PAGES - A Family of Block Ciphers

Dieter Schmidt*

May 3, 2015

Abstract

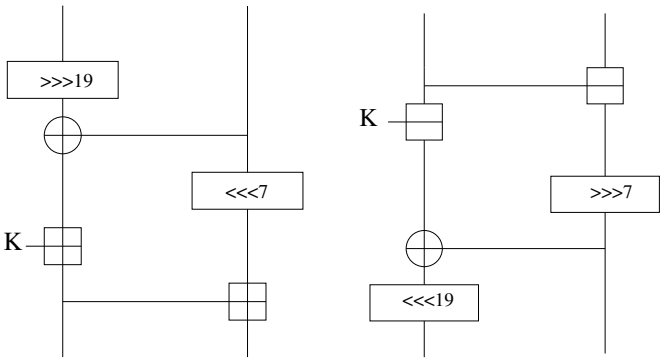
PAGES is block cipher family based on the design of Speck, see [1]. However, some intriguing design details of Speck were not used in the design of PAGES. PAGES has block size of 256 bit and comes in three version: PAGES-512, PAGES-768, and PAGES-1024, were the number denotes the key length. The number of rounds is 64, 96, or 128, respectively. PAGES uses variables of 128 bit, that is half the block size.

1 Introduction

The Simon and Speck Block Cipher Families, see [1], have an impressive performance and are easy to program on modern CPU's. In particular, the SSE performance, where eight encryptions are down in parallel (Speck128), outperforms other existing block ciphers. The SSE register in my laptop have a width of 128 bit, it sprang into my mind, that an implementation which uses the register at full size, could have an increased security. Since gcc version 4.8.2 supports the variables with 128 bit (unsigned __int128), it was easy to design a block cipher similar to Speck128, which utilizes 128 bit variables. Since the days of register poor CPU's are over, both AMD and Intel microprocessors have enough registers to store the variables of the cipher. The 128 bit variable are saved into two general purpose registers of 64 bit. The cipher is called PAGES and its reference implementation can be found in the appendix A. As NSA nor its employees claim intellectual property for its block ciphers, PAGES is also free of charge and I claim no intellectual property for this cipher. While it was straightforward to implement PAGES, some intriguing details in the Speck128 cipher were not used. In particular the eight bit rotation of the lefthand side were not increased by the factor of two. Instead the rotation amount was set to 19. The reason for that and the other changes can be found in the Section 4. The block cipher comes in three versions, PAGES-512, PAGES-768, and PAGES-1024, where the number denotes the key size.

This paper is organized as follows: Section 2 contains the preliminaries like definitions and so on. Section 3 comprises the assembly of PAGES. The design rationale is given in Section 4. We present the software performance of PAGES in Section 5 and conclude in Section 6.

*dieterschmidt@usa.com



(a) Encryption (b) Decryption

Figure 1: A round of PAGES

2 Preliminaries

$\mathbb{F}_2^{128}$ denotes the 128 dimensional vector space over GF(2). The following symbols are used to build 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 left circular shift of m positions, $\ggg m$ denotes the right circular shift of m positions, GCD denotes the greatest common divisor, and LCM denotes the least common multiple. PAGES has a block length of 256 bits and is divided into two halves of equal length (128 bit).

3 Assembly of PAGES

3.1 Encryption

The left half of PAGES is circular shifted by 19 position to the right, see figure 1a and the function encrypt of the reference implementation in the appendix A. Then the right half is XORed to the left half. The right half is then circular shifted by 7 bits to left. The round key is added modulo 2^{128} to the left half. Then the left half is added modulo 2^{128} to the right half. That concludes one round of encryption. The number of rounds can be 64, 96, or 128. The respective key sizes are 512 bit, 768 bit, or 1024 bit.

3.2 Decryption

The left half is subtracted from the right half modulo 2^{128}, see figure 1b and the function decrypt of the reference implementation in the appendix A. Then the round key is subtracted modulo 2^{128} from the left half. The right half is circular shifted to the right by 7 bits. The right half is XORed to the left half. The left half is then circular shifted to the left by 19 positions. That concludes one round of decryption.

3.3 Key Schedule

The key schedule of PAGES resembles the key schedule of the block cipher 1024, 1024XKS, 2048XKS-F, and 4096XKS-F, see [4, 5, 6]. The number of rounds can be 64, 96 , or 128. Divide the round number by 16 and you will get the size of the array the userkey is saved in (i.e. 4, 6, or 8), see the reference implementation in appendix A. Since the type of variable is __int128 the size of the key is 512 bit, 768 bit, and 1024 bit, respectively. The userkey is stored in the first 4, 6, or 8 round keys, see the function expand_key in the reference implementation in appendix A. The userkey is circular shifted by 61 bits to the left and stored in the next round keys. This is repeated 15 times until all the round keys have received their values. Then two 128 bit variables are set to 0 and encrypted. The output is stored in first two round keys or in the last round keys, depending on whether the macro FORWARD is defined. The block cipher is employed in Output Feedback Mode until all the round keys have received their values. The block cipher is now ready for encryption or decryption.

