The Anubis Block Cipher

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

[1] Scopus Tecnologia S. A.
Av. Mutinga, 4105 - Pirituba
BR–05110–000 São Paulo (SP), Brazil
pbarreto@scopus.com.br
[2] Cryptomathic NV,
Lei 8A,
B–3000 Leuven, Belgium
vincent.rijmen@cryptomathic.com

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. Anubis has been submitted as a candidate cryptographic primitive for the NESSIE project [19].

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 makes it possible to reduce 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)

As originally submitted for the NESSIE evaluation effort, ANUBIS employed a randomly generated substitution box (S-box) whose lack of internal structure tended to make efficient hardware implementation a challenging and tricky process. In contrast to that version, though, the present document describes an S-box that is much more amenable to hardware implementation, while not adversely affecting any of the software implementation techniques suggested herein. We propose renaming the original algorithm ANUBIS-0 and using the term ANUBIS for the final, modified version that uses the improved S-box design.

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

2.1 Finite fields

We will represent the field $\mathrm{GF}(2^4)$ as $\mathrm{GF}(2)[x]/p_4(x)$ where $p_4(x) = x^4 + x + 1$, and the field $\mathrm{GF}(2^8)$ as $\mathrm{GF}(2)[x]/p_8(x)$ where $p_8(x) = x^8 + x^4 + x^3 + x^2 + 1$. Polynomials $p_4(x)$ and $p_8(x)$ are the first primitive polynomials of degrees 4 and 8 listed in [16], and were chosen so that $g(x) = x$ is a generator of $\mathrm{GF}(2^4) \setminus \{0\}$ and $\mathrm{GF}(2^8) \setminus \{0\}$, respectively.

A polynomial $u = \sum_{i=0}^{m-1} u_i \cdot x^i \in \mathrm{GF}(2)[x]$, where $u_i \in \mathrm{GF}(2)$ for all $i = 0, \ldots, m-1$, will be denoted by the numerical value $\sum_{i=0}^{m-1} u_i \cdot 2^i$, and written in hexadecimal notation. For instance, we write $\mathtt{13}_x$ to denote $p_4(x)$.

2.2 MDS codes

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. [18], chapter 11, § 4, theorem 8).

2.3 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 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:

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

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:

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

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

$$\delta_S \equiv 2^{-n} \cdot \max_{a\neq 0,\, b} \#\{c \in \mathrm{GF}(2^n) | S[c \oplus a] \oplus S[c] = b\}.$$

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

The *correlation* $c(f, g)$ between two Boolean functions f and g is defined as:

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

The extreme value (i.e. either the minimum or the maximum, whichever is larger in absolute value) of the correlation between linear functions of input bits and linear functions of output bits of S is called the *bias* of S.

The λ-*parameter* of an S-box S is defined as the absolute value of the bias:

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

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

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

Given a $[k+m, k, d]$ 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* [21].

2.4 Miscellaneous notation

$\mathcal{M}_{m\times n}[\mathrm{GF}(2^8)]$ 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 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 $h_{ij} = a_{i\oplus j}$.

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.

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

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 S is discussed in detail in section 6.2. One of the design criteria for S imposes that it 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.

3.4 The linear diffusion layer θ

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

$$H = \begin{bmatrix} \mathtt{01}_x & \mathtt{02}_x & \mathtt{04}_x & \mathtt{06}_x \\ \mathtt{02}_x & \mathtt{01}_x & \mathtt{06}_x & \mathtt{04}_x \\ \mathtt{04}_x & \mathtt{06}_x & \mathtt{01}_x & \mathtt{02}_x \\ \mathtt{06}_x & \mathtt{04}_x & \mathtt{02}_x & \mathtt{01}_x \end{bmatrix},$$

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 $\sigma[k] : \mathcal{M}_{N\times 4}[\mathrm{GF}(2^8)] \rightarrow \mathcal{M}_{N\times 4}[\mathrm{GF}(2^8)]$, $4 \leqslant N \leqslant 10$, 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 $\pi : \mathcal{M}_{N\times 4}[\mathrm{GF}(2^8)] \rightarrow \mathcal{M}_{N\times 4}[\mathrm{GF}(2^8)]$, $4 \leqslant N \leqslant 10$, 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 $\omega : \mathcal{M}_{N\times 4}[\mathrm{GF}(2^8)] \rightarrow \mathcal{M}_{4\times 4}[\mathrm{GF}(2^8)]$, $4 \leqslant N \leqslant 10$, is a linear mapping based on the $[N+4, N, 5]$ MDS code with generator matrix $G_V = [I\, V^t]$, where $V = \mathrm{vdm}_N(\mathtt{01}_x, \mathtt{02}_x, \mathtt{06}_x, \mathtt{08}_x)$, i.e.

