The Anubis Block Cipher

Paulo S.L.M. Barreto[1]* and Vincent Rijmen[2]**

[1] Scopus Tecnologia S. A.
A. Mutinga, 4105 - Pirituba
BR–05110–000 São Paulo (SP), Brazil
pbarreto@scopus.com.br
[2] Katholieke Universiteit Leuven, Dept. ESAT,
Kard. Mercierlaan 94,
B–3001 Heverlee, Belgium
vincent.rijmen@esat.kuleuven.ac.be

Abstract. Anubis is a 128-bit block cipher that accepts a variable-length key. The cipher is a uniform substitution-permutation network whose inverse only differs from the forward operation in the key schedule. The design of both the round transformation and the key schedule is based upon the Wide Trail strategy and permits a wide variety of implementation tradeoffs.

1 Introduction

In this document we describe Anubis, a 128-bit block cipher that accepts a variable-length key.

Although Anubis is not a Feistel cipher, its structure is designed so that by choosing all round transformation components to be involutions, the inverse operation of the cipher differs from the forward operation in the key scheduling only. This property will allow reducing the required chip area in a hardware implementation, as well as the code and table size, which can be important when Anubis is used e.g. in a Java applet.

Anubis was designed according to the Wide Trail strategy [4]. In the Wide Trail strategy, the round transformation of a block cipher is composed of different invertible transformations, each with its own functionality and requirements. The *linear diffusion layer* ensures that after a few rounds all the output bits depend on all the input bits. The *nonlinear layer* ensures that this dependency is of a complex and nonlinear nature. The *round key addition* introduces the key material. One of the advantages of the Wide Trail strategy is that the different components can be specified quite independently from one another. We follow the Wide Trail strategy in the design of the key scheduling algorithm as well.

* Co-sponsored by the Laboratório de Arquitetura e Redes de Computadores (LARC) do Departamento de Engenharia de Computação e Sistemas Digitais da Escola Politécnica da Universidade de São Paulo (Brazil)

** F.W.O. Postdoctoral Researcher, sponsored by the Fund for Scientific Research – Flanders (Belgium)

This document is organised as follows. The mathematical preliminaries and notation employed are described in section 2. A mathematical description of the ANUBIS primitive is given in section 3. A statement of the claimed security properties and expected security level is made in section 4. An analysis of the primitive with respect to standard cryptanalytic attacks is provided in section 5 (a statement that there are no hidden weaknesses inserted by the designers is explicitly made in section 5.10). Section 6 contains the design rationale explaining design choices. Implementation guidelines to avoid implementation weaknesses are given in section 7. Estimates of the computational efficiency in software are provided in section 8. The overall strengths and advantages of the primitive are listed in section 9.

2 Mathematical preliminaries and notation

We now summarise the mathematical background and notation that will be used throughout this paper.

2.1 Finite fields

The finite field $\mathrm{GF}(2^8)$ will be represented as $\mathrm{GF}(2)[x]/p(x)$, where $p(x) = x^8 + x^4 + x^3 + x^2 + 1$ is the first primitive polynomial of degree 8 listed in [20]. The polynomial $p(x)$ was chosen so that $g(x) = x$ is a generator of $\mathrm{GF}(2^8) \setminus \{0\}$.

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 (hexadecimal digits enclosed in quotes). For instance, the polynomial $u = x^4 + x + 1$ will be represented by the hexadecimal byte value '13'. By extension, the reduction polynomial $p(x)$ may be written '11d'.

2.2 Matrix classes

$\mathcal{M}_{m \times n}[\mathrm{GF}(2^8)]$ denotes the set of $m \times n$ matrices over $\mathrm{GF}(2^8)$.

$\mathrm{vdm}_n(a_0, a_1, \ldots, a_{m-1})$ denotes the $m \times n$ Vandermonde matrix [11] whose second column consists of elements $a_0, a_1, \ldots, a_{m-1}$, i.e.

$$\mathrm{vdm}_n(a_0, a_1, \ldots, a_{m-1}) \equiv \begin{bmatrix} 1 & a_0 & a_0^2 & \ldots & a_0^{n-1} \\ 1 & a_1 & a_1^2 & \ldots & a_1^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & a_{m-1} & a_{m-1}^2 & \ldots & a_{m-1}^{n-1} \end{bmatrix}.$$

We write simply $\mathrm{vdm}(a_0, \ldots, a_{m-1})$ where the number of columns is not important for the discussion, or clear from the context.

If m is a power of 2, $\mathrm{had}(a_0, \ldots, a_{m-1})$ denotes the $m \times m$ Hadamard matrix [1] with elements $h_{ij} = a_{i \oplus j}$.

2.3 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 [22].

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 $G = [I\ A]$ 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. [22], chapter 11, § 4, theorem 8).

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 $l_\alpha : \mathrm{GF}(2)^n \to GF(2)$ the linear Boolean function consisting of the sum of the argument bits selected by the bits of α:

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

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:

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

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

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

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

$$\delta_S = \frac{1}{e_S(0,0)} \cdot \max_{a \neq 0,\, b} e_S(a,b).$$

The product $\delta \cdot e_S(0,0)$ is called the *differential uniformity* of S.

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

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

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:

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

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

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

Given an $[k+m, k, d]$ linear code over $\mathrm{GF}(2^p)$ with generator matrix $G = [I_{k \times k}\, M_{k \times m}]$, the linear mapping $\theta : \mathrm{GF}(2^p)^k \to \mathrm{GF}(2^p)^m$ defined by

$$\theta(a) = a \cdot M$$

has branch number $\mathcal{B}(\theta) = d$; if the code is MDS, such a mapping is called an *optimal diffusion mapping* [25].

2.5 Miscellaneous notation

Given a sequence of functions $f_m, f_{m+1}, \ldots, f_{n-1}, f_n$, $m \leqslant n$, we use the notation $\bigcirc_{r=m}^{n} f_r \equiv f_m \circ f_{m+1} \circ \cdots \circ f_{n-1} \circ f_n$, and $\bigcirc_{m}^{r=n} f_r \equiv f_n \circ f_{n-1} \circ \cdots \circ f_{m+1} \circ f_m$; if $m > n$, both expressions stand for the identity mapping.

3 Description of the ANUBIS primitive

The ANUBIS cipher is an iterated 'involutional'[1] block cipher that operates on a 128-bit *cipher state*. It uses a variable-length, $32N$-bit *cipher key* ($4 \leqslant N \leqslant 10$), and consists of a series of applications of a key-dependent round transformation to the cipher state.

In the following we will individually define the component mappings and constants that build up ANUBIS, then specify the complete cipher in terms of these components. The definitions are often parameterised by N, as many components are used both in the round structure and in the key schedule.

3.1 Input and output

The cipher state is internally viewed as a matrix in $\mathcal{M}_{4\times 4}[\mathrm{GF}(2^8)]$, and the cipher key as a matrix in $\mathcal{M}_{N\times 4}[\mathrm{GF}(2^8)]$. Therefore, 128-bit data blocks and $32N$-bit cipher keys (externally represented as byte arrays) must be mapped to and from the internal matrix format. This is done by function $\mu : \mathrm{GF}(2^8)^{4N} \rightarrow \mathcal{M}_{N\times 4}[\mathrm{GF}(2^8)]$ and its inverse:

$$\mu(a) = b \Leftrightarrow b_{ij} = a_{4i+j},\ 0 \leqslant i \leqslant N-1,\ 0 \leqslant j \leqslant 3.$$

3.2 The nonlinear layer γ

Function $\gamma : \mathcal{M}_{N\times 4}[\mathrm{GF}(2^8)] \rightarrow \mathcal{M}_{N\times 4}[\mathrm{GF}(2^8)]$, $4 \leqslant N \leqslant 10$, consists of the parallel application of a nonlinear substitution box $S : \mathrm{GF}(2^8) \rightarrow \mathrm{GF}(2^8)$, $x \mapsto S[x]$ to all bytes of the argument individually:

$$\gamma(a) = b \Leftrightarrow b_{ij} = S[a_{ij}],\ 0 \leqslant i \leqslant N-1,\ 0 \leqslant j \leqslant 3.$$

The substitution box was pseudo-randomly chosen and is listed in appendix B. The search criteria are described in section 6.2; one of them imposes that S be an involution, i.e. $S[S[x]] = x$ for all $x \in \mathrm{GF}(2^8)$. Therefore, γ itself is an involution.