4 Design Rationale

The round key in Speck is added modulo 2 (XOR) to left half of the block cipher and the right half is XORed with left half. However, when it comes to linear cryptanalysis (see [2]) addition modulo 2^n is more secure. The bias or effectiveness $\epsilon = |p - 1/2| = \frac{1}{2^{i+1}}$, where p denotes the probability of the linear approximation and i is the bit position (lsb=0). This was presented by [8] and with a different proof by [3]. Since addition in PAGES is modulo 2^{128} and the plaintext requirement for success is roughly $1/\epsilon^2$, the key is added modulo 2^{128} to the left half of the block cipher and then the left half is added modulo 2^{128} to right half. The equation for the bias holds also for the right half, since the additon of the left half can be separated in the addditon of the left half without the key and the addition of the key itself (i.e. addition is associative). To have three different operation, namely addition modulo 2^{128}, XOR, and rotation, the addition of the right half to the left half of block cipher was replaced by XOR.

The concept of diffusion in cryptography is known since Claude E. Shannon, see [7]. If one looks at Speck and PAGES, diffusion means in that case, that every plaintext bit is rotated to every bit position, provided that the number of rounds is sufficient. In Speck128, the left half of the block cipher is rotated by 8 bits to the right. Since GCD(8,64)=8, where 64 denotes half the block length and the bit length of the addition, a possible bit flip in the left half of the plaintext is after eight rounds in the same position as it was in the plaintext. Consequently, the amount of rotation of the left half was not doubled to 16, but was set to 19. Since 19 and 128 are coprime, i.e. GCD(19,128)=1, a possible bit flip in the left half of the plaintext is rotated to all the positions in left half of the block cipher, if the number of rounds is 128. For the same reason, the amount of circular shift to the left of the right half was not doubled to 6, but was set to 7.

Since the circular shifts are opposed to each other, one also has the sum taken into account. If the rotations are in the same direction, one has to take the difference into consideration. The sum is 19+7=26. Since GCD(26,128)=2, a possible bit flip in the left half is rotated to 64 positions to the right half and vice versa (we did not consider the addition). If we

take 17 instead of 19, the sum ist 17+7=24 and GCD(24,128)=8. Then the bit flip would have been rotated to 16 positions from the right half to the left half and vice versa. Since odd+odd=even (or odd-odd=even), one must take even values where power of 2 is one. When we come back to Speck128, we can see that the right rotation of the left half would be better with 10 positions, since GCD(10,64)=2. Thus a bit flip would be circular shifted to 32 different positions.

The minimum number of rounds was set to 64. Carry propagation, which is the only non-linear operation in $\mathbb{F}_2^{128}$, is essentially local. Carry propagation comes to a halt, when the two addends have at the same bit position zeros. The probability for that is 1/4, if the addends are pseudorandom, since two bits can have the values 00, 01, 10, and 11 with equal probabilities. Thus the average length of carry propagation is four, that is a halfbyte. Since every addition in the cipher propagates the carry four additional positions in average, the number of rounds in which all the 128 bits of one half are changed is 32, if in the plaintext one bit is flipped. To achieve an additonal security margin, to number of rounds was set to at least 64.

In Speck128, the number of rounds, in which all the bits of one half are changed due to carry propagation, if in the plaintext one bit is flipped, is on average 16. Double that number, and you will have the minimum number rounds for Speck128, which is 32 for 128 bit key length. However, we decided that PAGES-768 and PAGES-1024 should have an extra security margin. Hence the number of rounds was set to 96 and 128, respectively. The number of rounds for Speck128/192 and Speck128/256 is 33 and 34, see [1], respectively. The bit changes that are due to operations between the halves were not taken into consideration on the carry propagation, except for addition in Speck.

5 Software Performance

On an AMD A6 6310 (Quadcore) with 1.8 GHz the optimized reference implementation runs with throughput of approx. 400 Mbit/s, 265 Mbit/s, and 200 Mbit/s for PAGES-512, PAGES-768, and PAGES-1024, respectively. This represents 36 cycles/byte, 54 cycles/byte, and 72 cycles/byte, respectively. The key schedule was not taken into consideration. The operating system was Linux Mint 17 (64 bit) and the C-compiler was gcc version 4.8.2 with optimization -O. The code size of the reference implementation is approx. 8.5 kB.

If the reference implementation source code is compiled with the options -funroll-loops und -O3 the throughput increases to 432 Mbit/s, 287 Mbit/s, and 213 Mbit/s for PAGES-512, PAGES-768, and PAGES-1024, respectively. This represents 33.3 cycles/byte, 50.2 cycles/byte, and 67.6 cycles/byte, respectively. The code size is here approx. 12.5 kB. Note that the decryption throughput is in both cases about 10 Mbit/s higher then the encryption performance.