$$V = \begin{bmatrix} \mathtt{01}_x & \mathtt{01}_x & \mathtt{01}_x & \ldots & \mathtt{01}_x \\ \mathtt{01}_x & \mathtt{02}_x & \mathtt{02}_x^2 & \ldots & \mathtt{02}_x^{N-1} \\ \mathtt{01}_x & \mathtt{06}_x & \mathtt{06}_x^2 & \ldots & \mathtt{06}_x^{N-1} \\ \mathtt{01}_x & \mathtt{08}_x & \mathtt{08}_x^2 & \ldots & \mathtt{08}_x^{N-1} \end{bmatrix},$$

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

$$c^r_{0j} \equiv S[4(r-1)+j],\ c^r_{ij} \equiv 0,\ 1 \leqslant i < N, 0 \leqslant j \leqslant 3.$$

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 $\text{ANUBIS}[K] : \mathrm{GF}(2^8)^{16} \to \mathrm{GF}(2^8)^{16}$ given by

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

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:

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

Using the associativity of functional composition we can slightly change 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, we must define two security-related concepts:

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 [block length and key length]. This must be the case for all possible modes of access for the adversary and for any a priori key distribution.*

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.*

4.1 Goal

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

- 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].

4.2 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. [21], 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 $\lambda^{\mathcal{B}^2} = (16 \times 2^{-6})^{25} = 2^{-50}$. 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 [12], and typically applies to ciphers in which all transformations operate on well aligned data blocks, as is the case for ANUBIS where all transformations operate on bytes rather than individual bits. However, the fact that all submatrices of H are nonsingular makes a truncated differential attack against more than a few rounds of ANUBIS impossible, because the S/N ratio of an attack becomes too low. For 6 rounds or more, no attacks faster than exhaustive key search have been found.

5.3 Interpolation attacks

Interpolation attacks [11] 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 involved expression of the 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 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 [4]. Typically, this occurs for ciphers where 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, though it seems unlikely that this could be so due to the MDS nature of both θ and ω used in the key schedule. Moreover, it is unclear how such keys could possibly be used successfully in a related key attack.

5.6 SQUARE (*aka* saturation) attacks

In this section we describe an attack first presented in [5], together with its variants [8]. 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 [8] 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 [9] 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

Any n-round attack can be extended against $(n+1)$ or more rounds for long keys by simply guessing the whole K^{n+1} round key and proceeding with the n-round attack [17]. Each extra round increases the complexity by a factor 2^{128} S-box lookups. The best attack known against 7 rounds of ANUBIS has complexity about 2^{140} S-box lookups, hence the 8-round extension costs $2^{140+128} = 2^{268}$ 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

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.

Attacks based on linear cryptanalysis can sometimes be improved by using nonlinear approximations [13]. 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 boomerang attack [22] 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; in particular, all classical Feistel networks [7] have this property. Self-inverse ciphers similar to ANUBIS were described and analyzed in [26, 27]. The importance of involutional structure resides not only in the advantages for implementation, but also in the equivalent security of both encryption and decryption.

6.2 Choice of the substitution box

The originally submitted form of ANUBIS used a pseudo-randomly generated S-box, 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 bounds on δ and λ correspond to twice the minimum achievable values for these quantities. An additional condition, that the S-box has no fixed point, was imposed in an attempt to speed up the search. This condition was inspired by the empirical study reported in [26, section 2.3], 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. The polynomial and rational representations of S over $\mathrm{GF}(2^8)$ are checked as well, to avoid any obvious algebraic weakness (which could lead e.g. to interpolation attacks [11]).