3.3 The transposition τ

Mapping $\tau : \mathcal{M}_{4\times 4}[\mathrm{GF}(2^8)] \rightarrow \mathcal{M}_{4\times 4}[\mathrm{GF}(2^8)]$ simply transposes its argument:

$$\tau(a) = b \Leftrightarrow b = a^t \Leftrightarrow b_{ij} = a_{ji},\ 0 \leqslant i, j \leqslant 3.$$

Clearly τ is an involution.

[1] We explain in section 3.11 what we mean by an 'involutional' block cipher.

3.4 The linear diffusion layer θ

The diffusion layer $\theta : \mathcal{M}_{N\times 4}[\mathrm{GF}(2^8)] \to \mathcal{M}_{N\times 4}[\mathrm{GF}(2^8)]$, $4 \leqslant N \leqslant 10$, is a linear mapping based on the $[8, 4, 5]$ MDS code with generator matrix $G_H = [I\, H]$ where $H = \mathrm{had}(\text{'01'}, \text{'02'}, \text{'04'}, \text{'06'})$, i.e.

$$H = \begin{bmatrix} \text{'01'} & \text{'02'} & \text{'04'} & \text{'06'} \\ \text{'02'} & \text{'01'} & \text{'06'} & \text{'04'} \\ \text{'04'} & \text{'06'} & \text{'01'} & \text{'02'} \\ \text{'06'} & \text{'04'} & \text{'02'} & \text{'01'} \end{bmatrix},$$

so that

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

A simple inspection shows that matrix H is symmetric and unitary. Therefore, θ is an involution for $N = 4$.

3.5 The key addition $\sigma[k]$

The affine key addition $\sigma[k] : \mathcal{M}_{N\times 4}[\mathrm{GF}(2^8)] \to \mathcal{M}_{N\times 4}[\mathrm{GF}(2^8)]$, $4 \leqslant N \leqslant 10$, consists of the bitwise addition (exor) of a key matrix $k \in \mathcal{M}_{N\times 4}[\mathrm{GF}(2^8)]$:

$$\sigma[k](a) = b \Leftrightarrow b_{ij} = a_{ij} \oplus k_{ij},\ 0 \leqslant i \leqslant N-1,\ 0 \leqslant j \leqslant 3.$$

This mapping is also used to introduce round constants in the key schedule, and is obviously an involution.

3.6 The cyclical permutation π

Permutation $\pi : \mathcal{M}_{N\times 4}[\mathrm{GF}(2^8)] \to \mathcal{M}_{N\times 4}[\mathrm{GF}(2^8)]$, $4 \leqslant N \leqslant 10$, cyclically shifts each column of its argument independently, so that column j is shifted downwards by j positions:

$$\pi(a) = b \Leftrightarrow b_{ij} = a_{(i-j) \bmod N, j},\ 0 \leqslant i \leqslant N-1,\ 0 \leqslant j \leqslant 3.$$

3.7 The key extraction ω

The key extraction function $\omega : \mathcal{M}_{N\times 4}[\mathrm{GF}(2^8)] \to \mathcal{M}_{4\times 4}[\mathrm{GF}(2^8)]$, $4 \leqslant N \leqslant 10$, is a linear mapping based on the $[N+4, N, 5]$ MDS code with generator matrix $G_V = [I\, V^t]$, where $V = \mathrm{vdm}_N(\text{'01'}, \text{'02'}, \text{'06'}, \text{'08'})$, i.e.

$$V = \begin{bmatrix} \text{'01'} & \text{'01'} & \text{'01'} & \dots & \text{'01'} \\ \text{'01'} & \text{'02'} & \text{'02'}^2 & \dots & \text{'02'}^{N-1} \\ \text{'01'} & \text{'06'} & \text{'06'}^2 & \dots & \text{'06'}^{N-1} \\ \text{'01'} & \text{'08'} & \text{'08'}^2 & \dots & \text{'08'}^{N-1} \end{bmatrix},$$

so that

$$\omega(a) = b \Leftrightarrow b = V \cdot a.$$

3.8 The round constants c^r

The r-th round constant ($r > 0$) is a matrix $c^r \in \mathcal{M}_{N\times 4}[\mathrm{GF}(2^8)]$, $4 \leqslant N \leqslant 10$, defined as:

$$\begin{aligned} c^r_{0j} &= S[4(r-1)+j],\ 0 \leqslant j \leqslant 3, \\ c^r_{ij} &= 0, \qquad\qquad\qquad 1 \leqslant i < N, 0 \leqslant j \leqslant 3. \end{aligned}$$

3.9 The key schedule

The key schedule expands the cipher key $K \in \mathrm{GF}(2^8)^{4N}$, $4 \leqslant N \leqslant 10$, onto a sequence of round keys $K^0, \ldots, K^R$, with $K^r \in \mathcal{M}_{4\times 4}[\mathrm{GF}(2^8)]$:

$$\begin{aligned} \kappa^0 &= \mu(K), \\ \kappa^r &= (\sigma[c^r] \circ \theta \circ \pi \circ \gamma)(\kappa^{r-1}),\ \ r > 0, \\ K^r &= (\tau \circ \omega \circ \gamma)(\kappa^r),\ \ 0 \leqslant r \leqslant R; \end{aligned}$$

the composite mappings $\psi[c^r] \equiv \sigma[c^r] \circ \theta \circ \pi \circ \gamma$ and $\phi \equiv \tau \circ \omega \circ \gamma$ are called, respectively, the r-th round *key evolution function* and the *key selection function.* The initial γ applied to compute K^0 plays no cryptographic role and is only kept for simplicity.

3.10 The complete cipher

Anubis is defined for the cipher key $K \in \mathrm{GF}(2^8)^{4N}$ as the transformation $\textsc{Anubis}[K] : \mathrm{GF}(2^8)^{16} \to \mathrm{GF}(2^8)^{16}$ given by

$$\textsc{Anubis}[K] \equiv \mu^{-1} \circ \alpha_R[K^0, \ldots, K^R] \circ \mu,$$

where

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

The standard number of rounds R is defined as $R = 8 + N$ for $32N$-bit keys, $4 \leqslant N \leqslant 10$. The composite mapping $\rho[K^r] \equiv \sigma[K^r] \circ \theta \circ \tau \circ \gamma$ is called the *round function* (for the r-th round), and the related mapping $\rho'[K^R] \equiv \sigma[K^r] \circ \tau \circ \gamma$ is called the *last round function.*

3.11 The inverse cipher

We now show that Anubis is an involutional cipher, in the sense that the only difference between the cipher and its inverse is in the key schedule. We will need the following lemmas:

Lemma 1. $\tau \circ \gamma = \gamma \circ \tau$.

Proof. This follows from the fact that τ only transposes its argument without mixing elements, and γ only operates on individual elements, independently of their coordinates. □

Lemma 2. $\theta \circ \sigma[K^r] = \sigma[\theta(K^r)] \circ \theta$.

Proof. It suffices to notice that $(\theta \circ \sigma[K^r])(a) = \theta(K^r \oplus a) = \theta(K^r) \oplus \theta(a) = (\sigma[\theta(K^r)] \circ \theta)(a)$, for any $a \in \mathcal{M}_{N\times 4}[\mathrm{GF}(2^8)]$. □

We are now ready to state the main property of the inverse transform $\alpha_R^{-1}[K^0, \ldots, K^R]$:

Theorem 1. *Let* $\bar{K}^0 \equiv K^R$, $\bar{K}^R \equiv K^0$, *and* $\bar{K}^r \equiv \theta(K^{R-r})$, $0 < r < R$. *Then* $\alpha_R^{-1}[K^0, \ldots, K^R] = \alpha_R[\bar{K}^0, \ldots, \bar{K}^R]$.

