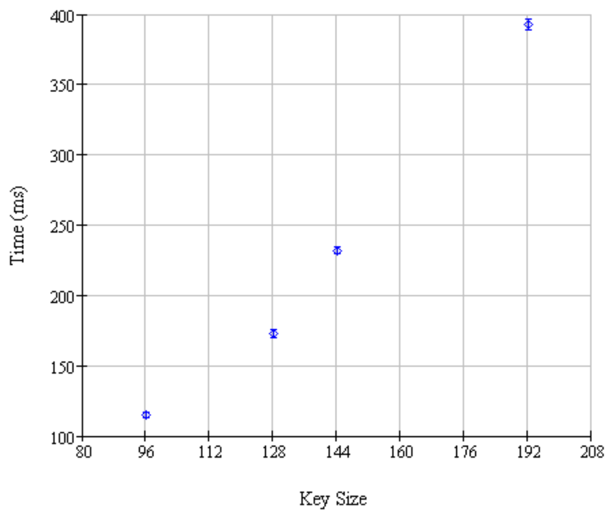

Figure 2: Running times for key schedule and encryption

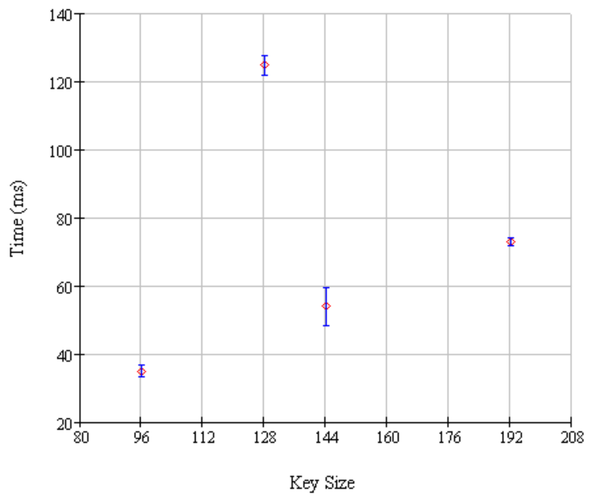

Figure 3: Running times for encryption only

7 Conclusion

We have described SACI, a new, special-purpose block cipher targeted at applications with heavily constrained resources, infrequent updating of keys and very low traffic of encrypted messages (the latter condition being an important security hypothesis). SACI was designed according to the Wide Trail strategy and displays both involutional structure and cyclic key schedule, two important features to overcome the limitations of the target platforms. The resulting cipher is compact and efficient, yet the security analysis shows quantitatively that, properly implemented and deployed, SACI is about as strong a cipher as can be designed under the assumed operational constraints.

8 Acknowledgements

We are grateful to Jorge Nakahara Jr. for his invaluable comments during the preparation of this work.

References

[1] J. Daemen, *Cipher and hash function design strategies based on linear and differential cryptanalysis*. Doctoral dissertation, Katholiek Universiteit Leuven, March 1995.

[2] H. Feistel, “Cryptography and computer privacy,” *Scientific American*, vol. 228, no. 5, pp. 15–23, 1973.

Table 3: The P and Q mini-boxes

u	$P[u]$	$Q[u]$
0	3	9
1	F	E
2	E	5
3	0	6
4	5	A
5	4	2
6	B	3
7	C	C
8	D	F
9	A	0
A	9	4
B	6	D
C	7	7
D	8	B
E	2	1
F	1	8

[3] X. Lai, J. L. Massey, and S. Murphy, “Markov ciphers and differential cryptanalysis,” in *Advances in Cryptology – Eurocrypt’91*, vol. 547 of *Lecture Notes in Computer Science*, (Brighton, UK), pp. 17–38, Springer, 1991.

[4] P. S. L. M. Barreto and V. Rijmen, “The ANUBIS block cipher,” in *First open NESSIE Workshop*, (Leuven, Belgium), NESSIE Consortium, November 2000.

[5] P. S. L. M. Barreto and V. Rijmen, “The KHAZAD legacy-level block cipher,” in *First open NESSIE Workshop*, (Leuven, Belgium), NESSIE Consortium, November 2000.

[6] J. Daemen, M. Peeters, G. V. Assche, and V. Rijmen, “The NOEKEON block cipher,” in *First open NESSIE Workshop*, (Leuven, Belgium), NESSIE Consortium, November 2000.