However, the extreme lack of structure in such an S-box hinders efficient hardware implementation; moreover, a flaw that went unnoticed in the random search program caused the value of λ for the original S-box to be incorrectly reported as 13×2^{-6} instead of the actual value 17×2^{-6} (corresponding to a negative bias), which is slightly worse than the design bound[2]. Although this is

[2] We thank the NESSIE evaluation team for pointing out this discrepancy [20].

still far too low to make classical linear attacks feasible, it can be easily remedied. Therefore, we now describe an alternative S-box that, besides exactly satisfying the design conditions, is amenable to much more efficient implementation in hardware, while not affecting the software implementation techniques presented here in any reasonable way.

The new S-box is illustrated in figure 1.

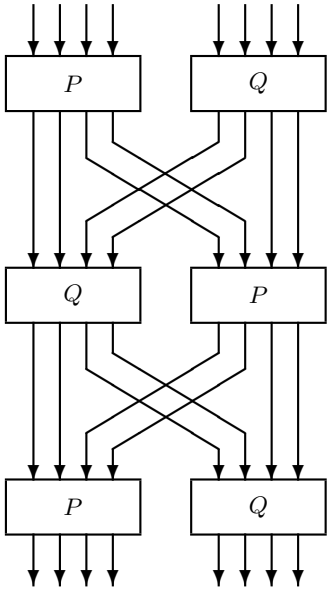

Fig. 1. Structure of the ANUBIS S-box. Both P and Q are pseudo-randomly generated involutions; the output from the upper and middle nonlinear layers are mixed through a simple linear shuffling.

The P and Q tables are pseudo-randomly generated involutions with optimal δ, λ, and ν, chosen so that the S-box built from them satisfies the design criteria listed at the beginning of this section. Tables 1 and 2 show the involutions found by the searching algorithm.

A description of the searching algorithm and a listing of the resulting S-box are given in the appendix.

Table 1. Actual P mini-box

u	0_x	1_x	2_x	3_x	4_x	5_x	6_x	7_x	8_x	9_x	A_x	B_x	C_x	D_x	E_x	F_x
$P[u]$	3_x	F_x	E_x	0_x	5_x	4_x	B_x	C_x	D_x	A_x	9_x	6_x	7_x	8_x	2_x	1_x

Table 2. Actual Q mini-box

u	0_x	1_x	2_x	3_x	4_x	5_x	6_x	7_x	8_x	9_x	A_x	B_x	C_x	D_x	E_x	F_x
$Q[u]$	9_x	E_x	5_x	6_x	A_x	2_x	3_x	C_x	F_x	0_x	4_x	D_x	7_x	B_x	1_x	8_x

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, hence the choice of Hadamard matrices. The actual choice involves coefficients with the lowest possible Hamming weight and integer values, which is advantageous for hardware and smart card implementations.

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 division of the key schedule into separate layers is directly taken from the round structure itself, which ultimately follows the Wide Trail strategy. The goal is to thwart attacks based on 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. 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. The actual choice meets these constraints in a simple and efficient manner 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 and ψ: For these composite mappings (see sections 3.9 and 3.10) we suggest a lookup-table approach. Let H_k be the k-th row of the Hadamard matrix H; using four tables $T_k[x] \equiv S[x] \cdot H_k,\ 0 \leqslant k \leqslant 3$, the rows b_i of $b = (\theta \circ \tau \circ \gamma)(a)$ and the rows d_i of $d = (\theta \circ \pi \circ \gamma)(c)$ can be calculated with four table lookups and three exor operations each:

$$b_i = \bigoplus_{k=0}^{3} T_k[a_{ki}],\ d_i = \bigoplus_{k=0}^{3} T_k[c_{(i-k) \bmod N,k}],\ 0 \leqslant i \leqslant 3;$$

the round key or round constant addition then completes the evaluation of ρ and ϕ. 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.

Implementation of ϕ: By definition (see section 3.9), $K^r = (\tau \circ \omega \circ \gamma)(\kappa^r) = (V \cdot \gamma(\kappa^r))^t = \gamma(\kappa^r)^t \cdot V^t$, so that $K^r_{ij} = \bigoplus_{k=0}^{N-1} S[\kappa^r_{ki}] \cdot a^k_j$. Hence a row $K^r_i \equiv [\, K^r_{i0}\ K^r_{i1}\ K^r_{i2}\ K^r_{i3}\,]$ of K^r can be computed by the following algorithm:

$K^r_i \leftarrow S[\kappa^r_{N-1,\,i}] \cdot [\, \mathtt{01}_x\ \mathtt{01}_x\ \mathtt{01}_x\ \mathtt{01}_x\,]$;
for $(k \leftarrow N-2; k \geqslant 0;\ k{-}{-})$ {
  $K^r_i \leftarrow S[\kappa^r_{ki}] \cdot [\, \mathtt{01}_x\ \mathtt{01}_x\ \mathtt{01}_x\ \mathtt{01}_x\,]$
    $\oplus\ K^r_{i,0} \cdot [\, \mathtt{01}_x\ \mathtt{00}_x\ \mathtt{00}_x\ \mathtt{00}_x\,] \oplus\ K^r_{i,1} \cdot [\, \mathtt{00}_x\ \mathtt{02}_x\ \mathtt{00}_x\ \mathtt{00}_x\,]$
    $\oplus\ K^r_{i,2} \cdot [\, \mathtt{00}_x\ \mathtt{00}_x\ \mathtt{06}_x\ \mathtt{00}_x\,] \oplus\ K^r_{i,3} \cdot [\, \mathtt{00}_x\ \mathtt{00}_x\ \mathtt{00}_x\ \mathtt{08}_x\,]$;
}

Using the following two tables:

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

a row K^r_i 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:

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

then K^r_i can be computed as $K^r_i = \bigoplus_{k=0}^{N-1} U_k[\kappa^r_{ki}]$ with only N table lookups and $N-1$ exor operations.

Implementation of θ for the inverse key schedule: The simplest approach is to use tables similar to those suggested for the implementation of ρ. However, instead of defining independent tables $T'_i[x] = x \cdot H_i,\ 0 \leqslant i \leqslant 3$, one can use the relation $T'_i[x] = T_i[S[x]]$ and extract the value of $S[x]$ from the $\mathtt{01}_x$ entries of the T tables with a masking 'and' operation. This way the T tables themselves can be used and no extra storage is needed.

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 $b = \theta(a)$ with 12 exors and 6 table lookups, using a table X that implements multiplication by the polynomial $g(x) = x$ in GF(2^8) (i.e. $X[u] \equiv x \cdot u$), and four registers r_0, r_1, r_2, r_3:

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

The following algorithm computes one row of $K^r = \phi(\kappa^r)$ with $5(N-1)$ exors and $1 + 7(N-1)$ table lookups, using auxiliary registers u, v, and k:

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

7.3 Techniques to avoid software implementation weaknesses

The attacks of Kocher *et al.* [14, 15] 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* [14] 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 shift followed by a conditional exor. This vulnerability is avoided by a table lookup implementation 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 [15]. 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 attainable 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 [10, 23] will carry over to some extent[3].

[3] In particular, the S-box structure can be implemented in about 1/5 the number of gates used by the implementation of the Rijndael S-box reported in [24], which takes about 500–600 gates [25].

8 Efficiency estimates

To obtain efficiency estimates we used the implementation with six tables described in section 7.1 on a 550 MHz Pentium III processor. We expect that optimised assembler code can reduce the cycle counts by a factor of at least 2.

8.1 Key setup

Table 3 lists the observed key setup efficiency. The increased cost of the decryption key schedule (13% to 35% more expensive than the encryption key schedule) is due to the application of θ to $R - 1$ round keys.

Table 3. 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

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. The observed efficiency is summarised on table 4.

Table 4. 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

Anubis is much more scalable than most modern ciphers, being very fast while avoiding excessive storage space and expensive or unusual instructions built in the processor; hence it is suitable for a wide variety of platforms. Its structure also favours extensively parallel execution of the component mappings, and its mathematical simplicity makes analysis easier.

9.1 Comparison with other SQUARE ciphers

Anubis bears many similarities with the block ciphers Square and Rijndael. We now list the most important differences.

The involutional structure: The use of involutions should in principle reduce the code size or area in software and hardware applications that implement both encryption and decryption.

The different S-box: The S-box of Anubis contains elements generated in a pseudo-random way and lacks a simple mathematical description needed for e.g. interpolation attacks; besides, the internal organisation of these elements potentially facilitates hardware implementation. On the other hand the differential and linear properties are suboptimal.

A more complex key schedule: This results in improved resistance against key based attacks, in particular shortcuts for long keys, at the cost of slower execution, reduced key agility, and larger code or gate count.