Proof. We start from the definition of $\alpha_R[K^0, \ldots, K^R]$:

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

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

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

The two above lemmas lead to:

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

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

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

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

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

That is, $\alpha_R^{-1}[K^0, \ldots, K^R] = \alpha_R[\bar{K}^0, \ldots, \bar{K}^R]$. □

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

4 Security goals

In this section, we present the goals we have set for the security of Anubis. A cryptanalytic attack will be considered successful by the designers if it demonstrates that a security goal described herein does not hold.

In order to formulate our goals, some security-related concepts need to be defined.

4.1 The set of ciphers for given block length and key length

A block cipher of block length v has $V = 2^v$ possible inputs. If the key length is u it defines a set of $U = 2^u$ permutations over $\{0,1\}^v$. The number of possible permutations over $\{0,1\}^v$ is $V!$. Hence the number of all possible block ciphers of dimensions u and v is

$$((2^v)!)^{(2^u)} = (V!)^U.$$

For practical values of the dimensions (e.g. v and u above 40), the subset of block ciphers with exploitable weaknesses form a negligible minority in this set.

4.2 K-Security

Definition 1 ([4]). *A block cipher is* K-secure *if all possible attack strategies for it have the same expected work factor and storage requirements as for the majority of possible block ciphers with the same dimensions. This must be the case for all possible modes of access for the adversary (known/chosen/adaptively chosen plaintext/ciphertext, known/chosen/adaptively chosen key relations ...) and for any a priori key distribution.*

K-security is a very strong notion of security. It can easily be seen that if one of the following weaknesses apply to a cipher, it cannot be called K-secure:

- Existence of key-recovering attacks faster than exhaustive search;
- Certain symmetry properties in the mapping (e.g. any complementation property);
- Occurrence of non-negligible classes of weak keys;
- Related-key attacks.

K-security is essentially a relative measure. It is quite possible to build a K-secure block cipher with a 5-bit block and key length. The lack of security offered by such a scheme is due to its small dimensions, not to the fact that the scheme fails to meet the requirements imposed by these dimensions. Clearly, the longer the key, the higher the security requirements.

4.3 Hermetic block ciphers

It is possible to imagine ciphers that have certain weaknesses and still are K-secure. An example of such a weakness would be a block cipher with a block length larger than the key length and a single weak key, for which the cipher mapping is linear. The detection of the usage of the key would take at least a few encryptions, while checking whether the key is used would only take a single encryption.

If this cipher would be used for encipherment, this single weak key would pose no problem. However, used as a component in a larger scheme, for instance as the compression function of a hash function, this property could introduce a way to efficiently generate collisions.

For these reasons we introduce yet another security concept, denoted by the term *hermetic.*

Definition 2 ([4]). *A block cipher is* hermetic *if it does not have weaknesses that are not present for the majority of block ciphers with the same block and key length.*

Informally, a block cipher is hermetic if its internal structure cannot be exploited in any attack.

4.4 Goal

For all allowed key lengths, the security goals are that the ANUBIS cipher is:

- K-secure;
- Hermetic.

If ANUBIS lives up to its goals, the strength against any known or unknown attacks is as good as can be expected from a block cipher with the given dimensions.

4.5 Expected strength

ANUBIS 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 (in the sense of being K-secure and hermetic).

This implies among other things, the following. The most efficient key-recovery attack for ANUBIS is exhaustive key search. Obtaining information from given plaintext-ciphertext pairs about other plaintext-ciphertext pairs cannot be done more efficiently than by determining the key by exhaustive key search. The expected effort of exhaustive key search depends on the bit length of the cipher key and is 2^{m-1} applications of ANUBIS for m-bit keys.

The rationale for this is that a considerable safety margin is taken with respect to all known attacks. We do however realise that it is impossible to make non-speculative statements on things unknown.

5 Analysis

5.1 Differential and linear cryptanalysis

Due to the Square pattern propagation theorem (cf. [25], proposition 7.9), for any two different input values it holds that the number of S-boxes with a different input value in four consecutive rounds is at least $\mathcal{B}^2 = 25$. As a consequence, no differential characteristic over four rounds has probability larger than $\delta^{\mathcal{B}^2} = (2^{-5})^{25} = 2^{-125}$. Due to the same theorem, no linear approximation over four rounds has input-output correlation larger than $\lambda^{\mathcal{B}^2} = (13 \times 2^{-6})^{25} \approx 2^{-57.5}$. This makes classical differential or linear attacks, as well as some advanced variants like differential-linear attacks, very unlikely to succeed for the full cipher.

5.2 Truncated differentials

The concept of truncated differentials was introduced in [15], and typically applies to ciphers in which all transformations operate on well aligned data blocks. Since in ANUBIS all transformations operate on bytes rather than individual bits, we investigated its resistance against truncated differentials. For 6 rounds or more, no attacks faster than exhaustive key search have been found.

5.3 Interpolation attacks

Interpolation attacks [13] generally depend on the cipher components (particularly the S-box) having simple algebraic structures that can be combined to give expressions with manageable complexity. In such attacks, the attacker constructs polynomials (or rational expressions) using cipher input/output pairs; if these polynomials have small degree, only few cipher input/output pairs are necessary to solve for their (key-dependent) coefficients. The complicated expression of the pseudo-randomly generated S-box in $GF(2^8)$, in combination with the effect of the diffusion layer, makes these types of attack infeasible for more than a few rounds.

5.4 Weak keys

The weak keys discussed in this subsection are keys that result in a block cipher mapping with detectable weaknesses. The best known case of such weak keys are those of IDEA [4]. Typically, this weakness occurs for ciphers in which the nonlinear operations depend on the actual key value. This is not the case for ANUBIS, where keys are applied using exor and all nonlinearity is in the fixed S-box. In ANUBIS, there is no restriction on key selection.

5.5 Related-key cryptanalysis

Related-key attacks generally rely upon slow diffusion and/or symmetry in the key schedule. The ANUBIS key schedule inherits many properties from the round structure itself, and was designed to cause fast, nonlinear diffusion of cipher key differences to the round keys. In particular, the Vandermonde matrix V in the key selection seems very effective in countering all kinds of key based attacks known.

For key lengths that are larger than the length of one round key, it is inevitable that there exist sets of keys that produce identical values for at least one round key. Of special interest for a related key attack seems the possibility to find two different keys with identical values for two consecutive round keys.

The code defined by V has distance 5 and operates separately on the 4 columns of κ^r. This means that two different κ^r-values can produce an equal round key K^r only if for all 4 columns it holds that they differ in either zero or at least 5 bytes. In order to produce equal round keys in two consecutive rounds r and $r+1$, this requirement has to be fulfilled for κ^r and κ^{r+1}. The diffusion layer

θ that is employed in the key evolution function ensures that in total at least 5 columns have to be different in κ^r and κ^{r+1}. We leave it as an open problem to determine whether classes of keys can be determined that produce identical round key values for two or more consecutive rounds. Even so, it is unclear how such keys could possibly be used successfully in a related key attack.

5.6 The Square attack and its variants

In this section we describe an attack first presented in [5], together with its variants [9]. We focus our attention on the α_R transform, as the occurrences of μ and its inverse merely change the data representation. We will denote by a^r the cipher state at the beginning of the r-round (input to γ), and by b^r the cipher state at the output of the σ key addition in the r-round; these quantities may be indexed to select a particular byte. For instance, b^1_{ij} is the byte at position (i, j) of the cipher state at the output of round 1.

The basic 4-round attack: Take a set of 256 plaintexts different from each other in a single byte (which assumes all possible values), the remaining 15 bytes being constant. After two rounds all 16 bytes of each cipher state a^3 in the set will take every value exactly once. After three rounds, the exor of all 256 cipher states a^4 at every byte position will be zero.

Consider a ciphertext $b^4 = \tau(\gamma(a^4)) \oplus K^4$; clearly $a^4 = \gamma(\tau(b^4) \oplus \tau(K^4))$. Now take a byte from $\tau(b^4)$, guess the matching byte from $\tau(K^4)$ and apply γ to the exor of these quantities. Do this for all 256 ciphertexts in the set and check whether the exor of the 256 results indeed equals zero. If it doesn't, the guessed key byte is certainly wrong. A few wrong keys (a fraction about 1/256 of all keys) may pass this test; repeating it for a second set of plaintexts leaves only the correct key value with overwhelming probability.

