The KHAZAD Legacy-Level 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. 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. KHAZAD has been submitted as a candidate cryptographic primitive for the NESSIE project [23].

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

KHAZAD was designed according to the Wide Trail strategy [8]. 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 Computer Architecture and Networking Laboratory (LARC), Department of Computer and Digital Systems Engineering, Escola Politécnica da Universidade de São Paulo (Brazil).

As originally submitted for the NESSIE evaluation effort, KHAZAD 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 KHAZAD-0 and using the term KHAZAD 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 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

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 [20], 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. [22], 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* [25].

2.4 Miscellaneous notation

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

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.

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 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.

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

3.2 The linear diffusion layer θ

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

$$H = \begin{bmatrix} \mathtt{01}_x & \mathtt{03}_x & \mathtt{04}_x & \mathtt{05}_x & \mathtt{06}_x & \mathtt{08}_x & \mathtt{0B}_x & \mathtt{07}_x \\ \mathtt{03}_x & \mathtt{01}_x & \mathtt{05}_x & \mathtt{04}_x & \mathtt{08}_x & \mathtt{06}_x & \mathtt{07}_x & \mathtt{0B}_x \\ \mathtt{04}_x & \mathtt{05}_x & \mathtt{01}_x & \mathtt{03}_x & \mathtt{0B}_x & \mathtt{07}_x & \mathtt{06}_x & \mathtt{08}_x \\ \mathtt{05}_x & \mathtt{04}_x & \mathtt{03}_x & \mathtt{01}_x & \mathtt{07}_x & \mathtt{0B}_x & \mathtt{08}_x & \mathtt{06}_x \\ \mathtt{06}_x & \mathtt{08}_x & \mathtt{0B}_x & \mathtt{07}_x & \mathtt{01}_x & \mathtt{03}_x & \mathtt{04}_x & \mathtt{05}_x \\ \mathtt{08}_x & \mathtt{06}_x & \mathtt{07}_x & \mathtt{0B}_x & \mathtt{03}_x & \mathtt{01}_x & \mathtt{05}_x & \mathtt{04}_x \\ \mathtt{0B}_x & \mathtt{07}_x & \mathtt{06}_x & \mathtt{08}_x & \mathtt{04}_x & \mathtt{05}_x & \mathtt{01}_x & \mathtt{03}_x \\ \mathtt{07}_x & \mathtt{0B}_x & \mathtt{08}_x & \mathtt{06}_x & \mathtt{05}_x & \mathtt{04}_x & \mathtt{03}_x & \mathtt{01}_x \end{bmatrix},$$

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.

3.3 The key addition $\sigma[k]$

The affine key addition $\sigma[k] : \mathrm{GF}(2^8)^8 \rightarrow \mathrm{GF}(2^8)^8$ 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:

$$c_i^r = S[8r + i],\ 0 \leqslant r \leqslant R,\ 0 \leqslant i \leqslant 7.$$

3.5 The round function $\rho[k]$

The r-th round function is the composite mapping $\rho[k] : \mathrm{GF}(2^8)^8 \rightarrow \mathrm{GF}(2^8)^8$, 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:

$$K_i^{-2} = K_i,\ K_i^{-1} = K_{8+i},\ 0 \leqslant i \leqslant 7.$$

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 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:

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

Using the associativity of functional composition we can slightly change the grouping of operations:

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

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, we must define two security-related concepts:

Definition 1 ([8]). *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 ([8]). *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 Goals

The security goals are that the KHAZAD cipher be:

- 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 [8].

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 $\lambda^{\mathcal{B}} = (16 \times 2^{-6})^9 = 2^{-18}$. 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 [18], and typically applies to ciphers in which all transformations operate on well aligned data blocks, as is the case for KHAZAD 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 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 [15] generally depend on the cipher components (particularly the S-box) having simple algebraic structures that can be combined to give polynomial or rational expressions with manageable complexity. The involved expression of the S-box in $\mathrm{GF}(2^8)$, combined with the effect of the diffusion layer, makes this type of attack infeasible for more than a few rounds.

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 [8]. Typically, this occurs for ciphers where 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 SQUARE *aka* saturation attacks

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_i^1 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

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 [21]. Each extra round 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

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

We see no obvious way to extend the Gilbert-Minier attack [13] against RIJNDAEL and other ciphers of the SQUARE family, since the attack makes direct use of the two-level diffusion structure of those ciphers.

Attacks based on linear cryptanalysis can sometimes be improved by using nonlinear approximations [19]. 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.

We were not able to find any other attack method, including slide [6], advanced slide [7], boomerang [27], and rectangle [3] attacks, that could break the KHAZAD 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; in particular, all classical Feistel networks [11] have this property. Self-inverse ciphers similar to KHAZAD were described and analyzed in [31, 32]. 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 KHAZAD 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 [31, 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 [15]). Finally, affine approximations to S are also considered; no such approximation was found that matches S in more than 19 points[2], too few to mount any attack we could conceive.

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[3]. Although this is 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.

[2] This is true for both the original S-box and the new, improved design.