9.2 Extensions

The chosen key selection function is based on matrix $\mathrm{vdm}_N(\mathtt{01}_x, \mathtt{02}_x, \mathtt{06}_x, \mathtt{08}_x)$, 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 $\mathrm{vdm}_N(\mathtt{02}_x, \mathtt{15}_x, \mathtt{1c}_x, \mathtt{3c}_x)$ for the default Anubis matrix. Especially crafted matrices might be chosen for particular key sizes; for instance, $\mathrm{vdm}_4(\mathtt{01}_x, \mathtt{02}_x, \mathtt{05}_x, \mathtt{06}_x)$ may have some implementation advantages for 128-bit keys.

10 Acknowledgements

We are grateful to the Cryptix development team, particularly Raïf Naffah, for carefully reading and suggesting improvements for the implementation guidelines provided in this paper, and for implementing several versions of Anubis in Java.

We are deeply indebted to Brian Gladman for providing software and hardware facilities to search for efficient mini-box implementations in terms of Boolean functions.

We would also like to thank the NESSIE project organisers and evaluation team for making this work possible.

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 (1998), `http://www.nist.gov/aes`.
7. H. Feistel, "Cryptography and computer privacy," *Scientific American*, v. 228, n. 5, 1973, pp. 15–23.
8. 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.
9. H. Gilbert and M. Minier, "A collision attack on 7 rounds of Rijndael," *Third Advanced Encryption Standard Candidate Conference*, NIST (2000), pp. 230–241.
10. T. Ichikawa, T. Kasuya, M. Matsui, "Hardware evaluation of the AES finalists," *Third Advanced Encryption Standard Candidate Conference*, NIST, April 2000, pp. 279–285.
11. 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.
12. L.R. Knudsen, "Truncated and higher order differentials," *Fast Software Encryption, LNCS 1008*, B. Preneel, Ed., Springer-Verlag, 1995, pp. 196–211.
13. 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.
14. 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.
15. P. Kocher, J. Jaffe, B. Jun, "Introduction to differential power analysis and related attacks," available from `http://www.cryptography.com/dpa/technical/`.
16. R. Lidl and H. Niederreiter, "Introduction to finite fields and their applications," Cambridge University Press, 1986.
17. S. Lucks, "Attacking seven rounds of Rijndael under 192-bit and 256-bit keys," *Third Advanced Encryption Standard Candidate Conference*, NIST (2000), pp. 215–229.
18. F.J. MacWilliams and N.J.A. Sloane, "The theory of error-correcting codes," *North-Holland Mathematical Library*, vol. 16, 1977.
19. NESSIE Project – New European Schemes for Signatures, Integrity and Encryption – home page: `http://cryptonessie.org`.
20. NESSIE evaluation team, private communication, 2001; see also E. Biham, O. Dunkelman, V. Furman, T. Mor, "Preliminary report on the NESSIE submissions Anubis, Camellia, IDEA, Khazad, Misty1, Nimbus, Q," NESSIE report, 2001 – available online at: `https://www.cosic.esat.kuleuven.ac.be/nessie/reports/tecwp3-011b.pdf`.

21. V. Rijmen, "Cryptanalysis and design of iterated block ciphers," *Doctoral Dissertation*, October 1997, K.U.Leuven.
22. D. Wagner, "The boomerang attack," *Fast Software Encryption, LNCS 1636*, L. Knudsen, Ed., Springer-Verlag, 1999, pp. 156–170.
23. B. Weeks, M. Bean, T. Rozylowicz, C. Ficke, "Hardware performance simulations of round 2 AES algorithms," *Third Advanced Encryption Standard Candidate Conference*, NIST, April 2000, pp. 286–304.
24. D. Whiting, "AES implementations in 0.25 micron ASIC," NIST AES electronic discussion forum posting, August 2000.
25. D. Whiting, private communication, 2001.
26. 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.
27. 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 S-box

The only part of the S-box structure still unspecified in figure 1 consists of the P and Q involutions, which are generated pseudo-randomly in a verifiable way.

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

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

The pseudo-random number generator is implemented with RIJNDAEL [6] in counter mode, with a fixed key consisting of 256 zero bits and an initial counter value consisting of 128 zero bits.