Adding a round at the end: The 4-round attack can be extended with an extra round at the end. In this case, the round lacking θ is the 5th rather than the 4th. We initially observe that the byte a^4_{ij} depends on the whole row j of b^4, which in turn depends on the whole column j of b^5. To obtain it, first guess column j of K^5 and compute row j of $b^4 = \gamma(\tau(b^5 \oplus K^5))$ (i.e. compute $b^4_{jk} = S[b^5_{kj} \oplus K^5_{kj}]$ for $k = 0, \ldots, 3$). Now take the byte at position (i, j) of $\tau(\theta(b^4))$, guess the matching byte from $\tau(\theta(K^4))$ and compute S on the exor of these quantities, recovering a^4_{ij} and proceeding as in the 4-round attack.

This attack recovers four bytes of the last round key and one byte of the last round key but one. The remaining bytes can be obtained in a variety of ways; in the simplest case the process above could be merely repeated four times, though three applications and a final exhaustive search for the few missing bytes are often enough, and more efficient.

Overall, 2^{40} partial key values must be checked. Since each set of plaintexts leaves about 1/256 of wrong keys, the whole process must be repeated for 5

sets of 256 plaintexts. However, after testing with the first set of 256 plaintexts, only $2^{40} \times 2^{-8} = 2^{32}$ partial key values survive, and only this fraction has to be tested with the second set of plaintexts. Therefore, the first check determines the complexity of the attack. The attack complexity is therefore 2^{40} key guesses $\times 2^8$ plaintexts $= 2^{48}$ S-box lookups.

Adding a round at the beginning: The basic idea is to choose a set of 256 plaintexts such that a single byte of b^1 takes all possible values over the set while the other 15 bytes remain constant, then proceed as with the 5-round attack above. This is achieved by selecting 2^{32} plaintexts with one column taking all 2^{32} values and the remaining 12 bytes constant, then guessing the four bytes of K^0 on the same column, and filtering 256 plaintexts that differ from each other in a single byte on that column.

There is a better way to add a round at the beginning, though. The 2^{32} plaintexts above may be viewed as 2^{24} groups of 256 encryptions that differ from each other in a single byte of b^1. Since the exor of the bytes at any position (i, j) of a^5 over each group is zero, the exor at that position over all 2^{32} plaintexts is also zero. Thus, a^5_{ij} is recovered as in the 5-round attack by guessing column j of K^6 plus one byte at position (i, j) of $\tau(\theta(K^5))$, and the result is exored over the 2^{32} encryptions, checking for zero. This test is somewhat weaker than the test in the 5-round attack and may still leave about $1/256$ of wrong keys; therefore it needs about 6×2^{32} chosen plaintexts. The effort involved is 2^{40} key guesses $\times$ 2^{32} chosen plaintexts $= 2^{72}$ S-box lookups.

The partial sum improvement: This is a dynamic programming technique; it trades computational effort for storage by reorganising the intermediate computations.

Consider the 6-round attack, where we compute the exor of a byte a^5_{ij} over the set of encryptions by guessing five key bytes. Let $k_0, \ldots, k_4$ be the guessed key bytes; we may write

$$a^5_{ij} = S[\bigoplus_{t=0}^{3} S_t[c_t \oplus k_t] \oplus k_4], \tag{1}$$

where the S_t are bijective S-boxes consisting of S followed by a multiplication by an element from the matrix H used in θ, and c_t are the ciphertext bytes on column j. Let $x_k = \bigoplus_{t=0}^{k} S_t[c_t \oplus k_t]$. When computing the exor of the a^5_{ij} bytes, terms with identical values of $x_k, c_{k+1}, \ldots, c_3$ cancel each other. Therefore we speed up the exor computation by first guessing the values of k_0 and k_1 and counting (mod 2) the occurrences of each triple (x_1, c_2, c_3) over the set of ciphertexts, then guessing k_2 and counting (mod 2) the occurrences of each pair (x_2, c_3) over the triples (x_1, c_2, c_3), then guessing k_3 and counting x_3 over the pairs (x_2, c_3), and finally guessing k_4 and computing the sum over the values of x_3.

Notice that we initially guessed 2^{16} pairs (k_0, k_1) and processed 2^{32} ciphertexts; for each pair (k_0, k_1) we guessed 2^8 values of k_2 and processed 2^{24} triples (x_1, c_2, c_3); for each triple (k_0, k_1, k_2) we guessed 2^8 values of k_3 and processed 2^{16} pairs (x_2, c_3); for each quadruple (k_0, k_1, k_2, k_3) we guessed 2^8 values of k_4 and processed 2^8 values of x_3. Overall, this amounts to 2^{48} evaluations of equation 1. This effort must be repeated for each of the 6 sets of encryptions used in the attack, resulting in 6×2^{48} S-box lookups, or about 2^{44} 6-round encryptions. The space requirement is $2^{24} + 2^{16} + 2^8$ bits for the counters.

There is an extension of the partial sum attack against 7 rounds of ANUBIS [9] requiring $2^{128} - 2^{119}$ plaintexts, 2^{64} bits of storage and an effort of about 2^{120} encryptions, and an extension to 8 rounds requiring $2^{128} - 2^{119}$ plaintexts, 2^{104} bits of storage and an effort of about 2^{204} encryptions. Although theoretically interesting, these extensions are really "on the edge": regardless of the cipher being used, an attacker that manages to convince the key owner to encrypt 99.8% of all possible plaintexts under the same key could build a dictionary and decrypt or forge at will an equal fraction of all possible message blocks, or an arbitrary message block with 99.8% probability – without ever knowing the key and without the effort of 2^{120} (let alone 2^{204}) extra encryptions.

5.7 The Gilbert-Minier attack

The previous attack uses the fact that three rounds of ANUBIS can easily be distinguished from a random permutation. The Gilbert-Minier attack [10] is based on a 4-round distinguisher. The attack breaks 7 rounds of ANUBIS with complexity: 2^{32} guesses for one column of the first round key $\times$ 2^{16} c-sets $\times$ 16 encryptions per entry $\times$ 2^{80} entries/table $\times$ 2 tables $= 2^{133}$ encryptions (about 2^{140} S-box lookups), plus 2^{32} chosen plaintexts. There is a speedup for 128-bit keys, making the attack marginally faster than exhaustive search, according to the attack authors.

5.8 A general extension attack

Stefan Lucks [21] presents a general extension of any n-round attack; the result is an attack against $(n+1)$ or more rounds that works for long keys. The idea is simply to guess the whole K^{n+1} round key and proceed with the n-round attack. Each extra round increases the complexity by a factor 2^{128} S-box lookups.

Starting from the 7-round Gilbert-Minier attack, the 8-round extension costs $2^{140} \times 2^{128}$ S-box lookups, or about 2^{261} 8-round encryptions; therefore it is faster than exhaustive key search for ANUBIS keys longer than 256 bits. An extension to 9 rounds against 320-bit keys would require an 8-round attack of complexity 2^{192} encryptions or less, but none is known.

5.9 Other attacks

Attacks based on linear cryptanalysis can sometimes be improved by using nonlinear approximations [16]. 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 ANUBIS.

The impossible differential attack described in [3] can be adapted to work against 5-round ANUBIS with $2^{29.5}$ chosen plaintexts and 2^{31} steps.

The boomerang attack [26] benefits from ciphers whose strength is different for encryption and decryption; this is hardly the case for ANUBIS, due to its involutional structure.

We were not able to find any other method to attack the cipher faster than exhaustive key search.

5.10 Designers' statement on the absence of hidden weaknesses

In spite of any analysis, doubts might remain regarding the presence of trapdoors deliberately introduced in the algorithm. That is why the NESSIE effort asks for the designers' declaration on the contrary.

Therefore we, the designers of ANUBIS, do hereby declare that there are no hidden weaknesses inserted by us in the ANUBIS primitive.

