The KHAZAD Legacy-Level 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. KHAZAD is a 64-bit (legacy-level) block cipher that accepts a 128-bit key. The cipher is a uniform substitution-permutation network whose inverse only differs from the forward operation in the key schedule. The overall cipher design follows the Wide Trail strategy, favours component reuse, and permits a wide variety of implementation tradeoffs.

1 Introduction

In this document we describe KHAZAD, a 64-bit (legacy-level) block cipher that accepts a 128-bit key.

Although KHAZAD 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 KHAZAD is used e.g. in a Java applet.

KHAZAD was designed according to the Wide Trail strategy [5]. 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 largely 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 KHAZAD 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 5.9). 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

If m is a power of 2, $\mathrm{had}(a_0, \ldots, a_{m-1})$ denotes the $m \times m$ Hadamard matrix [2] 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 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 KHAZAD primitive

The KHAZAD cipher is an iterated 'involutional'[1] block cipher that operates on a 64-bit *cipher state* represented as a vector in $\mathrm{GF}(2^8)^8$. It uses a 128-bit *cipher key* K represented as a vector in $\mathrm{GF}(2^8)^{16}$, 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 KHAZAD, then specify the complete cipher in terms of these components.

[1] We explain in section 3.8 what we mean by an 'involutional' block cipher.

3.1 The nonlinear layer γ

Function $\gamma : \mathrm{GF}(2^8)^8 \to \mathrm{GF}(2^8)^8$ consists of the parallel application of a nonlinear substitution box $S : \mathrm{GF}(2^8) \to \mathrm{GF}(2^8), x \mapsto S[x]$ to all bytes of the argument individually:

$$\gamma(a) = b \Leftrightarrow b_i = S[a_i],\ 0 \leqslant i \leqslant 7.$$

The substitution box was pseudo-randomly chosen and is listed in appendix A. 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.

3.2 The linear diffusion layer θ

The diffusion layer $\theta : \mathrm{GF}(2^8)^8 \to \mathrm{GF}(2^8)^8$ is a linear mapping based on the $[16, 8, 9]$ MDS code with generator matrix $G_H = [I\,H]$ where $H = \mathrm{had}(\text{`01'}, \text{`03'}, \text{`04'}, \text{`05'}, \text{`06'}, \text{`08'}, \text{`0b'}, \text{`07'})$, i.e.

$$H = \begin{bmatrix} \text{`01'} & \text{`03'} & \text{`04'} & \text{`05'} & \text{`06'} & \text{`08'} & \text{`0b'} & \text{`07'} \\ \text{`03'} & \text{`01'} & \text{`05'} & \text{`04'} & \text{`08'} & \text{`06'} & \text{`07'} & \text{`0b'} \\ \text{`04'} & \text{`05'} & \text{`01'} & \text{`03'} & \text{`0b'} & \text{`07'} & \text{`06'} & \text{`08'} \\ \text{`05'} & \text{`04'} & \text{`03'} & \text{`01'} & \text{`07'} & \text{`0b'} & \text{`08'} & \text{`06'} \\ \text{`06'} & \text{`08'} & \text{`0b'} & \text{`07'} & \text{`01'} & \text{`03'} & \text{`04'} & \text{`05'} \\ \text{`08'} & \text{`06'} & \text{`07'} & \text{`0b'} & \text{`03'} & \text{`01'} & \text{`05'} & \text{`04'} \\ \text{`0b'} & \text{`07'} & \text{`06'} & \text{`08'} & \text{`04'} & \text{`05'} & \text{`01'} & \text{`03'} \\ \text{`07'} & \text{`0b'} & \text{`08'} & \text{`06'} & \text{`05'} & \text{`04'} & \text{`03'} & \text{`01'} \end{bmatrix},$$

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.

