Kaweichel, an Extension of Blowfish for 64-Bit Architectures

Dieter Schmidt*

September 30, 2006

Abstract

In this article the block cipher Kaweichel is presented. It is an extension of Blowfish for 64-bit architectures. Its aim is to present a cipher for modern 64-Bit processors which utilizes commonplace instructions. A main objective of the development was to harden the cipher against known attacks on Blowfish. The author does not claim intellectual property on Kaweichel and the cipher will remain unpatented. A C reference implementation is available on the web.

1 Introduction

Measures to protect privacy are today virtually everyone’s business. Since the mid-seventies, when open research in cryptography began, numerous encryption methods have been published. For connections with low error rates or for fault-tolerant protocols, like the Internet Protocol, so called block ciphers are the first choice for encryption routines. Block ciphers encrypt a given number (typically a power of 2 and 64 or higher) of bits simultaneously under the control of a key. Opposite to stream ciphers, where under the control of a key a pseudorandom sequence is generated and XORed (addition modulo 2) with the plaintext, the ciphertext of a block cipher depends on all the bits of the plaintext. As a result, in general the breaking of a block cipher is more difficult than the breaking of a stream cipher with the same key length. This is called diffusion, i.e. the spreading of one bit of the plaintext to all the bits of the ciphertext and was first proposed by Shannon [9]. Shannon also proposed the so called confusion, i.e.

*Denkmalstrasse 16, D-57567 Daaden, Germany, dieterschmidt@usa.com

1This article is the edited English translation of [4]
a bit of the ciphertext should depend in a complex manner on the
bits of the plaintext and the key. Modern block ciphers realize these
demands as a necessary, but by no means sufficient condition for a
secure encryption. Given the development by open cryptography since
1975, additional criteria have been developed, such as immunity from

This paper is an attempt to develop a software-efficient and secure
block cipher with the instructions typical for a modern 64-bit micro-
processor (possible candidates are Alpha, G5, Hammer and Itanium
and others). The starting point of the development is the block ci-
pher Blowfish [8], which is free of claims of intellectual property. The
typical instructions of a 64-bit microprocessor, that are used, are the
loading of a register, addition, XOR, AND, rotation to the right and
shift to the right. Given favourable circumstances, that is a level-1
cache hit for program code and data, these instructions are carried
within one or two clock cycles.

2 Definitions
Let denote: $\oplus$ additon modulo $2^{64}$, $\oplus$ addition modulo 2 (XOR) of two
64-bit words, rotate the rotation of a 64-bit word by e positions to the
right. This rotation can be written as multiplication $2^{64-\epsilon} \mod 2^{64} - 1$
if the values of the 64-bit word is less then $2^{64} - 1$. S-box denotes a
8-bit to 64-bit substitutions-box, which can be expressed as indexed
addressing of 8-bit values into a table of 64-bit values.

3 Description of the block cipher
Kaweichel is a generalized Feistel cipher. The first Feistel cipher pub-
lished was the Data Encryption Standard (DES) of the U.S. govern-
ment, which was based on work by IBM. It is published in [1, 2].

When using a Feistel cipher the plaintext is first divided into two
equal halves. The size of the plaintext block is typically 64 bit or 128
bit, i.e. it is a power of 2. The left half of the plaintext block is used as
the input for a so called round function (F-function), which modifies
the input under the control of the round key. The output of the round
function is added modulo 2 (XORed) to the right half of the plaintext
block. After that, the two halves are exchanged and the procedure is
repeated, until the defined number of iterations (rounds) is reached.
After the last iteration, the two halves of the ciphertext block are
not exchangend. Thus the whole construction is self-inverse except
for the order of the round keys. This means that for encryption and
decryption the same hard- and software can be used, only the order of the round keys has to be inverted for decryption.

For the construction of the round function one chooses usually parallel substitutions (s-boxes). The output bits of these s-boxes are permuted in order to achieve diffusion. For the derivation of the round keys from the user key one has to choose a key schedule.

The basic idea behind this construction is that a weak, iterated encryption function will result in a cryptographically strong cipher. But there minimum requirements for the round function (F-function). It should, for example, offer sufficient resistance against differential [2] and linear cryptanalysis [5].

The construction of Kawiechel (see figures 1 and 2) differs in several points from that of a classical Feistel ciphers.

- Before the left data block is used as input for round function, a key $P_i$ is added modulo $2^{64}$ to that data block.
- Rather then using a round key for the round function, the s-boxes are key dependant. This method got first widely known with the block cipher Blowfish [8]. The advantage is, that differential [2] and linear cryptanalysis [5] are not applicable, since they require the knowledge of the s-boxes.
- After the output of the round function is added modulo 2 (XORED) to the right data block, the bits of the right block are rotated to the right by a fixed amount.
- The right and left halves are exchanged after the last iteration.
- After the last iteration a pair of keys $P_{32}, P_{33}$ is added modulo $2^{64}$ to both halves (final transformation).

Each of the points 1 and 3 to 5 causes the cipher not to be self-inverse, i.e. for encryption and decryption separate hard- and software needs to be implemented.

Kawiechel works with a block size of 128 bit, thus each half is 64 bit long. The 64 bit input to the round function is first divided into eight equal units of eight bit (one byte). Each of these units is used as an index into a table of $256 = 2^8$ values (s-box) in such a manner, that the least significant byte is used as input for s-box 0, the next byte for s-box 1, and so on, and the most significant byte is used as input for s-box 7. The 64 bit results of two adjacent s-boxes are added modulo $2^{64}$, e.g. the outputs of s-box 0 and s-box 1. This leaves four values. Two adjacent of these four values are added modulo 2 (XORED). This leaves two 64 bit values. They are added modulo $2^{64}$ to form the
Figure 1: One iteration of the block cipher Kaweichel

Figure 2: The final transformation
output of the round function. If one denotes the output of the s-boxes by \( S_0 \) to \( S_7 \), the output of the round function becomes:

\[
round\text{output} = ((S_0 \oplus S_1) \oplus (S_2 \oplus S_3)) \oplus ((S_4 \oplus S_5) \oplus (S_6 \oplus S_7))
\]

The combination of the output of the s-boxes was changed compared to Blowfish [8] to allow for a better parallelization in both hard- and software and to make Rijmen's attack [6] more difficult.

