2048XKS-F & 4096XKS-F - Two Software Oriented High Security Block Ciphers

Dieter Schmidt*

8. January 2013

Abstract

2048XKS-F (eXtended Key Schedule-Feistel) is a Feistel cipher with a block length of 2048 bit and a key size of 4096 bit or 8192 bit, respectively. It uses the round function of the Substitution-Permutation-Networks (SPN) 1024 [11] and 1024XKS [12] as the f-function. 4096XKS-F is a Feistel cipher with a block length of 4096 bit and a key size of 8192 bit or 16384 bit, respectively. It uses the round function of the Substitution-Permutation-Network (SPN) 2048XKS as the f-function. Both 2048XKS-F and 4096XKS-F have 32 rounds. Additionally, there are #define statements in the reference implementations to control which of the functions are compiled first, e.g. the diffusion layer or the s-box layer. In total, there are 6 #define statements in each reference implementation, making up 64 different ciphers. 2048XKS-F and 4096XKS-F are designed for 32 bit microprocessors with an integer hardware multiplier.

1 Introduction

IDEA and its predecessor PES (see [5, 6, 7]) was the first encryption algorithm, that used incompatible group operations to get diffusion and confusion (see [15]). IDEA uses addition modulo 2 (XOR), addition modulo 2^{16} and multiplication modulo $2^{16} + 1$. IDEA has not been broken in the open literature. However, IDEA is about twenty years old and due to its block length of 64 bits and keysize of 128 bits not state of the art.

Interestingly, James Bamford (see [1]) gives a note, what the National Security Agency (NSA) can achieve. He writes: *Another factor*

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

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.

A key space of 2^{256} equals about 10^{77}. The number of electrons in the universe equals about 10^{80}. So if you store all 256 bit keys, you have to control every thousandth electron of the universe, which is, of course, infeasible. Bringing down such an effort to 2^{40} or 2^{50}, that is what a computer can digest, is an art, which I do not understand.

But the meaning of Bamfords text is twofold: There are much more block ciphers than asymmetric encryption algorithms (see, for example, Wikipedia). A block cipher and an asymmetric encryption algorithm are used in a hybrid scheme. The payload is encrypted with the block cipher and the block cipher key is encrypted with asymmetric encryption algorithm (e.g. RSA, ElGamal, ECC). When the number of asymmetric encryption algorithms is small against the number of block ciphers, then the breaking of that asymmetric encryption algorithm should yield more benefits then the breaking of a block cipher. But the NSA breaks the block ciphers, which in turn leaves only one conclusion: the NSA has difficulties in solving asymmetric encryption algorithms with the current key size.

2048XKS-F has a block length of 2048 bits and a keyspace of 4096 bits or 8192 bits. 4096XKS-F has a block length of 4096 bits and a keyspace of 8192 bits or 16384 bits. Both use addition modulo 2 (XOR), addition modulo 2^{32}, addition modulo 2^{256}, circular shifts to the left of 32 bit integers, and multiplication modulo $2^{32} - 1$. Note that bitwise rotation (a circular shift by a to n positions to the left) can be expressed as $2^n * a \mod 2^{32} - 1$, if $a < 2^{32} - 1$.

At first, the key sizes of 2048XKS-F and 4096XKS-F seem to be exaggerated. But mind the following: The lifetime of a block cipher is about 20 years (DES,Triple-DES). The secrecy period of official documents in Germany is in general 30 years. So in total you must guard 50 years against cryptanalysis. But you can not anticipate every cryptanalytical attack. That is, why the key space is so high.

2048XKS-F and 4096XKS-F are Feistel-Networks. The s-boxes are based on multiplication modulo $2^{32} - 1$. This was invented by Daemen et al. in block cipher MMB (see [2]). The permutation is a modified Pseudo-Hadamard-Tranformation taken from SAFER (see [8, 9]). The key schedule is a modification from the key schedule of Blowfish (see [14]).

2048XKS-F and 4096XKS-F have 32 rounds. For decryption, only the order of the keys has to be inverted, but the individual key stays the same. For further insight, please turn to the function invert_keys in the reference implementations.

