# PAGES+, PAGES-, and PAGES– - Three Families of Block Ciphers

Dieter Schmidt[1]

October 4, 2015

[1]Dieter.Schmidt.37@t-online.de

## Abstract

PAGES+ is a family of block ciphers based on block ciphers Speck [2] and PAGES [9]. The block length was increased vom 128 bit to 512 bit and additional rounds were introduced to make the cipher more resistent against attacks. The number of rounds are 64, 96, and 128. The key size are 1024 bit, 1536 bit, and 2048 bit, respectively. The size of variables, as with PAGES, is 128 bit. Thus the variables can be stored in registers of the microprocessors of two leading suppliers. As with Speck, PAGES+ utilizes instructions with a low latency, such as addition modulo $2^{128}$, subtraction modulo $2^{128}$, XOR, and circular shifts (rotation). All these instructions are usually carried out within a few cycles. Hence despite the number of rounds is considerable, PAGES+ has a high software throughput. For a processor with a frequency of 2.5 GHz, the software throughput with the optimized version of the reference implementation is 30 megabyte per second with a key length of 2048 bit and a number of rounds of 128. In hardware or the implementation on a FPGA a considerable performance is expected, yet with a limited expense.

PAGES- is a family of block ciphers based on block ciphers Speck [2] and PAGES [9]. The block length was increased vom 128 bit to 256 bit and additional rounds were introduced to make the cipher more resistent against attacks. The number of rounds are 32, 48, 64, 96, and 128. The key size are 256 bit, 384 bit, 512 bit, 768 bit, and 1024 bit, respectively. The size of variables, as with Speck, is 64 bit. As with Speck, PAGES- utilizes instructions with a low latency, such as addition modulo $2^{64}$, subtraction modulo $2^{64}$, XOR, and circular shifts (rotation). All these instructions are usually carried out within one cycle. Hence despite the number of rounds is considerable, PAGES- has a high software throughput. For a processor with a frequency of 2.5 GHz, the software throughput with the optimized version of the reference implementation is 80 megabyte per second with a key length of 1024 bit and a number of rounds of 128.

PAGES– is a family of block ciphers based on block ciphers Speck [2] and PAGES [9]. The block length was increased vom 128 bit to 512 bit and additional rounds were introduced to make the cipher more resistent against attacks. The number of rounds are 32, 48, 64, 96, and 128. The key size are 512 bit, 768 bit, 1024 bit, 1536 bit, and 2048 bit respectively. The size of variables, as with Speck, is 64 bit. For a processor with a frequency of 2.5 GHz, the software throughput with the optimized version of the reference implementation is 65 megabyte per second with a key length of 2048 bit and a number of rounds of 128.

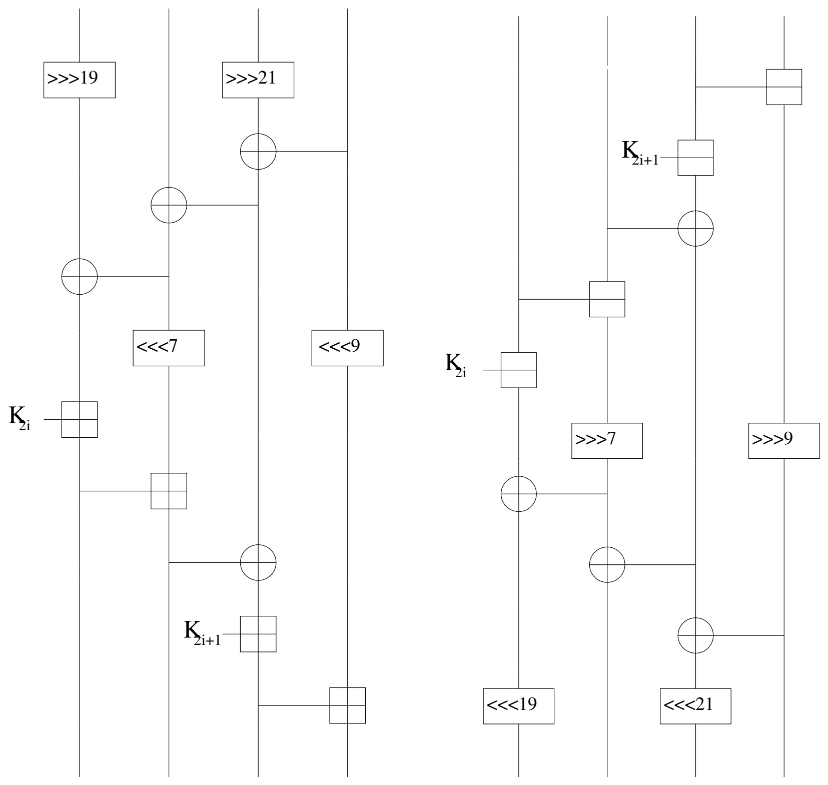
## 1 Introduction

Shannon [11] introduced diffusion into cryptography. Diffusion means that a symbol of the ciphertext depends on as much symbols of the plaintext. This is one of his experiences of cryptosystems in the Second World War, where usually one letter of plaintext was mapped on one letter of ciphertext. The first one open cryptosystem that used diffusion, was the block cipher Lucifer from Horst Feistel [3]. Lucifer is an iterated substitution-permutation-network (see e.g. [10]). In addition, there are Feistel-Networks, where the so called F-function maps on part of the cipher to another (see e.g [10]). The F-function can be a substitution-permutation-network, for example the Data Encryption Standard (DES) (see e.g. [10]). The substitution is usually achieved with a so called s-box. The s-box is several bits wide und maps usually one value to another. The s-box must be bijective. This is not requirement for the F-function. The permutation maps the bits onto the s-boxes of the next layer.

Virtually all block ciphers are iterated cryptosystems.You carry out a weak cryptographic transformation (a so called round) several times to achieve a strong cryptographic transformation. PAGES+, PAGES-, and PAGES– are iterated cryptosystems, too, but not a substitution-permutation-network and not a Feistel-cipher. Rather there are incompatible instructions, like addition, subtraction, XOR, and rotation. The necessary confusion (this was minted, too, by Shannon, see [11]. Confusion means that the ciphertext dependance on the plaintext and the key is rather complicated.) is achieved through addition und subtraction, which are non-linear over $\mathbb{F}_2^{128}$ or $\mathbb{F}_2^{64}$ (see below). The carry propagation makes the two operation non-linear. Addition and XOR are very similar. The consequence is that the number of rounds must be increased to obtain a strong cryptographic transformation. The increased cost of calculation is compensated by the fact that addition, subtraction, XOR, and rotation are instructions that has a latency of a few cycles. For microprocessor clocked with a frequency of 2.5 GHz the throughput of PAGES+ is 30 megabyte per second with a key size of 1024 bit and 128 rounds. The respective values for PAGES- and PAGES– are 80 megabytes per second and 65 megabytes per second. This holds for the general purpose register of the processor. For PAGES- and PAGES– there is a possibility to execute the algorithms in the SSE unit (see [2]). This results in increased performance, since several encryptions can be done in parallel.

The key is not introduced in the block cipher by XOR, but added modulo $2^{128}$ and modulo $2^{64}$. This results in an increased resistence against linear cryptanalysis (see [4, 5, 9, 12]).

The rest of the paper is organized as follows: Section 2 presents the assembly of PAGES+, the key schedule, and the software performance. Section 3 governs the assembly of PAGES- and Section 4 specifies the assembly of PAGES–. We conclude in Section 5.



