The Block Cipher BKSQ*

Joan Daemen[1] and Vincent Rijmen[2]

[1] PWI
Zweefvliegtuigstraat 10
B-1130 Brussel, Belgium
Daemen.J@protonworld.com
[2] Katholieke Universiteit Leuven, ESAT-COSIC
K. Mercierlaan 94,
B-3001 Heverlee, Belgium
vincent.rijmen@esat.kuleuven.ac.be

Abstract. In this paper we present a new 96-bit block cipher called BKSQ. The cipher can be implemented efficiently on a wide range of processors (including smartcards) and in hardware.

1 Introduction

We present a new 96-bit block cipher called BKSQ. The cipher has been designed according to the principles of the 'Wide trail design strategy' [2] in order to guarantee its resistance against linear and differential cryptanalysis. The structure of the cipher is a generalisation of the SQUARE cipher structure [3]. It has already been shown that SQUARE can be implemented very efficiently on a wide range of platforms. However its structure only allows block lengths of $8n^2$ bits; e.g., $n = 4$ gives a block length of 128 bits. In this paper we generalize the structure of the round transformation in order to allow block lengths of $8nm$ bits without sacrificing cryptographic strength or implementation efficiency. This is accomplished by changing the linear layer in the round transformation.

BKSQ is especially suited to be implemented on a smart card. Its block length of 96 bits allows it to be used as a (2nd) pre-image resistant one-way function. Most available block ciphers have a block length of 64 bits, but this block length is currently perceived as being too small for a secure one-way function. The next option in currently available block ciphers is a block length 128 bit, but this leads to one-way functions that are significantly slower. The block cipher BKSQ is tailored towards these applications. Still, it can also be used for efficient MACing and encryption on a Smart Card.

The remainder of this paper is organized as follows. In Section 2 we explain the structure of BKSQ and introduce the various components. In Section 3 we discuss the resistance of the cipher against various cryptanalysis methods. We conclude with a discussion of some implementation aspects in Section 4.

* This research has been sponsored by PWI.

2 Structure of BKSQ

BKSQ is an iterated block cipher with a block length of 96 bits and a key length of 96, 144 or 192 bits. The round transformation of BKSQ is *not* a Feistel network. In a typical Feistel network a part of the input bits of the round is transposed unchanged to another position at the output. This is not the case with BKSQ, where every bit of the input is treated in the same way. We call this type of round transformation a *uniform round transformation*. On the conceptual level, the round transformation is composed of four uniform invertible transformations. In an efficient implementation the transformations can be combined in a single set of table-lookups and exor operations.

Let the input of the cipher be denoted by a string of 12 bytes: $p_0 p_1 \dots p_{11}$. These bytes can be rearranged in a 3×4 array, or 'state' a.

$$a = \begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \end{bmatrix} = \begin{bmatrix} p_0 & p_3 & p_6 & p_9 \\ p_1 & p_4 & p_7 & p_{10} \\ p_2 & p_5 & p_8 & p_{11} \end{bmatrix}$$

The basic building blocks of the cipher operate on this array. Figure 1 gives a graphical illustration of the building blocks.

2.1 The Linear Transformation θ

θ is a linear operation that operates separately on each of the four columns of a state. We have

$$\theta(a) = \begin{bmatrix} 3 & 2 & 2 \\ 2 & 3 & 2 \\ 2 & 2 & 3 \end{bmatrix} \cdot \begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \end{bmatrix} .$$

Both addition and multiplication are performed in the finite field $\mathrm{GF}(2^8)$. This means that the addition operation corresponds to the bitwise exor and multiplication of two bytes is defined as a modular multiplication of polynomials with binary coefficients [10].

By defining the operations over a finite field and a careful choice of the coefficients of θ we can guarantee a high resistance against linear and differential cryptanalysis, following the design principles of the 'wide trail design strategy' [2, 13]. Furthermore, this choice for the coefficients makes it possible to implement θ very efficiently on an 8-bit processor with limited RAM.