2 The Algorithm of 2048XKS-F and 4096XKS-F

2.1 The S-Boxes

2048XKS-F and 4096XKS-F are Feistel ciphers. It uses as building blocks multiplication modulo $2^{32}-1$ as s-box and a modified diffusion layer from SAFER. Keys are applied before and after the s-boxes. 4096XKS-F has 64 s-boxes (multiplication modulo $2^{32}-1$), 2048XKS-F has 32 s-boxes. Each s-box has 32 bits, making a total width of the f-function of 2048 bits in 4096XKS-F or 1024 bit in 2048XKS-F. Let us denote in this article addition, subtraction and multiplication modulo 2^n-1 by respectively +,- and $\times$, ordinary multiplication by $*$, integer division by $\lfloor \div \rfloor$,XOR by $\oplus$, rotation by a bits to the left by $\lll$a. .

Multiplication modulo 2^n-1 as s-box was first used by Daemen et. al. [2, 3, 4]. The studied function is:

$$f^a(x) = \begin{cases} a \times x & \text{if} \quad x < 2^n - 1 \\ 2^n - 1 & \text{if} \quad x = 2^n - 1 \end{cases} \tag{1}$$

The calculation is easy:

$$a * b \ \ mod(2^n-1) = (a * b \ \ mod(2^n) + \lfloor \frac{a*b}{2^n} \rfloor)(1 + \frac{1}{2^n}) \tag{2}$$

The first righthand term is obtained by taking the least significant bits of the product, the second term by taking the remaining bits and shifting them to the right by n bits and add that to the first term. If a carry (i.e. bit 32 is set) results from that addition the result is incremented by 1. Note that [2] gives a wrong formula. It has been corrected in chapter 11 of Joan Daemens Ph.D. thesis [4]. Note that the last factor of the righthandside of the equation is not distributive.

Multiplication modulo 2^n-1 has interesting properties. A multiplication by 2 modulo 2^n-1 is equivalent by a rotation to left by one. Similarly $2^k \times a = a \lll k$. Further material can be found in [3].

The critical probability of the s-boxes with regard to differential cryptanalysis is 2^{-9}.

2.1.1 The order of s-boxes in 2048XKS-F

In 2048XKS-F the s-boxes are the same as in 1024XKS, if FACTORS (see Compiler Options) is defined. If FACTORS is not defined, the

decryption factors of 1024XKS are used. See the function encryption_factors at the top of the reference implementation for further details. There is an increase by two for the rotation values for each step ranging from 0 (left most s-box) to 30 (second right most s-box). The rotation values for the s-boxes with even rotation values are shown by the table:

position	0	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30
rotation	0	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30

For the s-boxes, which have odd rotation values the table is shown here:

position	1	3	5	7	9	11	13	15	17	19	21	23	25	27	29	31
rotation	7	5	3	1	31	29	27	25	23	21	19	17	15	13	11	9

If FACTORS is defined, the starting value of the multiplication is 0x25F1CDB and the rotation is to the left. If FACTORS is not defined, the starting value of the multiplication is 229459604 (the multiplicative inverse of 0x25F1CDB) and the rotation is to the right.

For further insight study the functions encryption_factors at the beginning of the reference implementation.

2.1.2 The order of s-boxes in 4096XKS-F

In 4096XKS-F the s-boxes are the same as in 2048XKS, if FACTORS (see Compiler Options) is defined. If FACTORS is not defined, the decryption factors of 2048XKS are used. See the function encryption_factors at the top of the reference implementation for further details.

In 4096XKS-F the s-boxes, which contain the even rotation numbers, are same as in 1024 [11]. To fill the left half of the 64 s-boxes, we use the factor 0x25F1CDB. There is an increase by two for the rotation values for each step ranging from 0 (left most s-box) to 30. The rotation values for the s-boxes with even rotation values are shown by the table:

position	0	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30
rotation	0	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30

For the s-boxes, which have odd rotation values, the table is shown here:

position	1	3	5	7	9	11	13	15	17	19	21	23	25	27	29	31
rotation	7	5	3	1	31	29	27	25	23	21	19	17	15	13	11	9