3.3 The key addition $\sigma[k]$

The affine key addition $\sigma[k] : \mathrm{GF}(2^8)^8 \to \mathrm{GF}(2^8)^8$ consists of the bitwise addition (exor) of a key vector $k \in \mathrm{GF}(2^8)^8$:

$$\sigma[k](a) = b \Leftrightarrow b_i = a_i \oplus k_i,\ 0 \leqslant i \leqslant 7.$$

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

3.4 The round constants c^r

The round constant for the r-th round is a vector $c^r \in \mathrm{GF}(2^8)^8$, defined as:

$$c^r_i = S[8r + i],\ 0 \leqslant i \leqslant R.$$

3.5 The round function $\rho[k]$

The r-th round function is the composite mapping $\rho[k] : \mathrm{GF}(2^8)^8 \to \mathrm{GF}(2^8)^8$, parameterised by the key vector $k \in \mathrm{GF}(2^8)^8$ and given by:

$$\rho[k] \equiv \sigma[k] \circ \theta \circ \gamma.$$

3.6 The key schedule

The key schedule expands the cipher key $K \in \mathrm{GF}(2^8)^{16}$ into a sequence of round keys $K^0, \ldots, K^R$, plus two initial values, K^{-2} and K^{-1}), with $K^r \in \mathrm{GF}(2^8)^8$. The initial values K^{-2} and K^{-1} are taken respectively from bytes 0 through 7 and 8 through 15 of the cipher key K:

$$\left.\begin{array}{l} K_i^{-2} = K_i, \\ K_i^{-1} = K_{8+i}, \end{array}\right\} \; 0 \leqslant i \leqslant 7.$$

The sequence of round keys are computed by means of a Feistel iteration based on the round function ρ and the round constants c^r:

$$K^r = \rho[c^r](K^{r-1}) \oplus K^{r-2}, \; 0 \leqslant r \leqslant R.$$

3.7 The complete cipher

Khazad is defined for the cipher key K as the transformation $\text{Khazad}[K] = \alpha_R[K^0, \ldots, K^R]$ applied to the plaintext, where

$$\alpha_R[K^0, \ldots, K^R] = \sigma[K^R] \circ \gamma \circ \left(\bigcirc_1^{r=R-1} \rho[K^r] \right) \circ \sigma[K^0].$$

The standard number of rounds is $R = 8$.

3.8 The inverse cipher

We now show that Khazad 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 lemma:

Lemma 1. *$\theta \circ \sigma[K^r] = \sigma[\theta(K^r)] \circ \theta$.*

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 $a \in \mathrm{GF}(2^8)^8$. □

Let $\bar{K}^0 \equiv K^R$, $\bar{K}^R \equiv K^0$, and $\bar{K}^r \equiv \theta(K^{R-r})$, $0 < r < R$. We are now ready to state the main property of the inverse Khazad cipher $\alpha_R^{-1}[K^0, \ldots, K^R]$:

Theorem 1. *$\alpha_R^{-1}[K^0, \ldots, K^R] = \alpha_R[\bar{K}^0, \ldots, \bar{K}^R]$.*

Proof. We start from the definition of R-round Khazad:

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

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

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

The above lemma leads to:

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

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

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

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

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

That is, $\alpha_R^{-1}[K^0, \ldots, K^R] = \alpha_R[\bar{K}^0, \ldots, \bar{K}^R]$, where $\bar{K}^0 \equiv K^R$, $\bar{K}^R \equiv K^0$, and $\bar{K}^r \equiv \theta(K^{R-r})$, $0 < r < R$. □

Corollary 1. *The* Khazad *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 Khazad. 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 ([5]). *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 ([5]). *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 KHAZAD cipher is:

- K-secure;
- Hermetic.

If KHAZAD 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

KHAZAD is expected 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 KHAZAD 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. Since the cipher uses 128-bit keys, the expected effort of exhaustive key search is 2^{127} applications of KHAZAD.