2.2 The Nonlinear Transformation γ

γ is a nonlinear byte substitution, identical for all bytes. We have

$$\gamma : b = \gamma(a) \Leftrightarrow b_{i,j} = \mathrm{S}_\gamma(a_{i,j}),$$

with S_γ an invertible 8-bit substitution table or S-box. The inverse of γ consists of the application of the inverse substitution S_γ^{-1} to all bytes of a state.

The design criteria for S_γ are minimal correlation between linear combinations of input bits and linear combinations of output bits, and a minimal upper bound for the entries in the exor-table (not counting the trivial (0,0) entry). For the construction of the S-box we started from the inverse mapping over GF(2), as explained in [12]. This mapping has very good resistance against linear and differential cryptanalysis. Subsequently we applied an affine transformation (over GF(2)) to the output bits, with the property that it has a complicated description in GF(2^8) in order to thwart interpolation attacks [5]. We ensured that the S-box has no fixed points ($S_\gamma[x] = x$) and no 'opposite fixed points' ($S_\gamma[x] = \overline{x}$). The input-output correlations of this S-box are upper bounded by 2^{-3} and the entries of the exor-table are upper bounded by 4.

2.3 The Byte Permutation π

The effect of π is a shift of the rows of a state. Every row is shifted a different amount. We have

$$\pi : b = \pi(a) \Leftrightarrow b_{i,j} = a_{i,j-i}.$$

The effect of π is that for every column of a the three elements are moved to three different columns in $\pi(a)$. Such a byte permutation is called 'diffusion-optimal' with respect to θ and it enhances the effect of the linear transformation θ.

2.4 Bitwise Round Key Addition σ

$\sigma[k^t]$ consists of the bitwise addition of a round key k^t. We have

$$\sigma[k^t] : b = \sigma[k^t](a) \Leftrightarrow b = a \oplus k^t.$$

The inverse of $\sigma[k^t]$ is $\sigma[k^t]$ itself.

2.5 The Cipher BKSQ

The building blocks are composed into the round transformation denoted by $\rho[k^t]$:

$$\rho[k^t] = \sigma[k^t] \circ \pi \circ \gamma \circ \theta \tag{1}$$

BKSQ is defined as R times the round operation, preceded by a key addition $\sigma[k^0]$ and by θ^{-1}:

$$\text{BKSQ}[k] = \rho[k^R] \circ \rho[k^{R-1}] \circ \ldots \circ \rho[k^2] \circ \rho[k^1] \circ \sigma[k^0] \circ \theta^{-1} \tag{2}$$

The number of rounds depends on the key length that is used. For 96-bit keys, there are 10 rounds; for 144-bit keys, there are 14 rounds and for 192-bit keys, the number of rounds is 18.