[3] We thank the NESSIE evaluation team for pointing out this discrepancy [24].

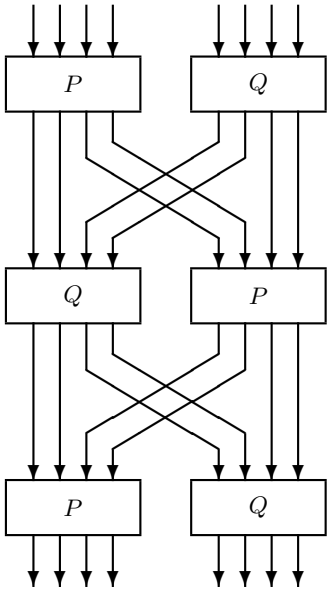

Fig. 1. Structure of the KHAZAD S-box. Both P and Q are pseudo-randomly generated involutions; the output from the upper and middle nonlinear layers are mixed through a simple linear shuffling.

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 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-like 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

Implementation of ρ: We suggest the following lookup-table approach. Let H_k be the k-th row of the Hadamard-like matrix H; using eight tables $T_k[x] \equiv S[x] \cdot H_k,\ 0 \leqslant k \leqslant 7$, i.e.:

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

then one can compute $b = (\theta \circ \gamma)(a) = \bigoplus_{k=0}^{7} T_k[a_k]$ 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.

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 $T'_i[x] = x \cdot H_i,\ 0 \leqslant i \leqslant 7$, 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 32-bit processors

Any Hadamard-like matrix H (of order m) shows the following structure:

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

where U and V are themselves Hadamard-like matrices (of order $m/2$). A 32-bit implementation may take advantage of this structure by representing elements $c \in \mathrm{GF}(2^8)^8$ as pairs $c = \begin{bmatrix} \hat{c}_0 & \hat{c}_1 \end{bmatrix}$ of elements $\hat{c}_i \in \mathrm{GF}(2^8)^4$:

$$b = \theta(a) \Leftrightarrow \begin{cases} \hat{b}_0 = \hat{a}_0 U \oplus \hat{a}_1 V, \\ \hat{b}_1 = \hat{a}_0 V \oplus \hat{a}_1 U, \end{cases}$$

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

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 $\mathrm{GF}(2^8)$ (i.e. $X[u] \equiv x \cdot u$) and fourteen temporary variables t_0 to t_7 and r_0 to r_5:

$t_0 \leftarrow a_0 \oplus a_1$; $t_1 \leftarrow a_2 \oplus a_3$; $t_2 \leftarrow a_4 \oplus a_5$; $t_3 \leftarrow a_6 \oplus a_7$;
$t_4 \leftarrow a_1 \oplus a_4$; $t_5 \leftarrow a_0 \oplus a_5$; $t_6 \leftarrow a_3 \oplus a_6$; $t_7 \leftarrow a_2 \oplus a_7$;
$r_0 \leftarrow t_0 \oplus X[X[t_1]]$; $r_2 \leftarrow X[X[a_5 \oplus a_6]]$;
$r_1 \leftarrow t_1 \oplus X[X[t_0]]$; $r_3 \leftarrow X[X[a_4 \oplus a_7]]$;
$r_4 \leftarrow t_3 \oplus r_2$; $r_5 \leftarrow t_3 \oplus r_3$;

$b_0 \leftarrow a_3 \oplus r_0 \oplus r_5 \oplus X[t_4 \oplus r_4]$; $b_1 \leftarrow a_2 \oplus r_0 \oplus r_4 \oplus X[t_5 \oplus r_5]$;
$r_4 \leftarrow t_2 \oplus r_2$; $r_5 \leftarrow t_2 \oplus r_3$;
$b_2 \leftarrow a_1 \oplus r_1 \oplus r_4 \oplus X[t_6 \oplus r_5]$; $b_3 \leftarrow a_0 \oplus r_1 \oplus r_5 \oplus X[t_7 \oplus r_4]$;
$r_0 \leftarrow t_2 \oplus X[X[t_3]]$; $r_2 \leftarrow X[X[a_1 \oplus a_2]]$;
$r_1 \leftarrow t_3 \oplus X[X[t_2]]$; $r_3 \leftarrow X[X[a_0 \oplus a_3]]$;
$r_4 \leftarrow t_1 \oplus r_2$; $r_5 \leftarrow t_1 \oplus r_3$;
$b_4 \leftarrow a_7 \oplus r_0 \oplus r_5 \oplus X[t_5 \oplus r_4]$; $b_5 \leftarrow a_6 \oplus r_0 \oplus r_4 \oplus X[t_4 \oplus r_5]$;
$r_4 \leftarrow t_0 \oplus r_2$; $r_5 \leftarrow t_0 \oplus r_3$;
$b_6 \leftarrow a_5 \oplus r_1 \oplus r_4 \oplus X[t_7 \oplus r_5]$; $b_7 \leftarrow a_4 \oplus r_1 \oplus r_5 \oplus X[t_6 \oplus r_4]$;

This implementation requires 56 exors and 24 table lookups. 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.* [16, 17] 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* [16] 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 7.2 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 [17]. 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 [14, 28] will carry over to some extent; 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 [29], which takes about 500–600 gates [30].