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

Because the branch number of θ is $\mathcal{B} = 9$ (cf. [26], proposition 1), no differential characteristic over two rounds has probability larger than $\delta^{\mathcal{B}} = (2^{-5})^9 = 2^{-45}$, and no linear approximation over two rounds has input-output correlation larger than $\lambda^{\mathcal{B}} = (13\times 2^{-6})^9 \approx 2^{-20.7}$. 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 [17], and typically applies to ciphers in which all transformations operate on well aligned data blocks. Since in KHAZAD all transformations operate on bytes rather than individual bits, we investigated its resistance against truncated differentials. The fact that all submatrices of H are nonsingular, makes a truncated differential attack against more than a few rounds of KHAZAD impossible, because the S/N ratio of an attack becomes too low. For 4 rounds or more, no truncated differential attacks can be mounted.

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 $\mathrm{GF}(2^8)$, 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 [5]. Typically, this weakness occurs for ciphers in which the nonlinear operations depend on the actual key value. This is not the case for KHAZAD, where keys are applied using exor and all nonlinearity is in the fixed S-box. In KHAZAD, 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 KHAZAD 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.

5.6 The Shark attack and its variants

In this section we present an attack first described in [25]. This attack works against KHAZAD reduced to 3 rounds. 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_i is the byte at position i of the cipher state at the output of round 1.

Take a set of 256 plaintexts different from each other in a single byte (which assumes all possible values), the remaining 7 bytes being constant. After one round all 8 bytes of each cipher state a^2 in the set will take every value exactly once. After two rounds, the exor of all 256 cipher states a^3 at every byte position will be zero.

Consider a ciphertext $b^3 = \gamma(a^3) \oplus K^3$; clearly $a^3 = \gamma(b^3 \oplus K^3)$. Now take a byte from b^3, guess the matching byte from K^3 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 K^3 value with overwhelming probability.

This attack recovers one byte of the last round key. The remaining bytes can be obtained by repeating the attack eight times. Overall, this attack requires 2^9 chosen plaintexts. However, almost all wrong key values can be eliminated after processing a single set of 2^8 plaintexts. The workload to recover one key byte is thus 2^8 key guesses $\times 2^8$ chosen plaintexts $= 2^{16}$ S-box lookups.

5.7 A general extension attack

Stefan Lucks [21] presents a general extension of any n-round attack; the result is an attack against $(n+1)$ rounds. The idea is simply to guess the whole K^{n+1} round key and proceed with the n-round attack. This increases the complexity by a factor 2^{64} S-box lookups.

The best attack known against 3 rounds of KHAZAD has complexity about 2^{16} S-box lookups, hence the 4-round extension costs $2^{16+64} = 2^{80}$ S-box lookups.

5.8 Other attacks

Attacks based on linear cryptanalysis can sometimes be improved by using nonlinear approximations [18]. 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 KHAZAD.

The boomerang attack [27] benefits from ciphers whose encryption and decryption strengths are different; this is hardly the case for KHAZAD, due to its involutional structure.

We see no obvious way to extend the Gilbert-Minier attack [10] against RIJNDAEL and other ciphers of the Square family, since the attack makes direct use of the two-level diffusion structure of those ciphers.

An extension of the Biham-Keller impossible differential attack on RIJNDAEL reduced to 5 rounds [4] can be applied to KHAZAD, reduced to 3 rounds. The attack requires 2^{13} chosen plaintexts and an effort of 2^{64} encryptions.

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

5.9 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 KHAZAD, do hereby declare that there are no hidden weaknesses inserted by us in the KHAZAD 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 KHAZAD were described and analyzed in [29, 30].

The importance of involutional structure resides not only in the advantages for implementation, but also in the equivalent security of both encryption and decryption [19].

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 KHAZAD 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 [29], 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 $\mathrm{GF}(2^8)$ 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 KHAZAD use circulant matrices for this purpose (cf. [26]), 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 7.3).