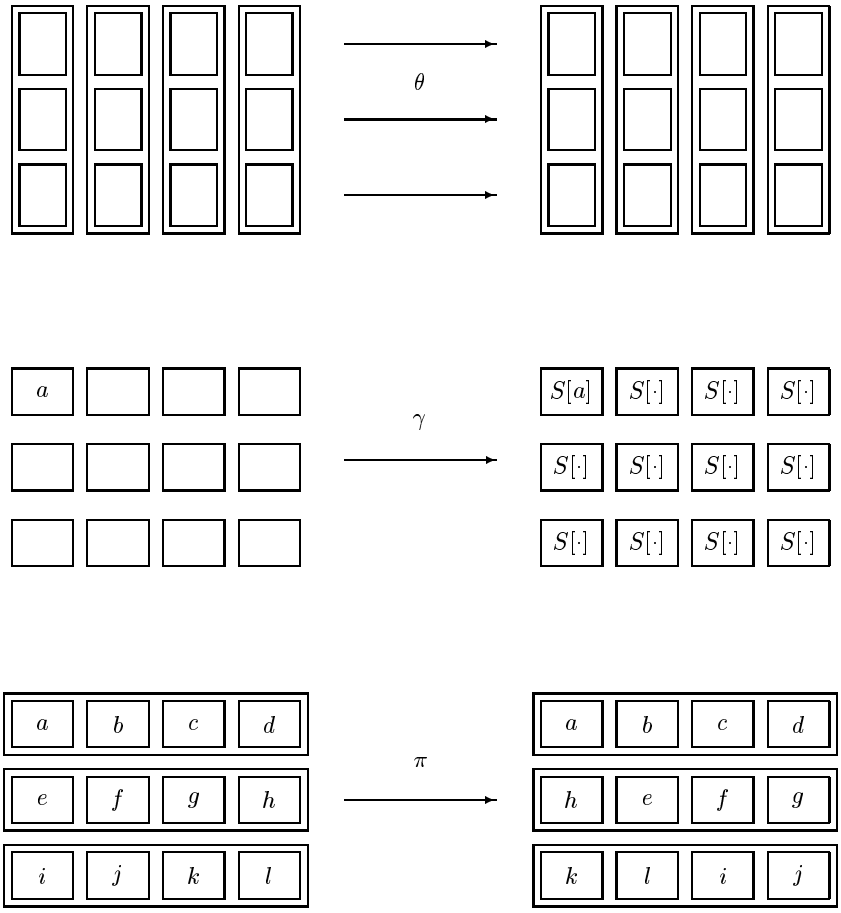

Fig. 1. Graphical illustration of the basic operations of BKSQ. θ consists of 4 parallel linear diffusion mappings. γ consists of 12 separate substitutions. π is a shift of the rows.

2.6 The Key Scheduling

The derivation of the round keys k^t from the cipher key K depends on the length of the cipher key. The key scheduling can be described as follows. The round keys have always a length of 96 bits. The cipher key K has a length of 96, 144 or 192 bits. All these lengths are multiples of 24, and all keys will be considered as an array of 24-bit columns. A round key has 4 columns, a cipher key has L columns, where $L = 4, 6$ or 8. We define an array w, consisting of $4(R+1)$ 24-bit columns, where R is the number of rounds in BKSQ. The array is constructed by repeated application of an invertible nonlinear transformation ψ: the first L columns are the columns of K, the next L are given by $\psi(K)$, the following columns are given by $\psi(\psi(K))$, etc.

The round keys k^t are extracted in a simple way from w:

$$k^t = \left[w_{4t}\ w_{4t+1}\ w_{4t+2}\ w_{4t+3} \right] .$$

Note that this key scheduling can be implemented without explicit use of the array w. If the application requires that the implementation uses a minimal amount of RAM, it is possible to calculate the round keys during the round operation, thereby only requiring storage of L 24-bit columns.

2.7 The Round Key Evolution ψ

The transformation ψ is defined in terms of the exor operation, a byte-rotation rot that rotates the bytes of a column,

$$\mathsf{rot}\left(\begin{bmatrix} a \\ b \\ c \end{bmatrix} \right) = \begin{bmatrix} b \\ c \\ a \end{bmatrix} ,$$

and a nonlinear substitution γ' that operates in exactly the same way as γ, but takes as argument column vectors instead of arrays:

$$\gamma'\left(\begin{bmatrix} a \\ b \\ c \end{bmatrix} \right) = \begin{bmatrix} S_\gamma(a) \\ S_\gamma(b) \\ S_\gamma(c) \end{bmatrix} .$$

The transformation ψ operates on blocks of L columns. For key lengths of 96 or 144 bits ($L = 4$ or 6), ψ is given by:

$$\psi([a_0 \ldots a_{L-1}]) = \left[b_0 \ldots b_{L-1} \right] \Leftrightarrow \begin{cases} b_0 = a_0 \oplus \gamma'(\mathsf{rot}(a_{L-1})) \oplus C_t \\ b_i = a_i \oplus b_{i-1} \qquad i = 1, 2, \ldots, L-1. \end{cases}$$