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

```
// initialize mini-boxes to the negation involution:
for (u ← 0; u < 256; u++) {
    P[u] ← ū; Q[u] ← ū;
}
// look for S-box conforming to the design criteria:
do {
    // swap mini-boxes (update the "older" one only)
```

$P \leftrightarrow Q$;
// generate a random involution free of fixed points:
do {
do {
// randomly select x and y such that
// $x \neq y$ and $Q[x] \neq y$ (this implies $Q[y] \neq x$):
$z \leftarrow$ RandomByte(); $x \leftarrow z \gg 4$; $y \leftarrow z \,\&\, \texttt{0F}_x$;
} **while** $(x = y \ \vee \ Q[x] = y)$;
// swap entries:
$u \leftarrow Q[x]$; $v \leftarrow Q[y]$;
$Q[x] \leftarrow v$; $Q[u] \leftarrow y$;
$Q[y] \leftarrow u$; $Q[v] \leftarrow x$;
} **while** $(\delta(Q) > 1/4 \ \vee \ \lambda(Q) > 1/2 \ \vee \ \nu(Q) < 3)$;
// build S-box from the mini-boxes (see figure 1):
$S \leftarrow$ ShuffleStructure(P, Q);
// test design criteria:
} **while** $(\#\text{FixedPoints}(S) > 0 \ \vee \ \delta(S) > 2^{-5} \ \vee \ \lambda(S) > 2^{-2} \ \vee \ \nu(S) < 7)$;

B Hardware implementation

Restricting the allowed logical gates to AND, OR, NOT, and XOR, the P and Q mini-boxes can be implemented with 18 logical gates each. Therefore, the complete S-box can be implemented with 108 gates.

The pseudo-code fragments shown in figure 2 illustrate this ($u = u_3x^3 + u_2x^2+u_1x+u_0 \in \text{GF}(2^4)$ denotes the mini-box input, $z = z_3x^3+z_2x^2+z_1x+z_0 \in \text{GF}(2^4)$ denotes its output, and the t_k denote intermediate values).

We point out, however, that the search for efficient Boolean expressions for the mini-boxes has not been thorough, and it is likely that better expressions exist.

For completeness, table 5 lists the resulting 8×8 Anubis S-box.

C 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 `:-)`

$z = P[u]$	$z = Q[u]$
$t_0 \leftarrow u_0 \oplus u_1;$	$t_0 \leftarrow \neg u_0;$
$t_1 \leftarrow u_0 \oplus u_3;$	$t_1 \leftarrow u_1 \oplus u_2;$
$t_2 \leftarrow u_2 \wedge t_1;$	$t_2 \leftarrow u_2 \wedge t_0;$
$t_3 \leftarrow u_3 \wedge t_1;$	$t_3 \leftarrow u_3 \oplus t_2;$
$t_4 \leftarrow t_0 \vee t_3;$	$t_4 \leftarrow t_1 \wedge t_3;$
$z_3 \leftarrow t_2 \oplus t_4;$	$z_0 \leftarrow t_0 \oplus t_4;$
$t_1 \leftarrow \neg t_1;$	$t_0 \leftarrow u_0 \oplus u_1;$
$t_2 \leftarrow u_1 \wedge u_2;$	$t_1 \leftarrow t_1 \oplus t_2;$
$t_4 \leftarrow u_3 \vee z_3;$	$t_0 \leftarrow t_0 \oplus t_3;$
$t_1 \leftarrow t_1 \oplus t_2;$	$t_1 \leftarrow t_1 \vee t_0;$
$z_0 \leftarrow t_4 \oplus t_1;$	$z_2 \leftarrow u_2 \oplus t_1;$
$t_4 \leftarrow u_2 \wedge t_1;$	$t_1 \leftarrow t_1 \wedge u_0;$
$t_2 \leftarrow t_2 \oplus u_3;$	$t_3 \leftarrow u_3 \wedge t_0;$
$t_2 \leftarrow t_2 \vee t_4;$	$t_3 \leftarrow t_3 \oplus t_2;$
$z_2 \leftarrow t_0 \oplus t_2;$	$z_1 \leftarrow t_1 \oplus t_3;$
$t_3 \leftarrow t_3 \oplus t_4;$	$t_1 \leftarrow u_2 \vee z_0;$
$t_1 \leftarrow t_1 \vee z_3;$	$t_0 \leftarrow t_0 \oplus t_3;$
$z_1 \leftarrow t_3 \oplus t_1;$	$z_3 \leftarrow t_1 \oplus t_0;$