(a) Encryption (b) Decryption

Figure 1: A round of PAGES+

## 2 Assembly of PAGES+

### 2.1 Preliminaries

$\mathbb{F}_2^{128}$ denotes the 128 dimensional vector space over GF(2). The following symbols will be used to construct the block cipher PAGES+: $\boxplus$ denotes addition modulo $2^{128}$, $\boxminus$ denotes subtraction modulo $2^{128}$, $\oplus$ denotes addition modulo 2 in $\mathbb{F}_2^{128}$ (XOR), $\lll m$ denotes the rotation to the left by m bits of a 128 bit variable, $\ggg m$ denotes the rotation to the right by m bits of a 128 bit variable. PAGES+ has a block size of 512 bit and is divided into four equal quarters of 128 bit. The key size und the rounds are as follows: 1024 bit and 64 rounds, 1536 bit and 96 rounds, and 2048 bit and 128 rounds. $B$ denotes a block of size 512 bit, and $b_i, 0 \leq k \leq 4$ denotes a block of 128 bit.

### 2.2 Encryption

A block $B$ of size 512 bit is divided into four equal subblocks of 128 bit, i.e. $B = b_0 \| b_1 \| b_2 \| b_3$, see also Figure 1 and the function encrypt in the reference implementation in appendix A. The even subblocks are rotated to the right:

$$b_0 = b_0 \ggg 19 \tag{1}$$
$$b_2 = b_2 \ggg 21 \tag{2}$$

The subblocks $b_2$ to $b_0$ are XORed with the subblocks to the right:

$$b_2 = b_2 \oplus b_3 \tag{3}$$
$$b_1 = b_1 \oplus b_2 \tag{4}$$
$$b_0 = b_0 \oplus b_1 \tag{5}$$

Now the odd subblocks are rotated to the left:

$$b_1 = b_1 \lll 7 \tag{6}$$
$$b_3 = b_3 \lll 9 \tag{7}$$

Now the following operations are carried out ($i$ is value of rounds):

$$b_0 = b_0 \boxplus K_{2*i} \tag{8}$$
$$b_1 = b_1 \boxplus b_0 \tag{9}$$
$$b_2 = b_2 \oplus b_1 \tag{10}$$
$$b_2 = b_2 \boxplus K_{2*i+1} \tag{11}$$
$$b_3 = b_3 \boxplus b_2 \tag{12}$$

This ends one round of encryption. The instructions are iterated until the number of rounds is reached.

## 2.3 Decryption

A block $B$ of size 512 bit is divided into four equal subblocks of 128 bit, i.e. $B = b_0 \| b_1 \| b_2 \| b_3$. See als Figure 1 and the function decrypt of the reference implementation of appendix A. Then the following operations are done ($i$ is the round value):

$$b_3 = b_3 \boxminus b_2 \tag{13}$$
$$b_2 = b_2 \boxminus K_{2*i+1} \tag{14}$$
$$b_2 = b_2 \oplus b_1 \tag{15}$$
$$b_1 = b_1 \boxminus b_0 \tag{16}$$
$$b_0 = b_0 \boxminus K_{2*i} \tag{17}$$

Now the odd subblocks are rotated to the right:

$$b_3 = b_3 \ggg 9 \tag{18}$$
$$b_1 = b_1 \ggg 7 \tag{19}$$

The subblocks $b_0$ to $b_2$ are XORed with the subblock to the right:

$$b_0 = b_0 \oplus b_1 \tag{20}$$
$$b_1 = b_1 \oplus b_2 \tag{21}$$
$$b_2 = b_2 \oplus b_3 \tag{22}$$

Now the even subblocks are rotated to the left:

$$b_2 = b_2 \lll 21 \tag{23}$$
$$b_0 = b_0 \lll 19 \tag{24}$$

This ends one round of decryption. The instructions are iterated until the number of rounds is reached.

## 2.4 Key Schedule

The key schedule of PAGES+ is complicated in comparison to other block ciphers. The goals are

- that all the bits of a round key are depending on all the bits of the userkey (completeness).
- that the round keys are statistically independant.

To achieve these aims, the block ciphers is employed in the derivations of the round keys, similar to the block ciphers 1024XKS, 2048XKS-F, 4096XKS-F and PAGES (see [7, 8, 9]). In the reference implementation the constants KEYLENGTH and NUMROUNDS are defined. They denote the length of the userkey in 64 bit blocks and the number of rounds, respectively. There is KEYLENGTH=NUMROUNDS/8 and 2*NUMROUNDS is the size of the round keys in blocks of 64 bits. In the arguments of the function expand_key are two arrays: userkey[KEYLENGTH] and keys[2*NUMROUNDS]. The array userkey hands over to the function expand_key the userkey. The array keys is initially undefined. The function expand_key copies the values of userkey to the least significant blocks of keys, i.e. keys[0]=userkey[0], keys[1]=userkey[1],…, keys[KEYLENGTH-1]=userkey[KEYLENGTH-1]. Now the least significant blocks of keys are in the whole rotated to the left by 121

(ROTROUNDKEY) bits and stored in keys[KEYLENGTH], keys[KEYLENGTH+1], ..., keys[2*KEYLENGTH-1]. The whole process of rotation is repeated 15 times until all the values of the array keys are defined from keys[0], keys[1], ..., to keys [2*NUMROUNDS-1].

Let $B$ denote a block of 512 bit divided into four equal subblocks of 128 bit, i.e. $B = b_0 \| b_1 \| b_2 \| b_3$. $B$ is encrypted in output feedback mode exactly NUMROUNDS/2 times. After each encryption, the start value is $i = 0$, the $b_k, 0 \leq k \leq 3$ are stored in the array keys. If the constant FORWARD is defined (conditional compilation):

$$keys[4 * i] = b_3 \tag{25}$$
$$keys[4 * i + 1] = b_2 \tag{26}$$
$$keys[4 * i + 2] = b_1 \tag{27}$$
$$keys[4 * i + 3] = b_0 \tag{28}$$

If the constant FORWARD is not defined, then the array keys is adressed the following way:

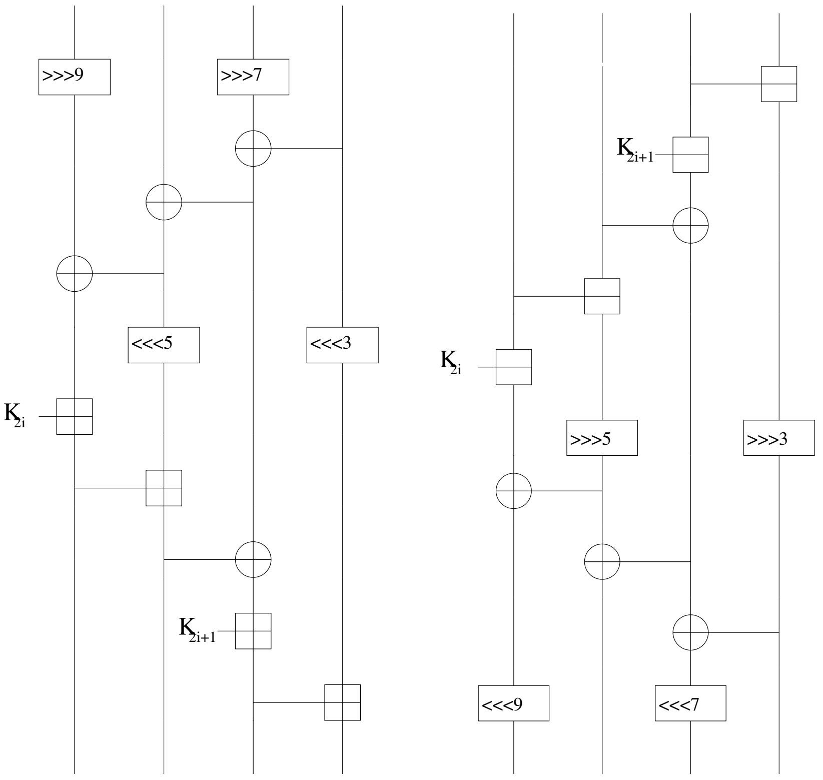
$$keys[2 * NUMROUNDS - 4 * i - 4] = b_3 \tag{29}$$
$$keys[2 * NUMROUNDS - 4 * i - 3] = b_2 \tag{30}$$
$$keys[2 * NUMROUNDS - 4 * i - 2] = b_1 \tag{31}$$
$$keys[2 * NUMROUNDS - 4 * i - 1] = b_0 \tag{32}$$

If all the values of keys are redefined the block cipher is ready for encryption or decryption.