The following table summarizes the results (TP denotes throughput, EC denotes encryption cost in cycles/byte):

Compiler Options	-O		-O3,-funroll-loops	
Cipher	TP Mbit/s	EC cyc/by	TP Mbit/s	EC cyc/by
PAGES-512	400	36.0	432	33.3
PAGES-768	265	54.0	287	50.2
PAGES-1024	200	72.0	213	67.6

6 Conclusion

I have presented the block cipher PAGES, which comes in three version PAGES-512, PAGES-768, and PAGES-1024. The number denotes thee key length, while the block size is 256 bit. The cryptographic community is invited to determine the security of the block cipher, and, if need be, make the necessary amendments. As its predecessor Speck, PAGES is free from any intellectual property claims.

References

[1] Beaulieu, Ray et.al.: *The Simon and Speck Families of Lightweight Block Ciphers*, IACR ePrint Archives, Report 2013/404 1, 4

[2] Matsui, Mitsuru: *Linear Cryptanalysis Method for DES Cipher*, in Tor Helleseth(ed.): Advances in Cryptology - EUROCRYPT 93, LNCS, Springer Verlag, Berlin, 1993 3

[3] Mukhopadhyay, Debdeep and Dipanwita RoyChowdhury: *Key Mixing in Block Cipher through Addition modulo* 2^n, IACR ePrint Archives, Report 2005/383 3

[4] Schmidt, Dieter: *1024 - A High Security Software Oriented Block Cipher*, IACR ePrint Archives, Report 2009/104 3

[5] Schmidt, Dieter: *1024XKS - A High Security Software Oriented Block Cipher Revisited*, IACR ePrint Archives, Report 2010/162 3

[6] Schmidt, Dieter: *2048XKS-F & 4096XKS-F - Two Software Oriented High Security Block Ciphers*, IACR ePrint Archives, Report 2013/136 3

[7] 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 3

[8] Wallén, Johan: *Linear Approximations of Addition modulo* 2^n, in Thomas Johansson(ed.): Fast Software Encryption (FSE) 2003, LNCS, Springer Verlag, Berlin, 2003 3

A Reference Implementation

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

Copyright 2015 by
```

```
Dieter Schmidt

This software is subject to the GNU General Public License.
This program is FREE software; you can redistribute
and/or modify it under the terms of the GNU General Public License
as published by the Free Software Foundation; either version 3 of the
license, or (at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY, without even the implied warranty of
MERCHENDABILITY or FITNESS FOR A PARTICULAR PURPOSE.
See the GNU General Public License for details.
You should have received a copy of the GNU General Public
License along with this program; if not, write to the

Free Software Foundation, Inc.,
59 Temple Place, Suite 330,
Boston, MA 02111-1307,
USA.

See http://www.gnu.org/licenses/gpl.txt for details.*/


#include<stdio.h>

#define INTLENGTH 128
#define NUMROUNDS 128 // 64 and 96 also possible
#define ROL(x,a) ((((x)<<(a))|((x)>>(INTLENGTH-(a)))))
#define ROR(x,a) ((((x)>>(a))|((x)<<(INTLENGTH-(a)))))
#define KEYLENGTH NUMROUNDS/16
#define ROTROUNDKEY 61
#define ROTROUNDDATA1 7
#define ROTROUNDDATA0 19

#define FORWARD

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

  unsigned long i;
  register unsigned __int128 a,b;

  a=data[0];b=data[1];
  for(i=0;i<NUMROUNDS;i++){
    a=ROR(a,ROTROUNDDATA0);
    a^=b;
    b=ROL(b,ROTROUNDDATA1);
    a+=keys[i];
    b+=a;
  }
  data[0]=a;data[1]=b;
  return;
}
```

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

  unsigned long i;
  register unsigned __int128 a,b;

  a=data[0];b=data[1];
  for(i=0;i<NUMROUNDS;i++){
    b-=a;
    a-=keys[NUMROUNDS-i-1];
    b=ROR(b,ROTROUNDDATA1);
    a^=b;
    a=ROL(a,ROTROUNDDATA0);
  }
  data[0]=a;data[1]=b;
  return;
}

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

  unsigned long i,j;
  unsigned __int128 data[2],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;
  for(i=0;i<(NUMROUNDS/2);i++){
    encrypt(data,keys);
    #ifdef FORWARD
      keys[2*i]=data[1];
      keys[2*i+1]=data[0];
    #else
      keys[NUMROUNDS-2-2*i]=data[1];
      keys[NUMROUNDS-2*i-1]=data[0];
    #endif
  }
  return;
}

int main(){

  unsigned __int128 data[2],userkey[KEYLENGTH],keys[NUMROUNDS];
```

```
  unsigned long i,j;

  data[0]=0;data[1]=1;
  i=(long) data[0];
  j=(long) data[1];
  printf("Before encryption %20lx%20lx\n",i,j);
  for(i=0;i<KEYLENGTH;i++) userkey[i]=i;
  expand_key(userkey,keys);
  encrypt(data,keys);
  i=(long) data[0];
  j=(long) data[1];
  printf("After encryption  %20lx%20lx\n",i,j);
  decrypt(data,keys);
  i=(long) data[0];
  j=(long) data[1];
  printf("After decryption  %20lx%20lx\n",i,j);
  return(0);
}
```