For a key length of 192 bits ($L = 8$), an extra nonlinearity is introduced and ψ is given by:

$$\begin{aligned} \psi([a_0 \ldots a_{L-1}]) \ &= \ \left[b_0 \ldots b_{L-1} \right] \\ &\Leftrightarrow \begin{cases} b_0 \ = a_0 \oplus \gamma'(\mathsf{rot}(a_{L-1})) \oplus c_t & \\ b_i \ = a_i \oplus b_{i-1} & i = 1, 2, \ldots, L/2 \\ b_{L/2} = a_{L/2} \oplus \gamma'(b_{L/2-1}) & \\ b_i \ = a_i \oplus b_{i-1} & i = L/2+1, \ldots, L-1. \end{cases} \end{aligned}$$

The vectors c_t have as functionality to remove the symmetry in the transformation. They are given by:

$$c_t = \begin{bmatrix} d_t \\ 0 \\ 0 \end{bmatrix},$$

where $d_0 = 1$ and $d_{t+1} = 2 \cdot d_t$. Multiplication is done in the Galois field (modulo the same polynomial as for γ and θ).

2.8 The Inverse Cipher

The structure of BKSQ lends itself to efficient implementations. For a number of modes of operation it is important that this is also the case for the inverse cipher. Therefore, BKSQ has been designed in such a way that the structure of its inverse is equal to that of the cipher itself, with the exception of the key schedule. Note that this identity in *structure* differs from the identity of *components and structure* in IDEA [9].

By means of a mathematical derivation, very similar to the one given in [3], it can be shown that the inverse cipher is equal to the cipher itself with γ, θ and π replaced by γ^{-1}, θ^{-1} and π^{-1} respectively and with different round key values.

3 Cryptanalysis

We discuss the resistance of BKSQ against cryptanalytic attacks.

3.1 Linear and Differential Cryptanalysis

BKSQ has been designed according to the 'wide trail design strategy,' introduced by J. Daemen [2]. This strategy is used to guarantee a low maximum probability of differential trails and a low maximum correlation of linear trails.

The S-box is selected such that the maximal entry in the exor-table equals four, and the maximal correlation equals 2^{-3}.

The choice of the coefficients of θ gives it a branch number of four, which means that every two-round characteristic or linear approximation has at least four active S-boxes. The choice of π ensures that every four-round characteristic or linear approximation has at least 16 active S-boxes.

We wrote a program to count the minimum number of active boxes in an n-round characteristic or linear approximation, for n going up to 20. It turns out that for $n \geq 4$, there are at least $4n$ active S-boxes in every n-round differential characteristic and every n-round linear approximation

3.2 Truncated Differentials

The concept of truncated differentials was first published by L.R. Knudsen [7]. The idea can be summarized by stating that we don't look at the specific difference that the individual bytes have, but only take into account whether the bytes in position (i, j) are equal or not. Using truncated differentials, it is possible to cryptanalyse versions of BKSQ that are reduced to seven rounds or less. The attack is based on the fact that a (truncated) difference of the following form:

$$\begin{bmatrix} x & 0 & y & 0 \\ 0 & 0 & 0 & 0 \\ z & 0 & u & 0 \end{bmatrix},$$

at the input of a round transformation, goes with probability 2^{-16} to an output difference of the same form. This one-round characteristic can be concatenated with itself to produce a multi-round characteristic. A more detailed description of this attack is given in Appendix A. It seems not possible to cryptanalyse more than seven rounds with this attack.

3.3 The SQUARE Attack

In [3] a new attack is described, that works for up to six rounds of SQUARE. Because of the similarity between the round transformations of BKSQ and SQUARE, the same attack also applies here. Instead of repeating the description of the attack, we list the complexities for this attack on BKSQ in Table 1. While the requirements of the attack are smaller, it seems not possible to extend the attack with an additional round.