## 2.5 Software Performance

On an AMD A6 6310 (Quadcore) with 1.8 GHz the following results were achieved. Operating system was Linux Mint 17 (64 bit) and the compiler was gcc version 4.8.2. Abbreviations: TP throughput, EC encryption cost in cycles/byte.

| COMPILER OPTIONS | | -O | | -O,-funroll-loops | |
|---|---|---|---|---|---|
| Key bit | Rounds | TP Mbit/s | EC cyc/by | TP Mbit/s | EC cyc/by |
| 1024 | 64 | 315 | 45.7 | 341 | 42.2 |
| 1536 | 96 | 216 | 67.0 | 234 | 61.5 |
| 2048 | 128 | 164 | 87.8 | 174 | 82.8 |



(a) Encryption (b) Decryption

Figure 2: A round of PAGES-

## 3 Assembly of PAGES-

### 3.1 Preliminaries

$\mathbb{F}_2^{64}$ denotes the 64 dimensional vector space over GF(2). The following symbols will be used to construct the block cipher PAGES-: $\boxplus$ denotes addition modulo $2^{64}$, $\boxminus$ denotes subtraction modulo $2^{64}$, $\oplus$ denotes addition modulo 2 in $\mathbb{F}_2^{64}$ (XOR), $\lll m$ denotes the rotation to the left by m bits of a 64 bit variable, $\ggg m$ denotes the rotation to the right by m bits of a 64 bit variable. PAGES- has a block size of 256 bit and is divided into four equal quarters of 64 bit. The key size und the rounds are as follows: 256 bit and 32 rounds, 384 bit and 48 rounds, 512 bit and 64 rounds, 784 bit and 96 rounds, and 1024 bit and 128 rounds. $B$ denotes a block of size 256 bit, and $b_i, 0 \leq k \leq 3$ denotes a block of 64 bit.

## 3.2 Encryption

A block $B$ of size 256 bit is divided into four equal subblocks of 64 bit, i.e. $B = b_0 \| b_1 \| b_2 \| b_3$, see also Figure 2 and the function encrypt in the reference implementation in appendix B. The even subblocks are rotated to the right:

$$b_0 = b_0 \ggg 9 \tag{33}$$
$$b_2 = b_2 \ggg 7 \tag{34}$$

The subblocks $b_2$ to $b_0$ are XORed with the subblocks to the right:

$$b_2 = b_2 \oplus b_3 \tag{35}$$
$$b_1 = b_1 \oplus b_2 \tag{36}$$
$$b_0 = b_0 \oplus b_1 \tag{37}$$

Now the odd subblocks are rotated to the left:

$$b_1 = b_1 \lll 5 \tag{38}$$
$$b_3 = b_3 \lll 3 \tag{39}$$

Now the following operations are carried out ($i$ is value of rounds):

$$b_0 = b_0 \boxplus K_{2*i} \tag{40}$$
$$b_1 = b_1 \boxplus b_0 \tag{41}$$
$$b_2 = b_2 \oplus b_1 \tag{42}$$
$$b_2 = b_2 \boxplus K_{2*i+1} \tag{43}$$
$$b_3 = b_3 \boxplus b_2 \tag{44}$$

This ends one round of encryption. The instructions are iterated until the number of rounds is reached.

## 3.3 Decryption

A block $B$ of size 256 bit is divided into four subblocks of 64 bit, i.e. $B = b_0 \| b_1 \| b_2 \| b_3$. See als Figure 2 and the function decrypt in the reference implementation in appendix B. Then the following operations are done ($i$ is the round value):

$$b_3 = b_3 \boxminus b_2 \tag{45}$$
$$b_2 = b_2 \boxminus K_{2*i+1} \tag{46}$$
$$b_2 = b_2 \oplus b_1 \tag{47}$$
$$b_1 = b_1 \boxminus b_0 \tag{48}$$
$$b_0 = b_0 \boxminus K_{2*i} \tag{49}$$

Now the odd subblocks are rotated to the right:

$$b_3 = b_3 \ggg 3 \tag{50}$$
$$b_1 = b_1 \ggg 5 \tag{51}$$

The subblocks $b_0$ to $b_2$ are XORed with the subblock to the right:

$$b_0 = b_0 \oplus b_1 \tag{52}$$
$$b_1 = b_1 \oplus b_2 \tag{53}$$
$$b_2 = b_2 \oplus b_3 \tag{54}$$

Now the even subblocks are rotated to the left:

$$b_2 = b_2 \lll 7 \tag{55}$$
$$b_0 = b_0 \lll 9 \tag{56}$$

This ends one round of decryption. The instructions are iterated until the number of rounds is reached.

## 3.4 Key Schedule

The key schedule of PAGES- is complicated in comparison to other block ciphers. The goals are

- that all the bits of a round key are depending on all the bits of the userkey (completeness).
- that the round keys are statistically independant.

To achieve these aims, the block ciphers is employed in the derivations of the round keys, similar to the block ciphers 1024XKS, 2048XKS-F, 4096XKS-F and PAGES (see [7, 8, 9]). In the reference implementation the constants KEYLENGTH and NUMROUNDS are defined. They denote the length of the userkey in 64 bit blocks and the number of rounds, respectively.

There is KEYLENGTH=NUMROUNDS/8 and 2*NUMROUNDS is the size of the round keys in blocks of 64 bits. In the arguments of the function expand_key are two arrays: userkey[KEYLENGTH] and keys[2*NUMROUNDS]. The array userkey hands over to the function expand_key the userkey. The array keys is initially undefined. The function expand_key copies the values of userkey to the least significant blocks of keys, i.e. keys[0]=userkey[0], keys[1]=userkey[1],. . ., keys[KEYLENGTH-1]=userkey[KEYLENGTH-1]. Now the least significant blocks of keys are in the whole rotated to the left by 61 (ROTROUNDKEY) bits and stored in keys[KEYLENGTH], keys[KEYLENGTH+1], . . ., keys[2*KEYLENGTH-1]. The whole process of rotation is repeated 15 times until all the values of the array keys are defined from keys[0], keys[1], . . ., to keys [2*NUMROUNDS-1].

Let $B$ denote a block of 256 bit divided into four equal subblocks of 64 bit, i.e. $B = b_0\|b_1\|b_2\|b_3$. $B$ is encrypted in output feedback mode exactly NUMROUNDS/2 times. After each encryption, the start value is $i = 0$, the $b_k, 0 \leq k \leq 3$ are stored in the array keys. If the constant FORWARD is defined (conditional compilation):

$$keys[4 * i] = b_3 \tag{57}$$
$$keys[4 * i + 1] = b_2 \tag{58}$$
$$keys[4 * i + 2] = b_1 \tag{59}$$
$$keys[4 * i + 3] = b_0 \tag{60}$$

If the constant FORWARD is not defined, then the array keys is adressed the following way:

$$keys[2 * NUMROUNDS - 4 * i - 4] = b_3 \tag{61}$$
$$keys[2 * NUMROUNDS - 4 * i - 3] = b_2 \tag{62}$$
$$keys[2 * NUMROUNDS - 4 * i - 2] = b_1 \tag{63}$$
$$keys[2 * NUMROUNDS - 4 * i - 1] = b_0 \tag{64}$$

If all the values of keys are redefined the block cipher is ready for encryption or decryption.