6.4 Structure of the key schedule

Adopting a Feistel key schedule provides a simple and effective way to expand a $2m$-bit cipher key onto m-bit round keys reusing the round function itself. This keeps the overall cipher structure uniformly m-bit oriented (in the sense that the natural data units occurring in the cipher are bytes and m-bit blocks).

6.5 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 while also reusing an available component (the S-box itself).

7 Implementation

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

7.1 64-bit processors

We suggest a lookup-table approach to implement ρ:

$$
\begin{aligned}
b &= (\theta \circ \gamma)(a) \\
&\Updownarrow \\
\begin{bmatrix} b_0 \; b_1 \; b_2 \; b_3 \; b_4 \; b_5 \; b_6 \; b_7 \end{bmatrix} &= \begin{bmatrix} S[a_0] \; S[a_1] \; S[a_2] \; S[a_3] \; S[a_4] \; S[a_5] \; S[a_6] \; S[a_7] \end{bmatrix} \cdot H \\
&\Updownarrow \\
\begin{bmatrix} b_0 \; b_1 \; b_2 \; b_3 \; b_4 \; b_5 \; b_6 \; b_7 \end{bmatrix} &= S[a_0] \cdot [\text{'01' '03' '04' '05' '06' '08' '0b' '07'}] \\
&\oplus S[a_1] \cdot [\text{'03' '01' '05' '04' '08' '06' '07' '0b'}] \\
&\oplus S[a_2] \cdot [\text{'04' '05' '01' '03' '0b' '07' '06' '08'}] \\
&\oplus S[a_3] \cdot [\text{'05' '04' '03' '01' '07' '0b' '08' '06'}] \\
&\oplus S[a_4] \cdot [\text{'06' '08' '0b' '07' '01' '03' '04' '05'}] \\
&\oplus S[a_5] \cdot [\text{'08' '06' '07' '0b' '03' '01' '05' '04'}] \\
&\oplus S[a_6] \cdot [\text{'0b' '07' '06' '08' '04' '05' '01' '03'}] \\
&\oplus S[a_7] \cdot [\text{'07' '0b' '08' '06' '05' '04' '03' '01'}].
\end{aligned}
$$

Using the following eight tables:

$$
\begin{aligned}
T_0[x] &= S[x] \cdot [\text{'01' '03' '04' '05' '06' '08' '0b' '07'}], \\
T_1[x] &= S[x] \cdot [\text{'03' '01' '05' '04' '08' '06' '07' '0b'}], \\
T_2[x] &= S[x] \cdot [\text{'04' '05' '01' '03' '0b' '07' '06' '08'}], \\
T_3[x] &= S[x] \cdot [\text{'05' '04' '03' '01' '07' '0b' '08' '06'}], \\
T_4[x] &= S[x] \cdot [\text{'06' '08' '0b' '07' '01' '03' '04' '05'}], \\
T_5[x] &= S[x] \cdot [\text{'08' '06' '07' '0b' '03' '01' '05' '04'}], \\
T_6[x] &= S[x] \cdot [\text{'0b' '07' '06' '08' '04' '05' '01' '03'}], \\
T_7[x] &= S[x] \cdot [\text{'07' '0b' '08' '06' '05' '04' '03' '01'}],
\end{aligned}
$$

a row of b can be calculated with eight table lookups and seven exor operations; the key addition then completes the evaluation of ρ. The T-tables require $2^8 \times 8$ 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.