6 Design rationale

6.1 Self-inverse structure

Involutional structure is found as part of many cipher designs. All classical Feistel networks [8] have this property, as do some more general iterated block ciphers like IDEA [23]. Self-inverse ciphers similar to ANUBIS were described and analyzed in [28, 29].

The importance of involutional structure resides not only in the advantages for implementation, but also in the equivalent security of both encryption and decryption [17].

6.2 Choice of the substitution box

The S-box S was pseudo-randomly chosen to satisfy the following conditions:

- S must be an involution, i.e. $S[S[x]] = x$ for all $x \in \mathrm{GF}(2^8)$.
- The δ-parameter must not exceed 8×2^{-8}.
- The λ-parameter must not exceed 16×2^{-6}.
- The nonlinear order ν must be maximum, namely, 7.

The values of δ and λ are constrained to be no more than twice the minimum achievable values. The actual ANUBIS S-box has $\lambda = 13 \times 2^{-6}$; experiences showed that involutions with $\delta < 8 \times 2^{-8}$ and $\lambda < 13 \times 2^{-6}$ are extremely rare, as none was found in a set of over 600 million randomly generated S-boxes.

The following auxiliary conditions were also imposed to speed up the S-box search:

- S must not have any fixed point, i.e. $S[x] \neq x$ for all $x \in \mathrm{GF}(2^8)$.

– The value of any difference $x \oplus S[x]$ must occur exactly twice (hence the set of all difference values consists of exactly 128 elements).

The absence of fixed points is inspired by the empirical study reported in section 2.3 of [28], where the strong correlation found between the cryptographic properties and the number of fixed points of a substitution box suggests minimising the number of such points. In a more general fashion, we empirically found that the fraction of random involutions with good values of δ and λ is increased not only by avoiding fixed points, but also by minimising the number of occurrences of any particular difference $x \oplus S[x]$ (or, equivalently, maximising the number of such differences).

Finally, the polynomial and rational representations of S over GF(2^8) were checked to avoid any obvious algebraic weakness. The random nature of the search tend to make these representations as involved as possible.

6.3 Choice of the diffusion layer

The actual matrix used in the diffusion layer θ was selected by exhaustive search. Although other ciphers of the same family as ANUBIS use circulant matrices for this purpose (cf. [5, 6]), it is not difficult to prove that no such matrix can be self-inverse. On the other hand, unitary Hadamard matrices can be easily computed that satisfy the MDS condition.

The actual choice involves coefficients with the lowest possible Hamming weight (which is advantageous for hardware implementations) and lowest possible integer values (which is important for smart card implementations as discussed in section 7.2).

6.4 On the transposition τ and the permutation π

Though equivalent form the security viewpoint, τ and π have quite different structural properties. The round function needs an involution to disperse bytes, hence π is not suitable for this task. The key schedule needs a mapping that keeps the key state dimensions ($N \times 4$) unchanged, hence τ is not suitable in general. Therefore both τ and π are needed in different contexts; for their appointed purposes, they are perhaps the simplest choices.

6.5 Structure of the key schedule

The key scheduling algorithm of ANUBIS is based on ideas from [25]. The division into separate layers for diffusion and confusion is directly taken from the round structure itself, which ultimately follows the Wide Trail strategy. The goal is to thwart attacks based on straightforward exploitation of relationships between round keys. The present choice also favours component reuse.

6.6 Choice of the key extraction function

The Vandermonde structure of the extraction function ω has several interesting properties.

First, it is not difficult to obtain MDS Vandermonde matrices by random search if the number of rows is not too large. The reason is simple: many submatrices of a Vandermonde matrix V are themselves Vandermonde matrices (e.g. submatrices built from the even columns of V), while the determinant of many other submatrices divides the determinant of some Vandermonde matrix (e.g. submatrices built from consecutive columns of V). Thus, merely choosing distinct nonzero elements on the second column of V ensures that many submatrices are nonsingular, and other simple tests speed up the search even more.

Second, it is possible to choose Vandermonde matrices with important properties for the key schedule. Thus, it was required that $\theta \circ \tau \circ \omega$, which is the effective key extraction function in the inverse cipher, satisfy the same conditions as ω itself; therefore, all square submatrices of $H \cdot V$ are nonsingular. We remark *en passant* that the occurrence of τ is motivated by the simple condition it implies on the product $H \cdot V$ (if τ were absent, these matrices would effectively multiply the key state at opposite sides). The restriction to 320-bit keys comes from the fact that this condition is no longer satisfied by extensions of V with more than 10 columns for the actually chosen V coefficients.

Third, multiplication by a Vandermonde matrix can be done quite efficiently with a fast polynomial evaluation algorithm, as seen in section 7.1. The ease to find suitable matrices provides freedom in the choice of coefficients, which can make the polynomial evaluation even more efficient. The actual coefficients were chosen to have the lowest possible Hamming weight (which is advantageous for hardware implementations) while satisfying the condition on $\theta \circ \tau \circ \omega$ as explained above.

6.7 Choice of the round constants

Good round 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. Making only one row of c^r different from zero makes the round constants simpler. The actual choice meets these constraints while also reusing an available component (the S-box itself).

7 Implementation

ANUBIS can be implemented very efficiently. On different platforms, different optimisations and tradeoffs are possible. We make here a few suggestions.

7.1 32-bit and 64-bit processors

Implementation of ρ: We suggest the following lookup-table approach.

$$
\begin{array}{l}
b = (\theta \circ \tau \circ \gamma)(a) \\
\Updownarrow \\
\begin{bmatrix} b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} \\ b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,0} & b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,0} & b_{3,1} & b_{3,2} & b_{3,3} \end{bmatrix} = \begin{bmatrix} S[a_{0,0}] & S[a_{1,0}] & S[a_{2,0}] & S[a_{3,0}] \\ S[a_{0,1}] & S[a_{1,1}] & S[a_{2,1}] & S[a_{3,1}] \\ S[a_{0,2}] & S[a_{1,2}] & S[a_{2,2}] & S[a_{3,2}] \\ S[a_{0,3}] & S[a_{1,3}] & S[a_{2,3}] & S[a_{3,3}] \end{bmatrix} \cdot \begin{bmatrix} \text{'01'} & \text{'02'} & \text{'04'} & \text{'06'} \\ \text{'02'} & \text{'01'} & \text{'06'} & \text{'04'} \\ \text{'04'} & \text{'06'} & \text{'01'} & \text{'02'} \\ \text{'06'} & \text{'04'} & \text{'02'} & \text{'01'} \end{bmatrix} \\
\Updownarrow \\
\begin{array}{l} \left[b_{k,0} \; b_{k,1} \; b_{k,2} \; b_{k,3} \right] = S[a_{0,k}] \cdot [\text{'01'} \; \text{'02'} \; \text{'04'} \; \text{'06'}] \oplus S[a_{1,k}] \cdot [\text{'02'} \; \text{'01'} \; \text{'06'} \; \text{'04'}] \\ \qquad \oplus S[a_{2,k}] \cdot [\text{'04'} \; \text{'06'} \; \text{'01'} \; \text{'02'}] \oplus S[a_{3,k}] \cdot [\text{'06'} \; \text{'04'} \; \text{'02'} \; \text{'01'}], \\ \qquad 0 \leqslant k \leqslant 3. \end{array}
\end{array}
$$

Using the following four tables:

$$
\begin{array}{ll}
T_0[x] = S[x] \cdot [\text{'01'} \; \text{'02'} \; \text{'04'} \; \text{'06'}], & T_1[x] = S[x] \cdot [\text{'02'} \; \text{'01'} \; \text{'06'} \; \text{'04'}], \\
T_2[x] = S[x] \cdot [\text{'04'} \; \text{'06'} \; \text{'01'} \; \text{'02'}], & T_3[x] = S[x] \cdot [\text{'06'} \; \text{'04'} \; \text{'02'} \; \text{'01'}],
\end{array}
$$

a row of b can be calculated with four table lookups and three exor operations; the key addition then completes the evaluation of ρ. The T-tables require 4×2^8 bytes of storage each. An implementation can use the fact that the corresponding entries of different T-tables are permutations of one another and save some memory at the expense of introducing extra permutations at runtime. Usually this decreases the performance of the implementation.