## 3.5 Software Performance

On an AMD A6 6310 (Quadcore) with 1.8 GHz the following results were achieved. Operating system was Linux Mint 17 (64 bit) and the compiler was gcc version 4.8.2. Abbreviations: TP throughput, EC encryption cost in cycles/byte.

| Compiler Options | | -O | | -O,-funroll-loops | |
|---|---|---|---|---|---|
| Key bit | Rounds | TP Mbit/s | EC cyc/by | TP Mbit/s | EC cyc/by |
| 256 | 32 | 1724 | 8.4 | 1928 | 7.5 |
| 384 | 48 | 1170 | 12.3 | 1285 | 11.2 |
| 512 | 64 | 886 | 16.3 | 978 | 14.7 |
| 768 | 96 | 590 | 24.4 | 655 | 22.0 |
| 1024 | 128 | 449 | 32.1 | 489 | 29.4 |

The key sizes for 256 bit and 384 bit are there for the sake of completeness. However, users are strictly warned to use these key lengths. Bamford wrote in May 2005 in [1]: **Another factor was the growing use of encryption and the NSA's inability, without spending excessive amounts of computer time and human energy, to solve commercial systems more complex than 256 bits**.

## 4 Assembly of PAGES–

### 4.1 Preliminaries

$\mathbb{F}_2^{64}$ denotes the 64 dimensional vector space over GF(2). The following symbols will be used to construct the block cipher PAGES–: $\boxplus$ denotes addition modulo $2^{64}$, $\boxminus$ denotes subtraction modulo $2^{64}$, $\oplus$ denotes addition modulo 2 in $\mathbb{F}_2^{64}$ (XOR), $\lll m$ denotes the rotation to the left by m bits of a 64 bit variable, $\ggg m$ denotes the rotation to the right by m bits of a 64 bit variable. PAGES– has a block size of 512 bit and is divided into eight equal eighth of 64 bit. The key size und the rounds are as follows: 512 bit and 32 rounds, 786 bit and 48 rounds, 1024 bit and 64 rounds, 1536 bit and 96 rounds, and 2048 bit and 128 rounds. $B$ denotes a block of size 512 bit, and $b_i, 0 \leq k \leq 7$ denotes a block of 64 bit.

### 4.2 Encryption

A block $B$ of size 512 bit is divided into eight equal subblocks of 64 bit, i.e. $B = b_0||b_1||b_2||b_3||b_4||b_5||b_6||b_7$, see also Figure 3 and the function encrypt in the reference implementation in appendix C. The even subblocks are rotated to the right:

$$b_0 = b_0 \ggg 9 \tag{65}$$
$$b_2 = b_2 \ggg 7 \tag{66}$$
$$b_4 = b_4 \ggg 11 \tag{67}$$
$$b_6 = b_6 \ggg 13 \tag{68}$$

The subblocks $b_6$ to $b_0$ are XORed with the subblocks to the right:

$$b_6 = b_6 \oplus b_7 \tag{69}$$
$$b_5 = b_5 \oplus b_6 \tag{70}$$
$$b_4 = b_4 \oplus b_5 \tag{71}$$
$$b_3 = b_3 \oplus b_4 \tag{72}$$
$$b_2 = b_2 \oplus b_3 \tag{73}$$
$$b_1 = b_1 \oplus b_2 \tag{74}$$
$$b_0 = b_0 \oplus b_1 \tag{75}$$

Now the odd subblocks are rotated to the left:

$$b_1 = b_1 \lll 5 \tag{76}$$
$$b_3 = b_3 \lll 3 \tag{77}$$
$$b_5 = b_5 \lll 7 \tag{78}$$
$$b_7 = b_7 \lll 9 \tag{79}$$

Now the following operations are carried out ($i$ is value of rounds):

$$b_0 = b_0 \boxplus K_{4*i} \tag{80}$$
$$b_1 = b_1 \boxplus b_0 \tag{81}$$
$$b_2 = b_2 \oplus b_1 \tag{82}$$
$$b_2 = b_2 \boxplus K_{4*i+1} \tag{83}$$
$$b_3 = b_3 \boxplus b_2 \tag{84}$$
$$b_4 = b_4 \oplus b_3 \tag{85}$$
$$b_4 = b_4 \boxplus K_{4*i+2} \tag{86}$$
$$b_5 = b_5 \boxplus b_4 \tag{87}$$
$$b_6 = b_6 \oplus b_5 \tag{88}$$
$$b_6 = b_6 \boxplus K_{4*i+3} \tag{89}$$
$$b_7 = b_7 \boxplus b_6 \tag{90}$$

This ends one round of encryption. The instructions are iterated until the number of rounds is reached.

## 4.3 Decryption

A block $B$ of size 512 bit is divided into eight subblocks of 64 bit, i.e. $B = b_0 \| b_1 \| b_2 \| b_3 \| b_4 \| b_5 \| b_6 \| b_7$. See als Figure 4 and the function decrypt of the reference implementation in appendix C. Then the following operations are done ($i$ is the round value):

$$b_7 = b_7 \boxminus b_6 \tag{91}$$
$$b_6 = b_6 \boxminus K_{4*i+3} \tag{92}$$
$$b_6 = b_6 \oplus b_5 \tag{93}$$
$$b_5 = b_5 \boxminus b_4 \tag{94}$$
$$b_4 = b_4 \boxminus K_{4*i+2} \tag{95}$$
$$b_4 = b_4 \oplus b_3 \tag{96}$$
$$b_3 = b_3 \boxminus b_2 \tag{97}$$
$$b_2 = b_2 \boxminus K_{4*i+1} \tag{98}$$
$$b_2 = b_2 \oplus b_1 \tag{99}$$
$$b_1 = b_1 \boxminus b_0 \tag{100}$$
$$b_0 = b_0 \boxminus K_{4*i} \tag{101}$$

Now the odd subblocks are rotated to the right:

$$b_7 = b_7 \ggg 9 \tag{102}$$
$$b_5 = b_5 \ggg 7 \tag{103}$$
$$b_3 = b_3 \ggg 3 \tag{104}$$
$$b_1 = b_1 \ggg 5 \tag{105}$$

The subblocks $b_0$ to $b_6$ are XORed with the subblock to the right:

$$b_0 = b_0 \oplus b_1 \tag{106}$$
$$b_1 = b_1 \oplus b_2 \tag{107}$$
$$b_2 = b_2 \oplus b_3 \tag{108}$$
$$b_3 = b_3 \oplus b_4 \tag{109}$$
$$b_4 = b_4 \oplus b_5 \tag{110}$$
$$b_5 = b_5 \oplus b_6 \tag{111}$$
$$b_6 = b_6 \oplus b_7 \tag{112}$$

Now the even subblocks are rotated to the left:

$$b_6 = b_6 \lll 13 \tag{113}$$
$$b_4 = b_4 \lll 11 \tag{114}$$
$$b_2 = b_2 \lll 7 \tag{115}$$
$$b_0 = b_0 \lll 9 \tag{116}$$

This ends one round of decryption. The instructions are iterated until the number of rounds is reached.

## 4.4 Key Schedule

The key schedule of PAGES– is complicated in comparison to other block ciphers. The goals are

- that all the bits of a round key are depending on all the bits of the userkey (completeness).
- that the round keys are statistically independant.