8 Efficiency estimates

To obtain efficiency estimates we used the 32-bit implementation with eight tables described in sections 7.1 and 7.2 on a 550 MHz Pentium III processor. Because the key schedule is based on the round function itself, no extra storage is needed besides that already required for implementing encryption/decryption.

8.1 Key setup

Table 3 lists the observed key setup efficiency. The increased cost of the decryption key schedule (68% more expensive than the setup of the encryption key schedule) is due to the application of θ to $R - 1$ round keys.

Table 3. Key setup efficiency

cycles (encryption schedule)	cycles (decryption schedule)
640	1076

8.2 Encryption and decryption

Since KHAZAD has involutional structure, encryption and decryption are equally efficient (for the same number of rounds). Table 4 summarises the observed efficiency.

Table 4. Encryption/decryption efficiency

cycles per byte	cycles per block	Mbit/s
51.2	409	86.0

By coding the primitive 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.

9 Advantages

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 bears many similarities with the block cipher SHARK [26]. We now 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 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.

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

10 Acknowledgements

We are grateful to the Cryptix development team, and particularly to Raïf Naffah, for carefully reading and suggesting improvements for the implementation guidelines provided in this paper, and for implementing several versions of KHAZAD 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. 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, O. Dunkelman, N. Keller, "The Rectangle Attack - Rectangling the Serpent," *Advances in Cryptology, Eurocrypt'2001, LNCS 2045*, B. Pfitzmann, Ed., Springer-Verlag, 2001, pp. 340–357.
4. 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.
5. E. Biham and N. Keller, "Cryptanalysis of reduced variants of RIJNDAEL," submission to the *Third Advanced Encryption Standard Candidate Conference*.
6. A. Biryukov and D. Wagner, "Slide attacks," *Fast Software Encryption, LNCS 1636*, L. Knudsen, Ed., Springer-Verlag, 1999, pp. 245–259.

7. A. Biryukov and D. Wagner, "Advanced slide attacks," *Fast Software Encryption, LNCS 1807*, B. Preneel, Ed., Springer-Verlag, 2000, pp. 589–606.
8. J. Daemen, "Cipher and hash function design strategies based on linear and differential cryptanalysis," *Doctoral Dissertation*, March 1995, K.U.Leuven.
9. 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.
10. J. Daemen and V. Rijmen, "AES proposal: RIJNDAEL," AES submission (1998), `http://www.nist.gov/aes`.
11. H. Feistel, "Cryptography and computer privacy," *Scientific American*, v. 228, n. 5, 1973, pp. 15–23.
12. N. Ferguson, J. Kelsey, S. Lucks, B. Schneier, M. Stay, D. Wagner, and D. Whiting, "Improved cryptanalysis of RIJNDAEL," *Fast Software Encryption, LNCS 1978*, B. Schneier, Ed., Springer-Verlag, 2001.
13. 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.
14. T. Ichikawa, T. Kasuya, M. Matsui, "Hardware evaluation of the AES finalists," *Third Advanced Encryption Standard Candidate Conference*, NIST, April 2000, pp. 279–285.
15. 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.
16. 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.
17. P. Kocher, J. Jaffe, B. Jun, "Introduction to differential power analysis and related attacks," available from `http://www.cryptography.com/dpa/technical/`.
18. L.R. Knudsen, "Truncated and higher order differentials," *Fast Software Encryption, LNCS 1008*, B. Preneel, Ed., Springer-Verlag, 1995, pp. 196–211.
19. 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.
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. NESSIE Project – New European Schemes for Signatures, Integrity and Encryption – home page: `http://cryptonessie.org`.
24. NESSIE evaluation team, private communication, 2001; see also "Security Evaluation of NESSIE First Phase," NESSIE deliverable D13, S. Murphy and J. White, Eds., 2001 – available online at: `https://www.cosic.esat.kuleuven.ac.be/nessie/deliverables/D13.pdf`.
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. D. Whiting, "AES implementations in 0.25 micron ASIC," NIST AES electronic discussion forum posting, August 2000.
30. D. Whiting, private communication, 2001.
31. 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.
32. 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 KHAZAD S-box

KHAZAD uses the same S-box as the ANUBIS cipher (cf. [1]); we describe here how it was generated for ease of reference.

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 [10] 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 Khazad S-box.

C The name

Thus he brought me back at last to the secret ways of Khazad-dûm ...

Khazad is named after *Khazad-dûm*, "the Mansion of the Khazâd," which in the tongue of the Dwarves is the name of the great realm and city of Dwarrowdelf, of the haunted *mithril* mines in Moria, the Black Chasm. But all this should be quite obvious – unless you haven't read J.R.R. Tolkien's "The Lord of the Rings," of course `:-)`

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

Fig. 2. Boolean expressions for P and Q

Table 5. The KHAZAD 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