For the number of rounds the author recommends for the time being \( N = 32 \). The maximum key length is thus 30 (i.e. \( N - 2 \)) words of 64 bits or 1920 bits. Shorter keys are appended with zeros to reach 1920 bit, but the length of key should not fall below 256 bits.

For the rotation the value 11 is used.

For the derivation of the round keys, the keys for the final transformation and the s-boxes, the following holds: First the round keys and the keys for the final transformation are assigned random or pseudo-random values. After that the s-boxes are assigned random or pseudo-random values, beginning with s-box 0 and Index (for details, see the function init_cipher in the reference implementation). In the reference implementation (see [7]) the binary digits of \( \pi \) (less the initial 3) are used for that purpose. After that, the 1920 bit long userkey is added modulo 2 (XORed) to the first 30 roundkeys \( P_i, i = 0 \ldots 29 \). This limitation of the userkey ensures that in the following encryptions all outputbits depend on all the bits of the userkey. After that the plaintext block is assigned the all-zero string and encrypted once. The round keys \( P_0 \) and \( P_1 \) are then assigned the right half and the left half of the ciphertext block. The cipher is then employed in Output Feedback Mode (OFB) and the values generated are assigned the next round keys \( P_2 \) and \( P_3 \). This is repeated, until all the round keys, the keys of the final transformation and the s-boxes have been assigned new values (see function expand_key of the reference implementation).

4 Security

The individual output bits of a s-box can be described as Boolean function of the eight input bits. Since the s-box entries depend on the key and the initial values they are random. Thus a certain output bit of a s-box can be described as a random Boolean function of the eight input variables. If one selects a Boolean function with eight arguments, there are \( 2^{256} \) possibilities with \( 256 = 2^8 \). The share of affine Boolean functions, which are especially unsuitable for cryptographic purposes, decreases rapidly as the number of arguments increases. On the other hand, wider input sizes of the s-boxes mean an increase in memory requirements, so that a satisfactory compromise had to be reached. The following table gives the share \( A \) of affine Boolean
functions and the memory requirement $S$ of the round function as a function of the input width $m$ of the s-boxes.

<table>
<thead>
<tr>
<th>(m/\text{bit})</th>
<th>(A)</th>
<th>S/kbyte</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>(2^{-1})</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>(2^{-11})</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>(2^{-247})</td>
<td>16</td>
</tr>
<tr>
<td>16</td>
<td>(2^{-65519})</td>
<td>2048</td>
</tr>
</tbody>
</table>

Given the limited size of the level-1 data cache in modern 64-bit processors the choice of $m=8$ is deemed feasible as well as sufficiently secure.

Another point of interest is the probability that a certain output bit of the round function is a constant (The possibility that a carry occurs in the five additions modulo \(2^6\) is neglected). To this end consider the algebraic normal form (ANF) of a Boolean function. A Boolean function with eight arguments has \(256=2^8\) coefficients in the ANF, which all can take the values 0 or 1. For an output bit of the round function to be a constant, the Boolean functions of two s-boxes must correspond in the last 255 coefficients of the ANF. This means, that for four s-boxes the coefficients of the ANF are free while the last 255 coefficients of the other four s-boxes are fixed. Thus the probability that a certain output bit of the round function is a constant is 

\[w = (2^{255})^4 = 2^{1020}.\]

If one compares this to the key length of 1920 bit, the value seems to high. For this reason the rotation was introduced. Now each bit of one of the halves is combined modulo 2 (XORed) with 16 different output bits of the round function. The probability that any 16 output bits of the round function are constant is:

\[W = (16!)^{-1} \times 64 \times 2^{-1020} \times 63 \times 2^{-1020} \times \ldots \times 49 \times 2^{-1020} < 2^{-44} \times 64^{16} \times (2^{-1020})^{16} = 2^{52} \times 2^{-16320} = 2^{-16268}\]

Another important point that must be considered when designing a modern block cipher is security from differential and linear cryptanalysis. Both methods are not applicable to Kaweichel, since the s-boxes are newly determined with each key.

Vincent Rijmen published in [6] an analysis of Blowfish with second order differentials, that breaks four of the 16 rounds. His attack is not applicable to Kaweichel, since it requires that the operations performed on the two halves commute. This is not the case, because addition modulo \(2^6\) and XOR do not commute. In addition to that, his attack calculates the s-boxes by approximating the addition in the round function by XOR. In Kaweichel for each s-box two additions have to be approximated by XOR, making the attack more difficult.

Another analysis of Bowfish can be found in the article by Serge Vaudenay [10]. His attacks are not applicable to Kaweichel, since
the rotation distributes the input difference to other s-boxes then the intended one.

5 Miscellaneous

The block cipher presented in this article is suitable for 64-bit processors with level-1 data cache larger than 16kbyte. For computers with limited resources like smartcard it is not recommended. The main area of use is bulk encryption of data. The key agility is rather poor. If one Megabyte of data is encrypted, the key expansion will use 1.5 % of the total time.

The author claims no intellectual property on Kaweichel and the cipher will remain unpatented. A C reference implementation is available from [7].

6 Acknowledgements

The author thanks Claus Grupen of Siegen University for continued encouragement and support.

References