To achieve these aims, the block ciphers is employed in the derivations of the round keys, similar to the block ciphers 1024XKS, 2048XKS-F, 4096XKS-F and PAGES (see [7, 8, 9]). In the reference implementation the constants KEYLENGTH and NUMROUNDS are defined. They denote the length of the userkey in 64 bit blocks and the number of rounds, respectively. There is KEYLENGTH=NUMROUNDS/4 and 4*NUMROUNDS is the size of the round keys in blocks of 64 bits. In the arguments of the function expand_key are two arrays: userkey[KEYLENGTH] and keys[4*NUMROUNDS]. The array userkey hands over to the function expand_key the userkey. The array keys is initially undefined. The function expand_key copies the values of userkey to the least significant blocks of keys, i.e. keys[0]=userkey[0], keys[1]=userkey[1],. . ., keys[KEYLENGTH-1]=userkey[KEYLENGTH-1]. Now the least significant blocks of keys are in the whole rotated to the left by 61 (ROTROUNDKEY) bits and stored in keys[KEYLENGTH], keys[KEYLENGTH+1], . . ., keys[2*KEYLENGTH-1]. The whole process of rotation is repeated 15 times until all the values of the array keys are defined from keys[0], keys[1], . . ., to keys [4*NUMROUNDS-1].

Let $B$ denote a block of 512 bit divided into four equal subblocks of 64 bit, i.e. $B = b_0 \| b_1 \| b_2 \| b_3 \| b_4 \| b_5 \| b_6 \| b_7$. $B$ is encrypted in output feedback mode exactly NUMROUNDS/2 times. After each encryption, the start value is $i = 0$, the $b_k, 0 \leq k \leq 7$ are stored in the array keys. If the constant FORWARD is defined (conditional compilation):

$$keys[8*i] = b_7 \tag{117}$$
$$keys[8*i+1] = b_6 \tag{118}$$
$$keys[8*i+2] = b_5 \tag{119}$$
$$keys[8*i+3] = b_4 \tag{120}$$
$$keys[8*i+4] = b_3 \tag{121}$$
$$keys[8*i+5] = b_2 \tag{122}$$
$$keys[8*i+6] = b_1 \tag{123}$$
$$keys[8*i+7] = b_0 \tag{124}$$

If the constant FORWARD is not defined, then the array keys is adressed the following way:

$$keys[4*NUMROUNDS-8*i-8]=b_7 \tag{125}$$
$$keys[4*NUMROUNDS-8*i-7]=b_6 \tag{126}$$
$$keys[4*NUMROUNDS-8*i-6]=b_5 \tag{127}$$
$$keys[4*NUMROUNDS-8*i-5]=b_4 \tag{128}$$
$$keys[4*NUMROUNDS-8*i-4]=b_3 \tag{129}$$
$$keys[4*NUMROUNDS-8*i-3]=b_2 \tag{130}$$
$$keys[4*NUMROUNDS-8*i-2]=b_1 \tag{131}$$
$$keys[4*NUMROUNDS-8*i-1]=b_0 \tag{132}$$

If all the values of keys are redefined the block cipher is ready for encryption or decryption.

### 4.5 Software Performance

On an AMD A6 6310 (Quadcore) with 1.8 GHz the following results were achieved. Operating system was Linux Mint 17 (64 bit) and the compiler was gcc version 4.8.2. Abbreviations: TP throughput, EC encryption cost in cycles/byte.

| Compiler Options | | -O | | -O,-funroll-loops | |
|---|---|---|---|---|---|
| Key bit | Rounds | TP Mbit/s | EC cyc/by | TP Mbit/s | EC cyc/by |
| 512 | 32 | 1560 | 9.2 | 1598 | 9.0 |
| 768 | 48 | 1024 | 14.1 | 1092 | 13.2 |
| 1024 | 64 | 780 | 18.5 | 799 | 18.0 |
| 1536 | 96 | 524 | 27.5 | 529 | 27.2 |
| 2048 | 128 | 395 | 36.5 | 405 | 35.6 |

## 5 Conclusion

We have presented the block ciphers PAGES+, PAGES-, and PAGES–. The block ciphers are free from any intellectual property restrictions. The reference implementations are subject to the GNU General Public License.

## References

[1] Bamford, James: *A Pretext for War*, Anchor Books, New York, 2005, p. 389 10

[2] Beaulieu, Ray; Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, Louis Wingers: *The Simon and Speck Families of Lightweight Block Ciphers*, IACR ePrint Archives, Report 2013/404 1

[3] Feistel, Horst: *Cryptography and Computer Privacy*, Scientific American, v. 228, n. 5, May 1973, pp. 15-23 1

[4] Matsui, Mitsuru: *Linear Cryptanalysis Method for DES Cipher*, in Tor Helleseth(ed.): Advances in Cryptology - EUROCRYPT 93, LNCS, Springer Verlag, Berlin, 1993 1

[5] Mukhopadhyay, Debdeep and Dipanwita RoyChowdhury: *Key Mixing in Block Cipher through Addition modulo* $2^n$, IACR ePrint Archives, Report 2005/383 1

[6] Schmidt, Dieter: *1024 - A High Security Software Oriented Block Cipher*, IACR ePrint Archives, Report 2009/104

[7] Schmidt, Dieter: *1024XKS - A High Security Software Oriented Block Cipher Revisited*, IACR ePrint Archives, Report 2010/162 4, 8, 13

[8] Schmidt, Dieter: *2048XKS-F & 4096XKS-F - Two Software Oriented High Security Block Ciphers*, IACR ePrint Archives, Report 2013/136 4, 8, 13

[9] Schmidt, Dieter: *PAGES - A Family of Block Ciphers*, IACR ePrint Archives, Report 2015/329 1, 4, 8, 13

[10] Schneier, Bruce: *Angewandte Kryptographie*, Addison-Wesley, Bonn, 1999, German Translation of Applied Cryptography 1

[11] Shannon, Claude Elwood: *Communication Theory of Secrecy Systems*, Bell Systems Technical Journal, v. 28, n. 4, 1949, pp. 656-715, Reprint in Sloane, N.J.A., A. Wyner (eds.): *Claude Elwood Shannon: Collected Papers*, IEEE Press, Piscataway, USA 1993 1

[12] Wallén, Johan: *Linear Approximations of Addition modulo* $2^n$, in Thomas Johansson(ed.): Fast Software Encryption (FSE) 2003, LNCS, Springer Verlag, Berlin, 2003 1