7.2 32-bit processors

Any Hadamard matrix H (of order m) shows a recursive structure, in the sense that

$$
H = \begin{bmatrix} U & V \\ V & U \end{bmatrix},
$$

where U and V are themselves Hadamard matrices (of order $m/2$). A 32-bit implementation may take advantage of this structure by representing elements

$a \in \mathrm{GF}(2^8)^8$ as $a = \begin{bmatrix} \hat{a}_0 & \hat{a}_1 \end{bmatrix}$, where $\hat{a}_0, \hat{a}_1 \in \mathrm{GF}(2^8)^4$:

$$\begin{gathered} b = \theta(a) \\ \Updownarrow \\ \begin{bmatrix} \hat{b}_0 & \hat{b}_1 \end{bmatrix} = \begin{bmatrix} \hat{a}_0 & \hat{a}_1 \end{bmatrix} \cdot \begin{bmatrix} U & V \\ V & U \end{bmatrix} \\ \Updownarrow \\ \hat{b}_0 = \hat{a}_0 U \oplus \hat{a}_1 V, \\ \hat{b}_1 = \hat{a}_0 V \oplus \hat{a}_1 U, \end{gathered}$$

with twice the complexity derived for 64-bit processors regarding the number of table lookups and exors, but using smaller tables (each requiring $2^8 \times 4$ bytes of storage).

7.3 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 $\sigma[k]$ transformation. For θ, it is necessary to implement the matrix multiplication.

The following piece of pseudo-code calculates $b = \theta(a)$, using a table X that implements multiplication by the polynomial $g(x) = x$ in $\mathrm{GF}(2^8)$ and six registers $r_0, r_1, r_2, r_3, r_4, r_5$:

```
$r_0 = a_0 \oplus a_1 \oplus X[X[a_2 \oplus a_3]];$
$r_1 = a_2 \oplus a_3 \oplus X[X[a_0 \oplus a_1]];$
$r_2 = a_6 \oplus a_7;$
$r_3 = X[X[a_5 \oplus a_6]];$
$r_4 = a_4 \oplus a_5;$
$r_5 = X[X[a_4 \oplus a_7]];$
$b_0 = a_3 \oplus r_0 \oplus r_2 \oplus r_5 \oplus X[a_1 \oplus a_4 \oplus r_2 \oplus r_3];$
$b_1 = a_2 \oplus r_0 \oplus r_2 \oplus r_3 \oplus X[a_0 \oplus a_5 \oplus r_2 \oplus r_5];$
$b_2 = a_1 \oplus r_1 \oplus r_4 \oplus r_3 \oplus X[a_3 \oplus a_6 \oplus r_4 \oplus r_5];$
$b_3 = a_0 \oplus r_1 \oplus r_4 \oplus r_5 \oplus X[a_2 \oplus a_7 \oplus r_4 \oplus r_3];$
$r_0 = a_4 \oplus a_5 \oplus X[X[a_6 \oplus a_7]];$
$r_1 = a_6 \oplus a_7 \oplus X[X[a_4 \oplus a_5]];$
$r_2 = a_2 \oplus a_3;$
$r_3 = X[X[a_1 \oplus a_2]];$
$r_4 = a_0 \oplus a_1;$
$r_5 = X[X[a_0 \oplus a_3]];$
$b_4 = a_7 \oplus r_0 \oplus r_2 \oplus r_5 \oplus X[a_5 \oplus a_0 \oplus r_2 \oplus r_3];$
$b_5 = a_6 \oplus r_0 \oplus r_2 \oplus r_3 \oplus X[a_4 \oplus a_1 \oplus r_2 \oplus r_5];$
$b_6 = a_5 \oplus r_1 \oplus r_4 \oplus r_3 \oplus X[a_7 \oplus a_2 \oplus r_4 \oplus r_5];$
$b_7 = a_4 \oplus r_1 \oplus r_4 \oplus r_5 \oplus X[a_6 \oplus a_3 \oplus r_4 \oplus r_3];$
```

This implementation requires 76 exors, 24 table lookups and 20 assignments. Notice that, if an additional table $X2$ is available, where $X2[u] \equiv X[X[u]]$, the number of table lookups drops to 16. There may be more efficient ways to implement θ, however; we did not search thoroughly all possibilities.

7.4 Techniques to avoid software implementation weaknesses

The attacks of Kocher *et al.* [15, 16] 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* [15] 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 7.2 and 7.3.