[7] A. M. Youssef, S. E. Tavares, and H. M. Heys, “A new class of substitution-permutation networks,” in *Selected Areas in Cryptography – SAC’96*, Proceedings, pp. 132–147, 1996.

[8] A. M. Youssef, S. Mister, and S. E. Tavares, “On the design of linear transformations for substitution permutation encryption networks,” in *Selected Areas in Cryptography – SAC’97*, Proceedings, pp. 40–48, 1997.

[9] L. R. Knudsen and D. Wagner, “On the structure of skipjack,” *Discrete Applied Mathematics*, vol. 111, no. 1, pp. 103–116, 2001.

[10] R. Oppliger, *Contemporary Cryptography*. Norwood, USA: Artech House, 2005.

[11] F. J. MacWilliams and N. J. A. Sloane, *The theory of error-correcting codes*, vol. 16. North-Holland Mathematical Library, 1977.

[12] J. Daemen, L. R. Knudsen, and V. Rijmen, "The block cipher SQUARE," in *Fast Software Encryption – FSE'97*, vol. 1267 of *Lecture Notes in Computer Science*, (Haifa, Israel), pp. 149–165, Springer, 1997.

[13] NIST, *Federal Information Processing Standard (FIPS 197) – Advanced Encryption Standard (AES)*. National Institute of Standards and Technology – NIST, November 2001.

[14] L. Knudsen, *Block ciphers – Analysis, Design and Applications*. Ph.d. thesis, Århus University, 1994.

[15] V. Rijmen, J. Daemen, B. Preneel, A. Bosselaers, and E. D. Win, "The cipher SHARK," in *Fast Software Encryption – FSE'96*, vol. 1039 of *Lecture Notes in Computer Science*, pp. 99–111, Springer, 1996.

[16] L. R. Knudsen, "Truncated and higher order differentials," in *Fast Software Encryption – FSE'95*, vol. 1008 of *Lecture Notes in Computer Science*, (New York, USA), pp. 196–211, Springer, 1995.

[17] T. Jakobsen and L. R. Knudsen, "The interpolation attack on block ciphers," in *Fast Software Encryption – FSE'95*, vol. 1267 of *Lecture Notes in Computer Science*, (Haifa, Israel), pp. 28–40, Springer, 1997.

[18] V. Rijmen, *Cryptanalysis and design of iterated block ciphers*. Doctoral dissertation, Katholiek Universiteit Leuven, October 1997.

[19] S. Lucks, "Attacking seven rounds of RIJNDAEL under 192-bit and 256-bit keys," in *Third Advanced Encryption Standard Candidate Conference – AES3*, (New York, USA), pp. 215–229, NIST, April 2000.

[20] E. Biham and N. Keller, "Cryptanalysis of reduced variants of RIJNDAEL," in *Third Advanced Encryption Standard Candidate Conference – AES3*, (New York, USA), NIST, April 2000.

[21] D. Wagner, "The boomerang attack," in *Fast Software Encryption – FSE'99*, vol. 1636 of *Lecture Notes in Computer Science*, (Rome, Italy), pp. 156–170, Springer, 1999.

[22] H. Gilbert and M. Minier, "A collision attack on seven rounds of RIJNDAEL," in *Third Advanced Encryption Standard Candidate Conference – AES3*, (New York, USA), pp. 230–241, NIST, April 2000.

[23] F. Muller, "A new attack against KHAZAD," in *Advances in Cryptology – Asiacrypt'2003*, vol. 2894 of *Lecture Notes in Computer Science*, pp. 347–358, Springer, 2003.

[24] L. R. Knudsen and M. J. B. Robshaw, "Non-linear approximations in linear cryptanalysis," in *Advances in Cryptology – Eurocrypt'96*, vol. 1070 of *Lecture Notes in Computer Science*, pp. 224–236, Springer, 1996.

[25] N. Courtois and J. Pieprzyk, "Cryptanalysis of block ciphers with overdefined systems of equations," in *Advances in Cryptology – Asiacrypt'2002*, vol. 2501 of *Lecture Notes in Computer Science*, pp. 267–287, Springer, 2002.

[26] A. Biryukov and C. DeCannière, "Block ciphers and systems of quadratic equations," in *Fast Software Encryption – FSE'2003*, vol. 2887 of *Lecture Notes in Computer Science*, pp. 274–289, Springer, 2003.