## A Reference Implementation of PAGES+

```
/*
The C reference implementation of the
block ciphers PAGES+ with 512 bit blocksize
for gcc compatible compilers.

Copyright 2015 by

Dieter Schmidt*/

#include<stdio.h>
#include<time.h>

#define INTLENGTH 128
#define NUMROUNDS 128 // 64,96, and 128 are possible
#define ROL(x,a) ((((x)<<(a))|((x)>>(INTLENGTH-(a)))))
#define ROR(x,a) ((((x)>>(a))|((x)<<(INTLENGTH-(a)))))
#define KEYLENGTH NUMROUNDS/8
#define ROTROUNDKEY 121
#define ROTROUNDDATA1 7
#define ROTROUNDDATA0 19
#define ROTROUNDDATA3 9
#define ROTROUNDDATA2 21

#define FORWARD

void encrypt(unsigned __int128 data[4],\
unsigned __int128 keys[2*NUMROUNDS]){

  register unsigned long i;
  register unsigned __int128 a,b,c,d;

  a=data[0];b=data[1];c=data[2];d=data[3];
  for(i=0;i<NUMROUNDS;i++){
    a=ROR(a,ROTROUNDDATA0);
    c=ROR(c,ROTROUNDDATA2);
    c^=d;
    b^=c;
    a^=b;
    b=ROL(b,ROTROUNDDATA1);
    d=ROL(d,ROTROUNDDATA3);
    a+=keys[2*i];
    b+=a;
    c^=b;
    c+=keys[2*i+1];
    d+=c;
  }
  data[0]=a;data[1]=b;data[2]=c;data[3]=d;
  return;
}

void decrypt(unsigned __int128 data[4],\
unsigned __int128 keys[2*NUMROUNDS]){

  register unsigned long i;
  register unsigned __int128 a,b,c,d;

  a=data[0];b=data[1];c=data[2];d=data[3];
  for(i=0;i<NUMROUNDS;i++){
    d-=c;
    c-=keys[2*NUMROUNDS-2*i-1];
    c^=b;
    b-=a;
    a-=keys[2*NUMROUNDS-2*i-2];
    d=ROR(d,ROTROUNDDATA3);
    b=ROR(b,ROTROUNDDATA1);
    a^=b;
    b^=c;
    c^=d;
    c=ROL(c,ROTROUNDDATA2);
    a=ROL(a,ROTROUNDDATA0);
  }
  data[0]=a;data[1]=b;data[2]=c;data[3]=d;
  return;
}

void expand_key(unsigned __int128 userkey[KEYLENGTH],\
unsigned __int128 keys[2*NUMROUNDS]){

  unsigned long i,j;
  unsigned __int128 data[4],a;

  for(i=0;i<KEYLENGTH;i++) keys[i]=userkey[i];
    for(i=1;i<16;i++){
      a=keys[(i-1)*KEYLENGTH];
      a>>=(INTLENGTH-ROTROUNDKEY);
      for(j=0;j<(KEYLENGTH-1);j++){
        keys[i*KEYLENGTH+j]=(keys[(i-1)*KEYLENGTH+j]\
        <<ROTROUNDKEY)|(keys[(i-1)*KEYLENGTH+j+1]\
        >>(INTLENGTH-ROTROUNDKEY));
    }
    keys[i*KEYLENGTH+KEYLENGTH-1]=\
    (keys[(i-1)*KEYLENGTH+KEYLENGTH-1]\
    <<ROTROUNDKEY)|a;
  }
  data[0]=0;data[1]=0;data[2]=0;data[3]=0;
  for(i=0;i<(NUMROUNDS/2);i++){
    encrypt(data,keys);
    #ifdef FORWARD
      keys[4*i]=data[3];
      keys[4*i+1]=data[2];
      keys[4*i+2]=data[1];
      keys[4*i+3]=data[0];
    #else
      keys[2*NUMROUNDS-4-4*i]=data[3];
      keys[2*NUMROUNDS-4*i-3]=data[2];
      keys[2*NUMROUNDS-4*i-2]=data[1];
      keys[2*NUMROUNDS-4*i-1]=data[0];
    #endif
  }
  return;
}

int main(){

  unsigned __int128 data[4],userkey[KEYLENGTH],keys[2*NUMROUNDS];
  unsigned long i,j,k,l;
  time_t start,stop;
  double r;

  data[0]=0;data[1]=1;data[2]=2;data[3]=3;
  i=(long) data[0];
  j=(long) data[1];
  k=(long) data[2];
  l=(long) data[3];
  printf("Before encryption %20lx%20lx%20lx%20lx\n",i,j,k,l);
  for(i=0;i<KEYLENGTH;i++) userkey[i]=i;
  expand_key(userkey,keys);
  encrypt(data,keys);
  i=(long) data[0];
  j=(long) data[1];
  k=(long) data[2];
  l=(long) data[3];
  printf("After encryption  %20lx%20lx%20lx%20lx\n",i,j,k,l);
  decrypt(data,keys);
  i=(long) data[0];
  j=(long) data[1];
  k=(long) data[2];
  l=(long) data[3];
  printf("After decryption  %20lx%20lx%20lx%20lx\n",i,j,k,l);
  time(&start);
  for(i=0;i<1024*1024*16;i++) encrypt(data,keys);
  time(&stop);
  r=((double) 8192)/difftime(stop,start);
  printf("%20.10lf\n",r);

  return(0);
}
```

## B Reference Implementation of PAGES-

```
/*
The C reference implementation of the
ciphers PAGES- with 256 bit blocksize
for gcc compatible compilers.

Copyright 2015 by

Dieter Schmidt*/

#include<stdio.h>
#include<time.h>
#include<x86intrin.h>

#define INTLENGTH 64
#define NUMROUNDS 128// 32,48,64,96,128 are possible
#define ROL _lrotl
#define ROR _lrotr
#define KEYLENGTH NUMROUNDS/8
#define ROTROUNDKEY 61
#define ROTROUNDDATA1 5
#define ROTROUNDDATA0 9
#define ROTROUNDDATA3 3
#define ROTROUNDDATA2 7
```

```
#define FORWARD

void encrypt(unsigned long data[4],\
unsigned long keys[2*NUMROUNDS]){

  register unsigned long i;
  register unsigned long a,b,c,d;

  a=data[0];b=data[1];c=data[2];d=data[3];
  for(i=0;i<NUMROUNDS;i++){
    a=ROR(a,ROTROUNDDATA0);
    c=ROR(c,ROTROUNDDATA2);
    c^=d;
    b^=c;
    a^=b;
    b=ROL(b,ROTROUNDDATA1);
    d=ROL(d,ROTROUNDDATA3);
    a+=keys[2*i];
    b+=a;
    c^=b;
    c+=keys[2*i+1];
    d+=c;
  }
  data[0]=a;data[1]=b;data[2]=c;data[3]=d;
  return;
}

void decrypt(unsigned long data[4],\
unsigned long keys[2*NUMROUNDS]){

  register unsigned long i;
  register unsigned long a,b,c,d;

  a=data[0];b=data[1];c=data[2];d=data[3];
  for(i=0;i<NUMROUNDS;i++){
    d-=c;
    c-=keys[2*NUMROUNDS-2*i-1];
    c^=b;
    b-=a;
    a-=keys[2*NUMROUNDS-2*i-2];
    d=ROR(d,ROTROUNDDATA3);
    b=ROR(b,ROTROUNDDATA1);
    a^=b;
    b^=c;
    c^=d;
    c=ROL(c,ROTROUNDDATA2);
    a=ROL(a,ROTROUNDDATA0);
  }
  data[0]=a;data[1]=b;data[2]=c;data[3]=d;
  return;
}

void expand_key(unsigned long userkey[KEYLENGTH],\
unsigned long keys[2*NUMROUNDS]){

  unsigned long i,j;
  unsigned long data[4],a;

  for(i=0;i<KEYLENGTH;i++) keys[i]=userkey[i];
    for(i=1;i<16;i++){
      a=keys[(i-1)*KEYLENGTH];
      a>>=(INTLENGTH-ROTROUNDKEY);
      for(j=0;j<(KEYLENGTH-1);j++){
        keys[i*KEYLENGTH+j]=(keys[(i-1)*KEYLENGTH+j]\
        <<ROTROUNDKEY)|(keys[(i-1)*KEYLENGTH+j+1]\
        >>(INTLENGTH-ROTROUNDKEY));
    }
    keys[i*KEYLENGTH+KEYLENGTH-1]=\
    (keys[(i-1)*KEYLENGTH+KEYLENGTH-1]\
    <<ROTROUNDKEY)|a;
  }
  data[0]=0;data[1]=0;data[2]=0;data[3]=0;
  for(i=0;i<(NUMROUNDS/2);i++){
    encrypt(data,keys);
    #ifdef FORWARD
      keys[4*i]=data[3];
      keys[4*i+1]=data[2];
      keys[4*i+2]=data[1];
      keys[4*i+3]=data[0];
    #else
      keys[2*NUMROUNDS-4-4*i]=data[3];
      keys[2*NUMROUNDS-4*i-3]=data[2];
      keys[2*NUMROUNDS-4*i-2]=data[1];
      keys[2*NUMROUNDS-4*i-1]=data[0];
    #endif
  }
  return;
```

```
}

int main(){

  unsigned long data[4],userkey[KEYLENGTH],keys[2*NUMROUNDS];
  unsigned long i,j,k,l;
  time_t start,stop;
  double r;

  data[0]=0;data[1]=1;data[2]=2;data[3]=3;
  i=(long) data[0];
  j=(long) data[1];
  k=(long) data[2];
  l=(long) data[3];
  printf("Before encryption %20lx%20lx%20lx%20lx\n",i,j,k,l);
  for(i=0;i<KEYLENGTH;i++) userkey[i]=i;
  expand_key(userkey,keys);
  encrypt(data,keys);
  i=(long) data[0];
  j=(long) data[1];
  k=(long) data[2];
  l=(long) data[3];
  printf("After encryption  %20lx%20lx%20lx%20lx\n",i,j,k,l);
  decrypt(data,keys);
  i=(long) data[0];
  j=(long) data[1];
  k=(long) data[2];
  l=(long) data[3];
  printf("After decryption  %20lx%20lx%20lx%20lx\n",i,j,k,l);
  time(&start);
  for(i=0;i<1024*1024*256;i++) encrypt(data,keys);
  time(&stop);
  r=((double) 65536)/difftime(stop,start);
  printf("%20.10lf\n",r);

  return(0);
}
```