Implementation of ϕ: By definition, $K^r = (\tau \circ \omega \circ \gamma)(\kappa^r) = (V \cdot \gamma(\kappa^r))^t = \gamma(\kappa^r)^t \cdot V^t$, so that $K^r_{ij} = \bigoplus_{k=0}^{N-1} S[\kappa^r_{ki}] \cdot a^k_j$, which can be computed by the following algorithm:

set $K^r_{ij} = S[\kappa^r_{N-1,\,i}]$;
for $(k = N-2;\; k \geqslant 0;\; k{-}{-})$ {
    set $K^r_{ij} = S[\kappa^r_{ki}] \oplus a_j \cdot K^r_{ij}$;
}

The columns of K^r_{ij} can be computed in parallel:

set $[K^r_{i0} \; K^r_{i1} \; K^r_{i2} \; K^r_{i3}] = S[\kappa^r_{N-1,\,i}] \cdot [\text{'01'} \; \text{'01'} \; \text{'01'} \; \text{'01'}]$;
for $(k = N-2;\; k \geqslant 0;\; k{-}{-})$ {
    set $[K^r_{i0} \; K^r_{i1} \; K^r_{i2} \; K^r_{i3}] = S[\kappa^r_{ki}] \cdot [\text{'01'} \; \text{'01'} \; \text{'01'} \; \text{'01'}]$
        $\oplus \; K^r_{i,0} \cdot [\text{'01'} \; \text{'00'} \; \text{'00'} \; \text{'00'}]$
        $\oplus \; K^r_{i,1} \cdot [\text{'00'} \; \text{'02'} \; \text{'00'} \; \text{'00'}]$
        $\oplus \; K^r_{i,2} \cdot [\text{'00'} \; \text{'00'} \; \text{'06'} \; \text{'00'}]$
        $\oplus \; K^r_{i,3} \cdot [\text{'00'} \; \text{'00'} \; \text{'00'} \; \text{'08'}]$;
}

Using the following two tables:

$$T_4[x] = S[x] \cdot [\text{'01'}\ \text{'01'}\ \text{'01'}\ \text{'01'}], \quad T_5[x] = x \cdot [\text{'01'}\ \text{'02'}\ \text{'06'}\ \text{'08'}],$$

a row of K_{ij}^r can be calculated with $1 + 5(N-1)$ table lookups, $4(N-1)$ bitwise 'and' operations (to properly mask the results of the T_5 lookups), and $4(N-1)$ exor operations.

Alternatively, if enough storage (including processor built-in cache) is available for the following N tables:

$$U_k[x] = S[x] \cdot [\text{'01'}\ \text{'02'}^k\ \text{'06'}^k\ \text{'08'}^k],\ 0 \leqslant k \leqslant N-1,$$

then each row K^r can be computed as $[K_{i0}^r\ K_{i1}^r\ K_{i2}^r\ K_{i3}^r] = \bigoplus_{k=0}^{N-1} U_k[\kappa_{ki}^r]$ with only N table lookups and $N-1$ exor operations.

Implementation of ψ: The same lookup-table approach suggested for ρ can be employed for ψ, reusing the same T-tables:

$$\begin{aligned}
b &= (\theta \circ \pi \circ \gamma)(a) \\
&\Updownarrow \\
[b_{k,0}\ b_{k,1}\ b_{k,2}\ b_{k,3}] &= S[a_{k,0}] \cdot [\text{'01'}\ \text{'02'}\ \text{'04'}\ \text{'06'}] \\
&\oplus S[a_{(k-1) \bmod N,1}] \cdot [\text{'02'}\ \text{'01'}\ \text{'06'}\ \text{'04'}] \\
&\oplus S[a_{(k-2) \bmod N,2}] \cdot [\text{'04'}\ \text{'06'}\ \text{'01'}\ \text{'02'}] \\
&\oplus S[a_{(k-3) \bmod N,3}] \cdot [\text{'06'}\ \text{'04'}\ \text{'02'}\ \text{'01'}], \\
&0 \leqslant k \leqslant 3, \\
&\Updownarrow \\
[b_{k,0}\ b_{k,1}\ b_{k,2}\ b_{k,3}] &= T_0[a_{k,0}] \\
&\oplus T_1[a_{(k-1) \bmod N,1}] \\
&\oplus T_2[a_{(k-2) \bmod N,2}] \\
&\oplus T_3[a_{(k-3) \bmod N,3}], \\
&0 \leqslant k \leqslant 3.
\end{aligned}$$

7.2 8-bit processors

On an 8-bit processor with a limited amount of RAM, e.g. a typical smart card processor, the previous approach is not feasible. On these processors the substitution is performed byte by byte, combined with the τ and the $\sigma[K^r]$ transformation. For θ, it is necessary to implement the matrix multiplication.

The following piece of pseudo-code calculates one row of $b = \theta(a)$, using a table X that implements multiplication by the polynomial $g(x) = x$ in GF(2^8), i.e. $X[u] = x \cdot u$, and four registers r_0, r_1, r_2, r_3:

$r_0 = X[a_{i1} \oplus a_{i3}];$
$r_1 = X[a_{i0} \oplus a_{i2}];$
$r_2 = X[X[a_{i2} \oplus a_{i3}]];$
$r_3 = X[X[a_{i0} \oplus a_{i1}]];$
$b_{i0} = a_{i0} \oplus r_0 \oplus r_2;$
$b_{i1} = a_{i1} \oplus r_1 \oplus r_2;$
$b_{i2} = a_{i2} \oplus r_0 \oplus r_3;$
$b_{i3} = a_{i3} \oplus r_1 \oplus r_3;$

This implementation requires 12 exors, 6 table lookups and 8 assignments per row. Notice that, if an additional table $X2$ is available, where $X2[u] \equiv X[X[u]]$, the number of table lookups drops to 4.

The same X table can be used to implement multiplication by '06' and '08'. The following algorithm computes one row of $K^r = \phi(\kappa^r)$ using auxiliary registers u, v, and k:

set $K^r_{i0} = K^r_{i1} = K^r_{i2} = K^r_{i3} = S[\kappa^r_{N-1,\,i}];$
for $(k = N - 2;\ k \geqslant 0;\ k{-}{-})$ {
    set $u = S[\kappa^r_{k\,i}];$
    set $v = X[K^r_{i2}];$
    set $K^r_{i0} = u \oplus K^r_{i0};$
    set $K^r_{i1} = u \oplus X[K^r_{i1}];$
    set $K^r_{i2} = u \oplus X[v] \oplus v;$
    set $K^r_{i3} = u \oplus X[X[X[K^r_{i3}]]];$
}

This implementation requires $5(N-1)$ exors, $1 + 7(N-1)$ table lookups and $4 + 6(N-1)$ assignments. With the extra $X2$ table the number of table lookups drops to $1 + 6(N-1)$ and the number of assignments to $4 + 5(N-1)$.

7.3 Techniques to avoid software implementation weaknesses

The attacks of Kocher *et al.* [18, 19] have raised the awareness that careless implementation of cryptographic primitives can be exploited to recover key material. In order to counter this type of attacks, attention has to be given to the implementation of the round transformation as well as the key scheduling of the primitive.

A first example is the *timing attack* [18] that can be applicable if the execution time of the primitive depends on the value of the key and the plaintext. This is typically caused by the presence of conditional execution paths. For instance, multiplication by a constant value over a finite field is sometimes implemented as a multiplication followed by a reduction, the latter being implemented as a conditional exor. This vulnerability is avoided by implementing the multiplication by a constant by means of table lookups, as proposed in sections 7.1 and 7.2.

A second class of attacks are the attacks based on the careful observation of the power consumption pattern of an encryption device [19]. Protection against this type of attack can only be achieved by combined measures at the hardware

and software level. We leave the final word on this issue to the specialists, but we hope that the simple structure and the limited number of operations in ANUBIS will make it easier to create an implementation that resists this type of attacks.