To fill the right half of the 64 s-boxes, we use the factor 229459604. This is the decryption factor of 0x25F1CDB or the multiplicatve inverse. The rotation values for s-boxes with even rotation are shown in the table:

position	32	34	36	38	40	42	44	46	48	50	52	54	56	58	60	62
rotation	0	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30

For the s-boxes, which have odd rotation values the table is shown here:

position	33	35	37	39	41	43	45	47	49	51	53	55	57	59	61	63
rotation	17	15	13	11	9	7	5	3	1	31	29	27	25	23	21	19

If FACTORS is defined, all the circular shifts are to the left. If FACTORS is not defined, all circular shifts are to the right and the first factor (to fill the left half of the s-box layer) is 229459604 and the factor 0x25F1CDB is used to fill the right half of the s-box layer.

For further insight study the functions encryption_factors at the beginning of the reference implementation.

2.2 Addition modulo 2^{256}

Addition modulo 2^{256} was introduced to give an upper bound for linear cryptanalysis. If we take [10], we can have an upper bound for linear cryptanalysis without being forced to examine the diffusion layer or the s-boxes.

2048XKS-F has two blocks of 1024 bits. This means addition modulo 2^{256} is applied four times, from left to right, sometimes after the s-boxes, sometimes before the s-boxes. 4096XKS-F has two blocks of 2048 bits. This means addition modulo 2^{256} is applied eight times, from left to right, sometimes after the s-boxes, sometimes before the s-boxes. The input or the output of one addition modulo 2^{256} is eight s-boxes.

One can argue that all the keys should be applied by addition modulo 2^{256}, so one can use less rounds. But the addition modulo 2 (XOR) of some keys is there to make the cryptanalysis more difficult by using different groups or to avoid symmetry attacks.

2.3 Diffusion Layer

The diffusion layer has as parent the diffusion layer from SAFER [8, 9]. However, there are three modifications:

1. 64 (4096XKS-F) or 32 (2048XKS-F) blocks instead of eight.
2. Four bytes instead of one byte as primitive unit. See [14].
3. Before the addition primitive units are being rotated.

Point two is clear. In a modern PC the CPU has a register size of four bytes, sometimes eight bytes. Obviously this will increase the speed.

The Pseudo-Hadamard-Transform is defined as:

$$b_1 = 2a_1 + a_2 \tag{3}$$
$$b_2 = a_1 + a_2 \tag{4}$$

It can be rewritten:

$$b_2 = a_1 + a_2 \tag{5}$$
$$b_1 = a_1 + b_2 \tag{6}$$

The Pseudo-Hadamard-Transform has one disadvantage. The least significant bit of b_1 is not dependent on a_1. Schneier et. al. [14] were aware, that b_1 is not dependent on the most significant bit of a_1. But there is no word on the least significant bit of a_1 (or at least I did not see it). Because $b_1 = 2a_1 + a_2$ the least significant bit of b_1 is a function of a_2 and not of a_1. Thus the least significant bit of b_1 is incomplete.

Therefore the Pseudo-Hadamard-Transform (PHT) was supplemented by rotation. For example, if a_0 is the leftmost element of the array, the equation is now:

$$a_1 = a_1 + (a_0 \lll 1) \tag{7}$$
$$a_0 = a_0 + (a_1 \lll 2) \tag{8}$$

The other rotation values were pseudo-randomly produced for each round. However, it is worth noting, that an element of the array with even index number has an odd rotation value and vice versa. The Inverse-Pseudo-Hadamard-Transform (IPHT) may also be used. For example, if b_0 is the leftmost element of the array, the equation is now:

$$b_0 = b_0 - (b_1 \lll 2) \tag{9}$$

$$b_1 = b_1 - (b_0 \lll 1) \tag{10}$$

The f-function of 4096XKS-F has a width of 64*32 bits=2048 bits. Each element of the array is rotated once, except for one element, that is not rotated, because rotation ranges from 0 to 31,which are 32 values. As the array has 64 elements, the rotation values start at b_{32} again, i.e. b_i and b_{i+32} have the same rotation value for $0 <= i < 32$. The one element, that is not rotated has always an odd index number. Every round of the Pseudo-Hadamard-Transform has a distinct rotation pattern, because rotation values were pseudo-randomly selected, except for 1 and 2 for the two leftmost elements of the array.