Table 1. Complexity of the SQUARE attack when applied to BKSQ.

Attack	# Plaintexts	Time	Memory
4-round	2^9	2^9	small
5-round	2^{11}	2^{32}	small
6-round	2^{24}	2^{56}	2^{24}

3.4 Timing Attacks

A class of timing attacks is described by Kocher in [8]. The underlying principle of the attacks is that if the execution time of an implementation depends on secret key bits, then a cryptanalyst can deduce information about these key bits. An implementation can be protected against timing attacks by removing all branches that are conditional on key bits. This can easily be done for BKSQ (cf. Section 4).

4 Implementation Aspects

The only operations required to implement BKSQ are exor, one-bit shifts and table-lookups. Therefore the cipher can be implemented efficiently on a wide range of processors and in hardware (no carry delays).

On 32-bit processors, the linear operations and the nonlinear substitution can be combined into a single set of table-lookups and exor operations. The key addition exists of a simple exor operation. This implementation requires about 1 kbyte of RAM for the tables.

On 8-bit processors, other optimisations are possible. The byte permutation can be combined easily with the nonlinear substitution and the key addition. Typically RAM is scarce on 8-bit processors. In that case the additions (exors) and multiplications of the transformation θ will be implemented explicitly. This observation has influenced the choice for the coefficients of θ.

Multiplication with 2 can be implemented with one shift and a (conditional) reduction (i.e., an exor). However, in order to keep the implementation resistant against timing attacks, we implement this multiplication with a table lookup. This requires an additional 256 bytes of ROM. Multiplication with 3 can be implemented as a multiplication with 2 and an addition ($3x = 2x \oplus x$).

The design of the key scheduling facilitates 'just-in-time' calculation of the round keys, a necessity when the processor has not enough RAM to store all the round keys in advance.

A timing attack resistant implementation of BKSQ on the Motorola 68HC08 microprocessor fits easily in less than 1kbyte of ROM, requires 28 bytes of RAM and takes less than 7000 cycles (for the 10-round version).

References

1. E. Biham and A. Shamir, "Differential cryptanalysis of DES-like cryptosystems," *Journal of Cryptology*, Vol. 4, No. 1, 1991, pp. 3–72.
2. J. Daemen, "Cipher and hash function design strategies based on linear and differential cryptanalysis," *Doctoral Dissertation*, March 1995, K.U.Leuven.
3. 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. Also available as `http://www.esat.kuleuven.ac.be/~rijmen/square/fse.ps.gz`.
4. J. Daemen and V. Rijmen, "Linear frameworks for block ciphers," *COSIC internal report 96-3*, 1996.
5. 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.
6. J. Kelsey, B. Schneier and D. Wagner, "Key-schedule cryptanalysis of IDEA, G-DES, GOST, SAFER, and Triple-DES," *Advances in Cryptology, Proceedings Crypto'96, LNCS 1109*, N. Koblitz, Ed., Springer-Verlag, 1996, pp. 237–252.
7. L.R. Knudsen, "Truncated and higher order differentials," *Fast Software Encryption, LNCS 1008*, B. Preneel, Ed., Springer-Verlag, 1995, pp. 196–211.
8. P.C. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS and other systems," *Advances in Cryptology, Proceedings Crypto'96, LNCS 1109*, N. Koblitz, Ed., Springer-Verlag, 1996, pp. 146–158.