7.4 Hardware implementation

We have currently no figures on the available performance and required area or gate count of ANUBIS in ASIC or FPGA, nor do we have a description in VHDL. However, we expect that the results on RIJNDAEL [12, 27] will carry over to a large extent.

8 Efficiency estimates

8.1 Key setup

Table 1 lists the observed key setup efficiency on a 550 MHz Pentium III platform. The key setup is implemented with two tables as described in section 7.1. The storage needed for the tables is $2^8 \times 4 \times 2 = 2048$ bytes.

Table 1. Key setup with two auxiliary tables

key size (bits)	cycles (encryption)	cycles (decryption)
128	3352	4527
160	4445	5709
192	6644	8008
224	8129	9576
256	9697	11264
288	11385	12931
320	13475	15169

The increased cost of the decryption key schedule is due to the application of θ to $R-1$ round keys. This is done with the same tables used for encryption and decryption, together with the T_4 table to invert the implicit γ transform. The setup of decryption keys is therefore between 13% and 35% more expensive than the setup of encryption keys.

We point out that the reference implementation used to measure efficiency is not fully optimised. By using dedicated key setup assembler code for each allowed key size (together with the N-table implementation described in section 7.1), we expect a reduction of the cycle counts by a factor of at least 2.

8.2 Encryption and decryption

Since ANUBIS has involutional structure, encryption and decryption are equally efficient for any given number of rounds, which in turn is determined by the key size. Table 2 summarises the observed efficiency on a 550 MHz Pentium III platform. We use the four-table implementation described in section 7.1.

Table 2. Encryption/decryption efficiency

key size (bits)	cycles per byte	cycles per block	Mbit/s
128	36.8	589	119.5
160	39.3	628	112.1
192	41.6	665	105.9
224	43.8	701	100.5
256	46.3	740	95.1
288	48.5	776	90.7
320	50.8	813	86.6

9 Advantages

By design, ANUBIS is much more scalable than most modern ciphers, in the sense of being very fast while avoiding excessive storage space (for both code and tables) and expensive or unusual instructions built in the processor; this makes it suitable for a wide variety of platforms. The same structure also favours extensively parallel execution of the component mappings, and its mathematical simplicity tends to make analysis easier.

9.1 Comparison with other Square ciphers

In this section, we list the most important differences between ANUBIS and two other ciphers developed according to the Wide Trail strategy, namely, SQUARE and RIJNDAEL.

The involutional structure: The fact that all components of ANUBIS are involutions should in principle reduce the code size or area in software, respectively hardware applications that implement both encryption and decryption.

The different S-box: The S-box of ANUBIS is generated in a pseudo-random way. The advantage of this lack of structure is that providing a simple mathematical description seems more difficult. The polynomial expansion of the S-box is certainly more involved. The disadvantages are the suboptimal differential and linear properties, and the more complex hardware implementation.

A more complex key scheduling: The advantage is the improved resistance against key based attacks, in particular the shortcuts for long keys. The disadvantage is the higher cost: slower execution, a reduced key agility, larger code or gate count.

9.2 Extensions

The chosen key selection function is based on matrix $\mathrm{vdm}_N(\text{'01'}, \text{'02'}, \text{'06'}, \text{'08'})$, which is MDS for $N \leqslant 10$. It is possible to extend the key schedule to support keys up to 512 bits (i.e. $N \leqslant 16$) by substituting $\mathrm{vdm}_N(\text{'02'}, \text{'15'}, \text{'1c'}, \text{'3c'})$

for the default ANUBIS matrix. Especially crafted matrices might be chosen for particular key sizes; for instance, $\mathrm{vdm}_4(\texttt{'01'}, \texttt{'02'}, \texttt{'05'}, \texttt{'06'})$ may have some implementation advantages for 128-bit keys.

10 The name

ANUBIS was the Egyptian god of embalming and entombment – hence, by extension, the god of 'encryption'. The name seems therefore suitable for a cipher; besides, this choice makes ANUBIS the only cipher with an associated curse against information privacy invaders `:-)`

References

1. K.G. Beauchamp, "Walsh functions and their applications," Academic Press, 1975.
2. E. Biham, A. Biryukov, A. Shamir, "Cryptanalysis of Skipjack Reduced to 31 Rounds using Impossible Differentials," *Advances in Cryptology, Eurocrypt'99, LNCS 1592*, J. Stern, Ed., Springer-Verlag, 1999, pp. 55–64.
3. E. Biham and N. Keller, "Cryptanalysis of reduced variants of RIJNDAEL," submission to the *Third Advanced Encryption Standard Candidate Conference.*
4. J. Daemen, "Cipher and hash function design strategies based on linear and differential cryptanalysis," *Doctoral Dissertation*, March 1995, K.U.Leuven.
5. J. Daemen, L.R. Knudsen and V. Rijmen, "The block cipher SQUARE," *Fast Software Encryption, LNCS 1267*, E. Biham, Ed., Springer-Verlag, 1997, pp. 149–165.
6. J. Daemen and V. Rijmen, "AES proposal: RIJNDAEL," AES submission, `http://www.nist.gov/aes`.
7. W.V. Davies, "Reading the Past – Egyptian Hieroglyphics," University of California Press/British Museum, 1987.
8. H. Feistel, "Cryptography and computer privacy," *Scientific American*, v. 228, n. 5, 1973, pp. 15–23.
9. N. Ferguson, J. Kelsey, S. Lucks, B. Schneier, M. Stay, D. Wagner, and D. Whiting, "Improved Cryptanalysis of RIJNDAEL," to appear in *Fast Software Encryption'00*, Springer-Verlag.
10. H. Gilbert and M. Minier, "A collision attack on 7 rounds of RIJNDAEL," *Third Advanced Encryption Standard Candidate Conference*, NIST, April 2000, pp. 230–241.
11. K. Hoffman and R. Kunze, "Linear Algebra (2nd ed.)," Prentice Hall, 1971.
12. T. Ichikawa, T. Kasuya, M. Matsui, "Hardware evaluation of the AES finalists," presented at the *Third Advanced Encryption Standard Candidate Conference.*
13. T. Jakobsen and L.R. Knudsen, "The interpolation attack on block ciphers," *Fast Software Encryption, LNCS 1267*, E. Biham, Ed., Springer-Verlag, 1997, pp. 28–40.
14. J. Kelsey, B. Schneier, D. Wagner, C. Hall, "Cryptanalytic Attacks on Pseudorandom Number Generators," *Fast Software Encryption, LNCS 1372*, S. Vaudenay , Ed., Springer-Verlag, 1998, pp. 168–188.
15. L.R. Knudsen, "Truncated and higher order differentials," *Fast Software Encryption, LNCS 1008*, B. Preneel, Ed., Springer-Verlag, 1995, pp. 196–211.
16. L.R. Knudsen, M.J.B. Robshaw, "Non-linear approximations in linear cryptanalysis," *Advances in Cryptology, Eurocrypt'96, LNCS 1070*, U. Maurer, Ed., Springer-Verlag, 1996, pp. 224–236.