The f-function of 2048XKS-F has a width of 32*32 bis=1024 bits. There are 32 rotation values, 0 to 31, without repetition.

For more information, please turn to the functions pht and ipht in the reference implementation.

2.4 The Key Schedule

The round key generation of 2048XKS-F is as follows: First calculate the round keys in the 1024 manner, i.e. do the rotation by 455 bit to left. Then take a 2048 bit all zero string and let it pass through the algorithm. The resulting bit string is the first round key, 1024 bit before the s-box layer, 1024 bit after the s-box layer. Let the algorithm work in Output Feedback Mode (OFB). Each time the bit string has passed through the algorithm, a round key is assigned that bit string in ascending order. Given the number of rounds (32),the Output Feedback Mode (OFB) is applied 32 times. This key schedule was inspired by Blowfish [13].

The round key generation of 4096XKS-F is as follows: First calculate the round keys in the 1024 [11] manner, i.e. do the rotation by 455 bit to left. Then take a 4096 bit all zero string and let it pass through the algorithm. The resulting bit string is the first round key, 2048 bit before the s-box layer, 2048 bit after the s-box layer. Let the algorithm work in Output Feedback Mode (OFB). Each time the bit string has passed through the algorithm, a round key is assigned that bit string in ascending order. Given the number of rounds (32),the Output Feedback Mode (OFB) is applied 32 times.

However, that "forward mode" has a disadvantage: The first half of the first round key is assigned the bit string of the first OFB round. When encryption is applied, the first half of the first round key and the bit string have the same value. When they are added modulo

2 (XOR), the result is the all zero string. As the s-boxes left the zeroes unchanged, the first non-zero input is the second half of the first round key, or the second half of the user key. However, this is the only "error" that occurs in the "forward mode".

To avoid the "error", 4096XKS-F and 2048XKS-F have a mode of key scheduling which I describe as the "backward mode". This means that Output Feedback Mode is still employed, but the round keys are assigned the value in descending order, i.e the last key of the f-function is assigned the value first. This "backward mode" has not the same error as the "forward mode".

To distinguish the modes in the reference implementation, there is a variable for the preprocessor named #define FORWARD. When the #define statement is true, then the key scheduling is in "forward mode". If the #define statement is not true, then the key scheduling is in "backward mode". To accomplish that, you could erase the #define statement or leave it as a commentary, i.e. to the beginning of the #define statement insert /* und the end of the statement insert */. The code of the reference implementation, which are influenced by the #define statement, are the functions encrypt and decrypt quite at the end of the reference implementation.

3 Compiler Options

The reference implementations was build for the compiler gcc (Cygwin B20.1 for Windows 95). Quite at the beginning of the reference implementation there six #define statements. Each #define statement can be on or of. In the reference implementation, all #define statements are on. To switch on #define statement of, simple mark the statement as a commentary by putting a /* at the beginning and a */ at the end. In total, the six #define statements are making up $2^6 = 64$ different ciphers.

#define FORWARD: If FORWARD is defined, the keys are placed in the cipher in ascending order. If FORWARD is not defined, the keys are placed in the cipher in descending order. Also see subsection Key Schedule.

#define BIG_KEY: If BIG_KEY is defined, the user key has the size of 16384 bits (4096XKS-F) or 8192 bits (2048XKS-F). If BIG_KEY is not defined, the user key has the size of 8192 bits (4096XKS-F) or 4096 bits (2048XKS-F).

#define TOP: If TOP is defined, the XOR, s-boxes and addition modulo 2^{256} are carried out before the diffusion layer is applied. If TOP is undefined, the diffusion layer (pht or ipht) is carried out first.

#define PHT: If PHT is defined, the diffusion layer is made up of the Pseudo-Hadamard-Transform (pht). If PHT is not defined, the diffusion layer is made up of the Inverse-Pseudo-Hadamard-Transform (ipht).

#define PRIMARY: If PRIMARY is defined, the order of execution is XOR, s-boxes, addition modulo 2^{256}. If PRIMARY is not defined, the order of execution is addition modulo 2^{256}, s-boxes, XOR.