A second class of attacks are the attacks based on the careful observation of the power consumption pattern of an encryption device [16]. 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 KHAZAD will make it easier to create an implementation that resists this type of attacks.

7.5 Hardware implementation

We have currently no figures on the attainable performance and required area or gate count of KHAZAD in ASIC or FPGA, nor do we have a description in VHDL. However, we expect that the results on RIJNDAEL [12, 28] will carry over to some extent.

8 Efficiency estimates

8.1 Key setup

Table 1 lists the observed key setup efficiency on a 550 MHz Pentium III platform. Since the key schedule is based on the round function itself, no extra storage is needed besides that already required for implementing encryption/decryption.

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 (the implicit γ transform in those tables is undone with an extra T_7 lookup and word masking). The setup of decryption keys is therefore 68% more expensive than the setup of encryption keys.

Table 1. Key setup efficiency

cycles (encryption schedule)	cycles (decryption schedule)
717	1206

We point out that the reference implementation used to measure efficiency is not fully optimised, and that the platform word size in 32 bits rather than 64 bits. By coding the key setup in assembler and running the test on a native 64-bit processor, we expect a reduction of the cycle counts by a factor of at least 2.

8.2 Encryption and decryption

Since KHAZAD has involutional structure, encryption and decryption are equally efficient (for the same number of rounds). Table 2 summarises the observed efficiency on a 550 MHz Pentium III platform. We use the eight-table implementation described in section 7.1.

Table 2. Encryption/decryption efficiency

cycles per byte	cycles per block	Mbit/s
67.0	536	65.7

9 Advantages

By design, KHAZAD 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 SHARK

KHAZAD has many similarities with the block cipher SHARK [26]. In this section, we list the most important differences.

The involutional structure: The fact that all components of KHAZAD 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 KHAZAD 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.

The different key scheduling: The key scheduling of KHAZAD executes faster than that of SHARK, and still provides adequate security. In particular for the processing of short messages, the performance of the key scheduling is important.

References

1. P.S.L.M. Barreto and V. Rijmen, "The ANUBIS block cipher," NESSIE submission, 2000.
2. K.G. Beauchamp, "Walsh functions and their applications," Academic Press, 1975.
3. 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.
4. E. Biham and N. Keller, "Cryptanalysis of reduced variants of RIJNDAEL," submission to the *Third Advanced Encryption Standard Candidate Conference.*
5. J. Daemen, "Cipher and hash function design strategies based on linear and differential cryptanalysis," *Doctoral Dissertation*, March 1995, K.U.Leuven.
6. 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.
7. J. Daemen and V. Rijmen, "AES proposal: RIJNDAEL," AES submission (1998), `http://www.nist.gov/aes`.
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," *Third Advanced Encryption Standard Candidate Conference*, NIST, April 2000, pp. 279–285.
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. 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.
16. P. Kocher, J. Jaffe, B. Jun, "Introduction to differential power analysis and related attacks," available from `http://www.cryptography.com/dpa/technical/`.

17. L.R. Knudsen, "Truncated and higher order differentials," *Fast Software Encryption, LNCS 1008*, B. Preneel, Ed., Springer-Verlag, 1995, pp. 196–211.
18. 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.
19. 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.
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. V. Rijmen, J. Daemen, B. Preneel, A. Bosselaers, E. De Win, "The cipher SHARK," *Fast Software Encryption, LNCS 1039*, D. Gollman , Ed., Springer-Verlag, 1996, pp. 99–111.
27. D. Wagner, "The boomerang attack," *Fast Software Encryption, LNCS 1636*, L. Knudsen, Ed., Springer-Verlag, 1999, pp. 156–170.
28. 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.
29. 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.
30. 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 The substitution box

KHAZAD uses the same S-box as the ANUBIS cipher (cf. [1], where the pseudo-random generation method is described in detail). We list the S-box contents here for ease of reference.

```
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
};
```