Fig. 2. Boolean expressions for P and Q

Table 5. The ANUBIS S-box

	00_x	01_x	02_x	03_x	04_x	05_x	06_x	07_x	08_x	09_x	$0A_x$	$0B_x$	$0C_x$	$0D_x$	$0E_x$	$0F_x$
00_x	BA_x	54_x	$2F_x$	74_x	53_x	$D3_x$	$D2_x$	$4D_x$	50_x	AC_x	$8D_x$	BF_x	70_x	52_x	$9A_x$	$4C_x$
10_x	EA_x	$D5_x$	97_x	$D1_x$	33_x	51_x	$5B_x$	$A6_x$	DE_x	48_x	$A8_x$	99_x	DB_x	32_x	$B7_x$	FC_x
20_x	$E3_x$	$9E_x$	91_x	$9B_x$	$E2_x$	BB_x	41_x	$6E_x$	$A5_x$	CB_x	$6B_x$	95_x	$A1_x$	$F3_x$	$B1_x$	02_x
30_x	CC_x	$C4_x$	$1D_x$	14_x	$C3_x$	63_x	DA_x	$5D_x$	$5F_x$	DC_x	$7D_x$	CD_x	$7F_x$	$5A_x$	$6C_x$	$5C_x$
40_x	$F7_x$	26_x	FF_x	ED_x	$E8_x$	$9D_x$	$6F_x$	$8E_x$	19_x	$A0_x$	$F0_x$	89_x	$0F_x$	07_x	AF_x	FB_x
50_x	08_x	15_x	$0D_x$	04_x	01_x	64_x	DF_x	76_x	79_x	DD_x	$3D_x$	16_x	$3F_x$	37_x	$6D_x$	38_x
60_x	$B9_x$	73_x	$E9_x$	35_x	55_x	71_x	$7B_x$	$8C_x$	72_x	88_x	$F6_x$	$2A_x$	$3E_x$	$5E_x$	27_x	46_x
70_x	$0C_x$	65_x	68_x	61_x	03_x	$C1_x$	57_x	$D6_x$	$D9_x$	58_x	$D8_x$	66_x	$D7_x$	$3A_x$	$C8_x$	$3C_x$
80_x	FA_x	96_x	$A7_x$	98_x	EC_x	$B8_x$	$C7_x$	AE_x	69_x	$4B_x$	AB_x	$A9_x$	67_x	$0A_x$	47_x	$F2_x$
90_x	$B5_x$	22_x	$E5_x$	EE_x	BE_x	$2B_x$	81_x	12_x	83_x	$1B_x$	$0E_x$	23_x	$F5_x$	45_x	21_x	CE_x
$A0_x$	49_x	$2C_x$	$F9_x$	$E6_x$	$B6_x$	28_x	17_x	82_x	$1A_x$	$8B_x$	FE_x	$8A_x$	09_x	$C9_x$	87_x	$4E_x$
$B0_x$	$E1_x$	$2E_x$	$E4_x$	$E0_x$	EB_x	90_x	$A4_x$	$1E_x$	85_x	60_x	00_x	25_x	$F4_x$	$F1_x$	94_x	$0B_x$
$C0_x$	$E7_x$	75_x	EF_x	34_x	31_x	$D4_x$	$D0_x$	86_x	$7E_x$	AD_x	FD_x	29_x	30_x	$3B_x$	$9F_x$	$F8_x$
$D0_x$	$C6_x$	13_x	06_x	05_x	$C5_x$	11_x	77_x	$7C_x$	$7A_x$	78_x	36_x	$1C_x$	39_x	59_x	18_x	56_x
$E0_x$	$B3_x$	$B0_x$	24_x	20_x	$B2_x$	92_x	$A3_x$	$C0_x$	44_x	62_x	10_x	$B4_x$	84_x	43_x	93_x	$C2_x$
$F0_x$	$4A_x$	BD_x	$8F_x$	$2D_x$	BC_x	$9C_x$	$6A_x$	40_x	CF_x	$A2_x$	80_x	$4F_x$	$1F_x$	CA_x	AA_x	42_x