9. X. Lai, J.L. Massey and S. Murphy, "Markov ciphers and differential cryptanalysis," *Advances in Cryptology, Proceedings Eurocrypt'91, LNCS 547*, D.W. Davies, Ed., Springer-Verlag, 1991, pp. 17–38.
10. R. Lidl and H. Niederreiter, *Introduction to finite fields and their applications*, Cambridge University Press, 1986.
11. M. Matsui, "Linear cryptanalysis method for DES cipher," *Advances in Cryptology, Proceedings Eurocrypt'93, LNCS 765*, T. Helleseth, Ed., Springer-Verlag, 1994, pp. 386–397.
12. K. Nyberg, "Differentially uniform mappings for cryptography," *Advances in Cryptology, Proceedings Eurocrypt'93, LNCS 765*, T. Helleseth, Ed., Springer-Verlag, 1994, pp. 55–64.
13. V. Rijmen, "Cryptanalysis and design of iterated block ciphers," *Doctoral Dissertation*, October 1997, K.U.Leuven.

A The Truncated Differential Attack

Using truncated differentials, reduced versions of BKSQ can be cryptanalysed. Up to seven rounds can be broken using the following approach. First we explain a one-round truncated differential, afterwards we describe how this can be used in an attack.

A.1 A One-Round Truncated Differential

Let a and b be two states, where $a_{i,j} = b_{i,j}$ for all (i,j), except for (i,j) equal to $(0,0)$, $(0,2)$, $(2,0)$, or $(2,2)$. We say then that $a \oplus b$ is an α-difference. Let $c = \theta(a)$ and $d = \theta(b)$. Since θ operates independently on every column, we have that

$$c_{i,j} = d_{i,j} \qquad i = 0,1,2; j = 1,3.$$

Our choice for the c_j coefficients ensures that in the first and the third column ($j = 0,2$) at least two of the three bytes differ between c and d. More precise, we have for $j = 0,2$:

$$c_{i,j} \neq d_{i,j} \qquad i = 0,2;$$

and

$$c_{1,j} = d_{1,j} \Leftrightarrow a_{0,j} \oplus b_{0,j} = a_{2,j} \oplus b_{2,j}. \tag{3}$$

Equation (3) shows that with probability $(2^{-8})^2$, an α-difference at the input of θ will go to an α-difference at the output.

Since γ and $\sigma[k^t]$ operate on the independent bytes, every truncated difference remains unchanged with probability one. Since π leaves the first row of a state unchanged, and shifts the third row by two positions, an α-difference at the input goes to an α-difference at the output with probability one.

We conclude that an α-difference at the input of a round transformation $\rho[k]$ goes to an α-difference at the output with probability 2^{-16}.

A.2 Constructing a Multiple-Round Truncated Differential

The one-round differential from the previous section can be concatenated. It can also be preceded and followed by a few special rounds with enhanced probability.

For the end rounds, we consider what happens when (3) is not followed: with a probability close to $1 - 2^{-7}$, an alpha difference at the input of a round will go to an output difference of the following form:

$$\begin{bmatrix} x_1 & 0 & x_2 & 0 \\ 0 & x_3 & 0 & x_4 \\ x_5 & 0 & x_6 & 0 \end{bmatrix} . \tag{4}$$

This difference is still highly structured: in the second an fourth column there is only one byte different from zero. This will be visible in the output of the next round, where the differences will show a specific relation. We conjecture that a differential attack will be possible if (4) is used in the last but one round. In the first rounds we can enhance the probability of our differential by guessing some key bits.

Summarising, we can attack seven rounds in the following way. We guess two bytes of information about the key. If we guess right we can construct pairs with an α-difference that pass the first two rounds of the truncated differential with probability one. The next three rounds are passed with probability 2^{-16} each. In the sixth round, we let the difference spread out (probability close to one) and in the seventh round we detect and/or verify the remaining structure. The probability of the characteristic is about 2^{-48}.

Note that it is not possible to extend this characteristic by adding a fourth $(\alpha \to \alpha)$-round. The probability that a random pair will produce an α-difference at the input of the last round is also given by 2^{-48} (6 bytes should be zero). Therefore it makes no sense to consider differential characteristics that will result in this difference with a lower probability. It might even be that the signal-to-noise ratio of the seven-round attack is already too low.