17. L.R. Knudsen and D. Wagner, "On The Structure of Skipjack," to appear in *Discrete Applied Mathematics*, special issue on Coding Theory and Cryptology, C. Carlet, Ed.
18. P.C. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems," *Advances in Cryptology, Crypto '96, LNCS 1109*, N. Koblitz, Ed., Springer-Verlag, 1996, pp. 104–113.
19. P. Kocher, J. Jaffe, B. Jun, "Introduction to differential power analysis and related attacks," available from http://www.cryptography.com/dpa/technical/.
20. R. Lidl and H. Niederreiter, "Introduction to finite fields and their applications," Cambridge University Press, 1986.
21. S. Lucks, "Attacking seven rounds of RIJNDAEL under 192-bit and 256-bit keys," *Third Advanced Encryption Standard Candidate Conference*, NIST, April 2000, pp. 215–229.
22. F.J. MacWilliams and N.J.A. Sloane, "The theory of error-correcting codes," *North-Holland Mathematical Library*, vol. 16, 1977.
23. X. Lai, J.L. Massey and S. Murphy, "Markov ciphers and differential cryptanalysis," *Advances in Cryptology, Eurocrypt'91, LNCS 547*, D.W. Davies, Ed., Springer-Verlag, 1991, pp. 17–38.
24. National Institute of Standards and Technology, "FIPS 186-2, Digital Signature Standard (DSS)," January 27, 2000.
25. V. Rijmen, "Cryptanalysis and design of iterated block ciphers," *Doctoral Dissertation*, October 1997, K.U.Leuven.
26. D. Wagner, "The Boomerang Attack," *Fast Software Encryption, LNCS 1636*, L. Knudsen, Ed., Springer-Verlag, 1999, pp. 156–170.
27. B. Weeks, M. Bean, T. Rozylowicz, C. Ficke, "Hardware performance simulations of round 2 AES algorithms," presented at the *Third Advanced Encryption Standard Candidate Conference.*
28. A.M. Youssef, S.E. Tavares, and H.M. Heys, "A New Class of Substitution-Permutation Networks," *Workshop on Selected Areas in Cryptography, SAC'96*, Workshop record, 1996, pp. 132–147.
29. A.M. Youssef, S. Mister, and S.E. Tavares, "On the Design of Linear Transformations for Substitution Permutation Encryption Networks," *Workshop on Selected Areas of Cryptography, SAC'97*, Workshop record, 1997, pp. 40–48.

A Generation of the ANUBIS substitution box

To ensure that the involutional S-box used by ANUBIS is generated verifiably pseudo-randomly, the S-box search algorithm uses a PRNG similar to the one specified for generating DSA [24] parameters[2].

1. The PRNG is keyed by the 20-byte array $k = \text{SHA-1}(m)$, where m is the ASCII-coded string "`An offering which Anubis gives`" (in Middle Kingdom Egyptian, *ḥtp dỉ 'Inpw*, cf. [7], pp. 14 and 46).

[2] It is pointed out in [14] that the DSA PRNG is not suitable for generating pseudo-random bytes that must be kept secret, e.g. for establishing secret keys. This is not the present case, as the PRNG is only used to pseudo-randomly generate the ANUBIS involutional S-box in a verifiable fashion.

2. The PRNG maintains two 20-byte arrays, a state $s^{(i)}$ and a buffer $b^{(i)}$, both initialised to zero. The value of $b^{(i)}$ is computed from the key k and the state $x^{(i)}$. Pseudo-random bytes are taken from $b^{(i)}$ (for implementation convenience, this is done from right to left, i.e. from index 19 down to index 0). At any moment, the PRNG keeps track of the number of remaining bytes on $b^{(i)}$ (initially zero).
3. When the number of remaining available bytes on $b^{(i)}$ drops to zero, $b^{(i)}$ and $x^{(i)}$ are updated as follows:
   (a) $b^{(i+1)} = \text{SHA-1}((k + s^{(i)}) \bmod 2^{160})$.
   (b) $s^{(i+1)} = (s^{(i)} + b^{(i)} + 1) \bmod 2^{160}$.
   where the the arrays represent 160-bit integers in little-endian order (i.e. the value associated to $x^{(i)}$ is $\sum_{t=0}^{19} 256^t \cdot x^{(i)}[t]$).

The ANUBIS S-box is pseudo-randomly generated according to the following algorithm, which makes use of a stream of pseudo-random bytes produced by the above PRNG:

1. Generate a list p containing a random permutation of all 255 nonzero byte values (p is the *difference list*).
2. Mark all entries of S as initially undefined.
3. For each preimage x of the S-box being constructed do the following:
   (a) Sequentially extract from p the first unused difference d such that entry $S[x \oplus d]$ is undefined; if none is found, start over with a new random permutation p.
   (b) Let $y = x \oplus d$: set $S[x] = y$ and $S[y] = x$ (thus implicitly marking entry $S[x]$ and $S[y]$ as defined and the difference d as used).
   (c) Replace the gap left by d in p by the last element of p (this step is done this way merely to simplify the handling of p, which may be implemented as an array of bytes).

At the end of this algorithm, S contains a pseudo-random involution over $\text{GF}(2^8)$ in such a way that, for all $x \in \text{GF}(2^8)$, $S[x] \neq x$, and the difference value $S[x] \oplus x$ occurs exactly twice over the table. Now test S for $\delta \leqslant 8$, $\lambda \leqslant 16$, and $\nu = 7$; if the test fails, start over with a new random permutation p. Repeat the whole process a number of times (over 600 million in our actual run) and select the generated S box with the smallest values of δ and λ. A final inspection of the polynomial and rational expansions of S is conducted to check for any obvious weakness (quite unlikely for pseudo-randomly generated S-boxes).

B The substitution box

```
const byte sbox[256] = {
0xa7, 0xd3, 0xe6, 0x71, 0xd0, 0xac, 0x4d, 0x79,
0x3a, 0xc9, 0x91, 0xfc, 0x1e, 0x47, 0x54, 0xbd,
0x8c, 0xa5, 0x7a, 0xfb, 0x63, 0xb8, 0xdd, 0xd4,
0xe5, 0xb3, 0xc5, 0xbe, 0xa9, 0x88, 0x0c, 0xa2,
```

```
0x39, 0xdf, 0x29, 0xda, 0x2b, 0xa8, 0xcb, 0x4c,
0x4b, 0x22, 0xaa, 0x24, 0x41, 0x70, 0xa6, 0xf9,
0x5a, 0xe2, 0xb0, 0x36, 0x7d, 0xe4, 0x33, 0xff,
0x60, 0x20, 0x08, 0x8b, 0x5e, 0xab, 0x7f, 0x78,
0x7c, 0x2c, 0x57, 0xd2, 0xdc, 0x6d, 0x7e, 0x0d,
0x53, 0x94, 0xc3, 0x28, 0x27, 0x06, 0x5f, 0xad,
0x67, 0x5c, 0x55, 0x48, 0x0e, 0x52, 0xea, 0x42,
0x5b, 0x5d, 0x30, 0x58, 0x51, 0x59, 0x3c, 0x4e,
0x38, 0x8a, 0x72, 0x14, 0xe7, 0xc6, 0xde, 0x50,
0x8e, 0x92, 0xd1, 0x77, 0x93, 0x45, 0x9a, 0xce,
0x2d, 0x03, 0x62, 0xb6, 0xb9, 0xbf, 0x96, 0x6b,
0x3f, 0x07, 0x12, 0xae, 0x40, 0x34, 0x46, 0x3e,
0xdb, 0xcf, 0xec, 0xcc, 0xc1, 0xa1, 0xc0, 0xd6,
0x1d, 0xf4, 0x61, 0x3b, 0x10, 0xd8, 0x68, 0xa0,
0xb1, 0x0a, 0x69, 0x6c, 0x49, 0xfa, 0x76, 0xc4,
0x9e, 0x9b, 0x6e, 0x99, 0xc2, 0xb7, 0x98, 0xbc,
0x8f, 0x85, 0x1f, 0xb4, 0xf8, 0x11, 0x2e, 0x00,
0x25, 0x1c, 0x2a, 0x3d, 0x05, 0x4f, 0x7b, 0xb2,
0x32, 0x90, 0xaf, 0x19, 0xa3, 0xf7, 0x73, 0x9d,
0x15, 0x74, 0xee, 0xca, 0x9f, 0x0f, 0x1b, 0x75,
0x86, 0x84, 0x9c, 0x4a, 0x97, 0x1a, 0x65, 0xf6,
0xed, 0x09, 0xbb, 0x26, 0x83, 0xeb, 0x6f, 0x81,
0x04, 0x6a, 0x43, 0x01, 0x17, 0xe1, 0x87, 0xf5,
0x8d, 0xe3, 0x23, 0x80, 0x44, 0x16, 0x66, 0x21,
0xfe, 0xd5, 0x31, 0xd9, 0x35, 0x18, 0x02, 0x64,
0xf2, 0xf1, 0x56, 0xcd, 0x82, 0xc8, 0xba, 0xf0,
0xef, 0xe9, 0xe8, 0xfd, 0x89, 0xd7, 0xc7, 0xb5,
0xa4, 0x2f, 0x95, 0x13, 0x0b, 0xf3, 0xe0, 0x37
};
```