## C Reference Implementation of PAGES–

```
/*
The C reference implementation of the
ciphers PAGES-- with 512 bit blocksize
```

```
for gcc compatible compilers.

Copyright 2015 by

Dieter Schmidt*/

#include<stdio.h>
#include<time.h>
#include<x86intrin.h>

#define INTLENGTH 64
#define NUMROUNDS 128// 32,48,64,96,128 are possible
#define ROL _lrotl
#define ROR _lrotr
#define KEYLENGTH NUMROUNDS/4
#define ROTROUNDKEY 61
#define ROTROUNDDATA1 5
#define ROTROUNDDATA0 9
#define ROTROUNDDATA3 3
#define ROTROUNDDATA2 7
#define ROTROUNDDATA5 7
#define ROTROUNDDATA4 11
#define ROTROUNDDATA7 9
#define ROTROUNDDATA6 13


#define FORWARD

void encrypt(unsigned long data[8],\
unsigned long keys[4*NUMROUNDS]){

  register unsigned long i;
  register unsigned long a,b,c,d,e,f,g,h;

  a=data[0];b=data[1];c=data[2];d=data[3];
  e=data[4];f=data[5];g=data[6];h=data[7];
  for(i=0;i<NUMROUNDS;i++){
    a=ROR(a,ROTROUNDDATA0);
    c=ROR(c,ROTROUNDDATA2);
    e=ROR(e,ROTROUNDDATA4);
    g=ROR(g,ROTROUNDDATA6);
    g^=h;
    f^=g;
    e^=f;
    d^=e;
    c^=d;
    b^=c;
    a^=b;
    b=ROL(b,ROTROUNDDATA1);
    d=ROL(d,ROTROUNDDATA3);
    f=ROL(f,ROTROUNDDATA5);
    h=ROL(h,ROTROUNDDATA7);
    a+=keys[4*i];
    b+=a;
    c^=b;
    c+=keys[4*i+1];
    d+=c;
    e^=d;
    e+=keys[4*i+2];
    f+=e;
    g^=f;
    g+=keys[4*i+3];
    h+=g;
  }
  data[0]=a;data[1]=b;data[2]=c;data[3]=d;
  data[4]=e;data[5]=f;data[6]=g;data[7]=h;
  return;
 }

void decrypt(unsigned long data[8],\
unsigned long keys[4*NUMROUNDS]){

  register unsigned long i;
  register unsigned long a,b,c,d,e,f,g,h;

  a=data[0];b=data[1];c=data[2];d=data[3];
  e=data[4];f=data[5];g=data[6];h=data[7];
  for(i=0;i<NUMROUNDS;i++){
    h-=g;
    g-=keys[4*NUMROUNDS-4*i-1];
    g^=f;
    f-=e;
    e-=keys[4*NUMROUNDS-4*i-2];
    e^=d;
    d-=c;
    c-=keys[4*NUMROUNDS-4*i-3];
    c^=b;
    b-=a;
    a-=keys[4*NUMROUNDS-4*i-4];
    h=ROR(h,ROTROUNDDATA7);
    f=ROR(f,ROTROUNDDATA5);
    d=ROR(d,ROTROUNDDATA3);
    b=ROR(b,ROTROUNDDATA1);
    a^=b;
    b^=c;
    c^=d;
    d^=e;
    e^=f;
    f^=g;
    g^=h;
    g=ROL(g,ROTROUNDDATA6);
    e=ROL(e,ROTROUNDDATA4);
    c=ROL(c,ROTROUNDDATA2);
    a=ROL(a,ROTROUNDDATA0);
  }
  data[0]=a;data[1]=b;data[2]=c;data[3]=d;
  data[4]=e;data[5]=f;data[6]=g;data[7]=h;
  return;
}

void expand_key(unsigned long userkey[KEYLENGTH],\
unsigned long keys[4*NUMROUNDS]){

  unsigned long i,j;
  unsigned long data[8],a;

  for(i=0;i<KEYLENGTH;i++) keys[i]=userkey[i];
    for(i=1;i<16;i++){
      a=keys[(i-1)*KEYLENGTH];
      a>>=(INTLENGTH-ROTROUNDKEY);
      for(j=0;j<(KEYLENGTH-1);j++){
        keys[i*KEYLENGTH+j]=(keys[(i-1)*KEYLENGTH+j]\
        <<ROTROUNDKEY)|(keys[(i-1)*KEYLENGTH+j+1]\
        >>(INTLENGTH-ROTROUNDKEY));
    }
    keys[i*KEYLENGTH+KEYLENGTH-1]=\
    (keys[(i-1)*KEYLENGTH+KEYLENGTH-1]\
    <<ROTROUNDKEY)|a;
  }
  data[0]=0;data[1]=0;data[2]=0;data[3]=0;
  data[4]=0;data[5]=0;data[6]=0;data[7]=0;
  for(i=0;i<(NUMROUNDS/2);i++){
    encrypt(data,keys);
    #ifdef FORWARD
      keys[8*i]=data[7];
      keys[8*i+1]=data[6];
      keys[8*i+2]=data[5];
      keys[8*i+3]=data[4];
      keys[8*i+4]=data[3];
      keys[8*i+5]=data[2];
      keys[8*i+6]=data[1];
      keys[8*i+7]=data[0];
    #else
      keys[4*NUMROUNDS-8-8*i]=data[7];
      keys[4*NUMROUNDS-8*i-7]=data[6];
      keys[4*NUMROUNDS-8*i-6]=data[5];
      keys[4*NUMROUNDS-8*i-5]=data[4];
      keys[4*NUMROUNDS-4-8*i]=data[3];
      keys[4*NUMROUNDS-8*i-3]=data[2];
      keys[4*NUMROUNDS-8*i-2]=data[1];
      keys[4*NUMROUNDS-8*i-1]=data[0];
    #endif
  }
  return;
}

int main(){

  unsigned long data[8],userkey[KEYLENGTH],keys[4*NUMROUNDS];
  unsigned long i,j,k,l,m,n,o,p;
  time_t start,stop;
  double r;

  data[0]=0;data[1]=1;data[2]=2;data[3]=3;
  data[4]=4;data[5]=5;data[6]=6;data[7]=7;
  i=(long) data[0];
  j=(long) data[1];
  k=(long) data[2];
  l=(long) data[3];
  m=(long) data[4];
  n=(long) data[5];
  o=(long) data[6];
  p=(long) data[7];
  printf("Before encryption %20lx%20lx%20lx%20lx%20lx%20lx%20lx%20lx\n"\
  ,i,j,k,l,m,n,o,p);
  for(i=0;i<KEYLENGTH;i++) userkey[i]=i;
  expand_key(userkey,keys);
  encrypt(data,keys);
  i=(long) data[0];
  j=(long) data[1];
  k=(long) data[2];
  l=(long) data[3];
  m=(long) data[4];
  n=(long) data[5];
  o=(long) data[6];
  p=(long) data[7];
  printf("After encryption  %20lx%20lx%20lx%20lx%20lx%20lx%20lx%20lx\n"\
  ,i,j,k,l,m,n,o,p);
  decrypt(data,keys);
  i=(long) data[0];
  j=(long) data[1];
  k=(long) data[2];
  l=(long) data[3];
  m=(long) data[4];
  n=(long) data[5];
  o=(long) data[6];
  p=(long) data[7];
  printf("After decryption  %20lx%20lx%20lx%20lx%20lx%20lx%20lx%20lx\n"\
  ,i,j,k,l,m,n,o,p);
  time(&start);
  for(i=0;i<1024*1024*128;i++) decrypt(data,keys);
  time(&stop);
  r=((double) 65536)/difftime(stop,start);
  printf("%20.10lf\n",r);

  return(0);
}
```

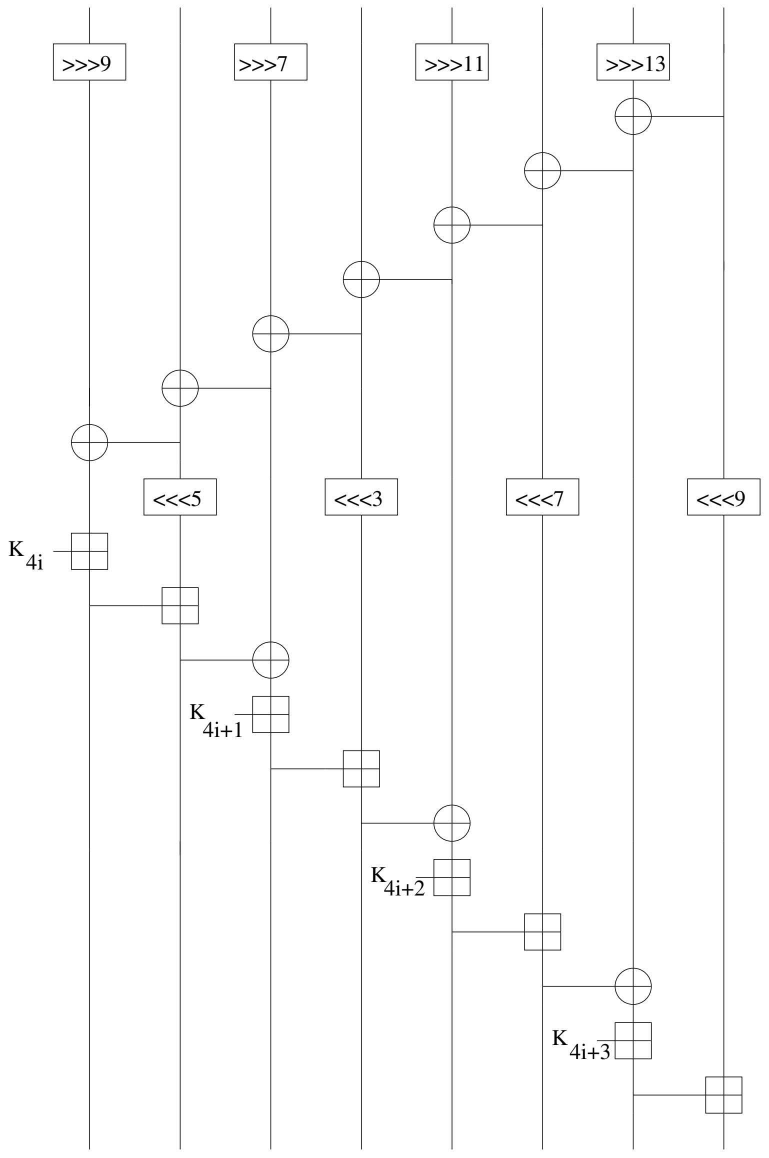


Figure 3: A encryption round of PAGES–

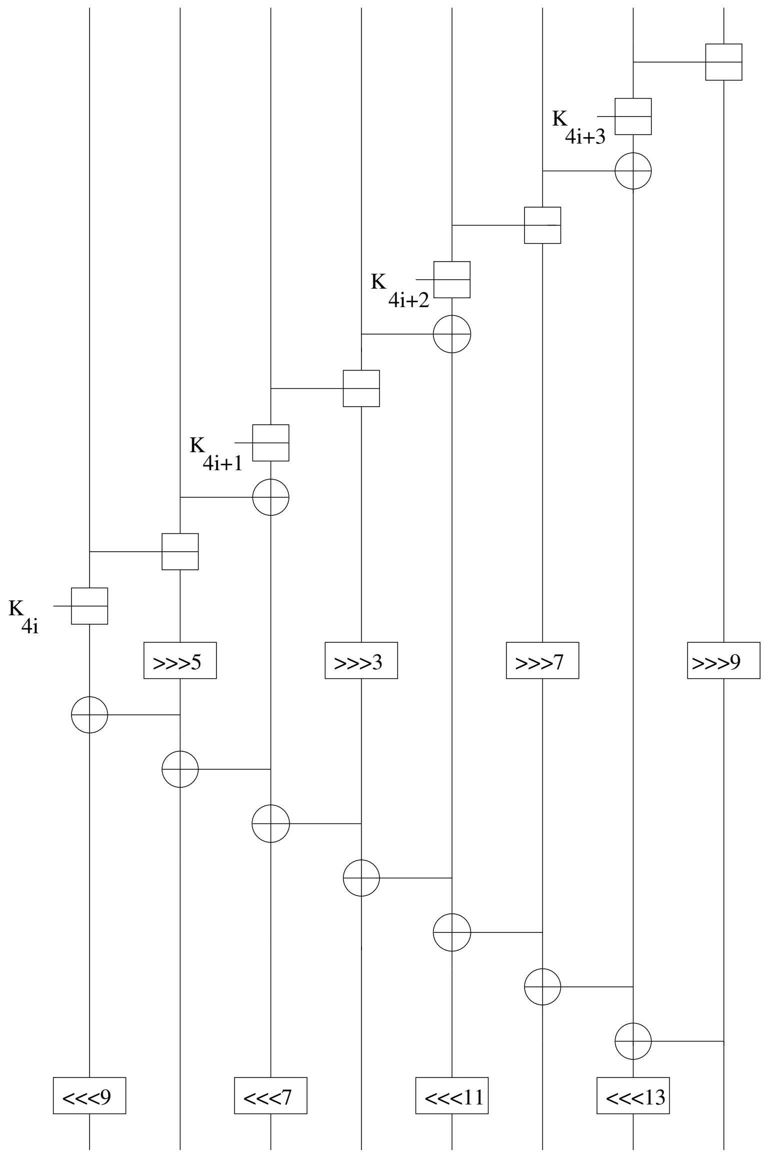


Figure 4: A decryption round of PAGES–