#define FACTORS: If FACTORS is defined, the order of the s-boxes is the same as in 2048XKS (4096XKS-F) or in 1024XKS (2048XKS-F) encryption mode. If FACTORS is not defined, the order of the s-boxes is the same as in 2048XKS (4096XKS-F) or in 1024XKS (2048XKS-F) decryption mode.

4 Implementation Consideration

The reference implementation is programmed in the language C. Unfortunately, this lacks instructions, which in assembler (processor language) are quite common. This makes the reference implementation a bit clumsy. In the reference implementation you must have 64 bit variables to allow for the carry. These 64 bit variables are then shifted to right by 32 digits to get added in the next round of calculations. This is true for the addtion modulo 2^{256} and the s-boxes (see functions modmult and crypt). Also the rotations (see function pht and ipht) are made up with the shift to left, a shift to right, and OR (see the definitions of the reference implementation). For example, in the Intel Architecture for 32 bit microprocessors (IA32), the s-boxes are pogrammed in this way:

```
MOV EAX,data
MOV EBX,factor
MUL EBX
ADD EAX,EDX
ADC EAX,0
MOV data,EAX
```

Also the addition modulo 2^{256} looks like this:

```
MOV EAX,data0
MOV EBX,key0
ADD EAX,BX
MOV data0,EAX
MOV EAX,data1
MOV EBX,key1
```

```
ADC EAX,EBX
MOV data1,EAX
          .
          .
          .
MOV EAX,data7
MOV EBX,key7
ADC EAX,EBX
MOV data7,EAX
```

The rotations of the diffusion layer are a part of the IA32. For the first values of the function pht (a_0 and a_1), the programming looks like this:

```
MOV EAX,a0
MOV EBX,a1
MOV ECX,EAX
ROL ECX,1
ADD EBX,ECX
MOV EDX,EBX
ROL EDX,2
ADD EAX,EDX
MOV a1,EBX
MOV a0,EAX
```

In the ideal case, the programming of 2048XKS-F and 4096XKS-F should be all in assembler.

5 Derivatives of the XKS Family

If a 64 bit microprocessor is available, then multiplication modulo $2^{64} - 1$ can be used as a s-box. In the AMD world simply replace the 32 bit instructions with 64 bit instructions, that's it (see Implementation Consideration). In the Intel World it is a bit more complicated, because Intel microprocessor calculate only the top 64 bit or the bottom 64 bit. Hence, two multiplications have to be used here.

A factor f must not only fulfill $gcd(f, 2^{64} - 1) = 1$, but must be immune to differential and linear cryptanalysis. The algorithm presented in [2, 3] to reduce vulnerability against differential cryptanalysis can be used straightforword. To reduce vulnerability against linear cryptanalysis, an algorithm has to be developed or other measures have to be taken. This can be, as is the case with the XKS family, e.g. [10].

For 32 bit microprocessor multiplication modulo $2^{32} + 1$ and for 64

bit microprocessor multiplication modulo $2^{64} + 1$ can be used. They are similar to multiplication modulo $2^n - 1$, only the high bits are subtracted from the low bits. In multiplication modulo $2^n - 1$, the high bits are added to the low bits (see Implementation Consideration). However, measures against standard cryptanalytical techniques have to be found. See [7] for further information.

6 Intellectual Property

2048XKS-F and 4096XKS-F are free. The reference implementations are covered by the GNU General Public License.

References

[1] Bamford, James: *The Shadow Factory. The Ultra-Secret NSA from 9/11 to the Eavesdropping on America*, p. 138, First Anchor Books Edition, New York, 2009

[2] Daemen, Joan; Luc Van Linden; René Govaerts and Joos Vandewalle: Propagation Properties of Multiplication Modulo $2^n - 1$, appeared in the *Proceedings of the 13th Symposium on Information Theory in Benelux, Werkgemeenschap voor Informatie- en Communicatietheorie*, pp. 111-118, 1992, Available from: `https://www.cosic.esat.kuleuven.be/publications/article-136.pdf`

[3] Daemen, Joan; René Govaerts and Joos Vandewalle: Block Ciphers Based on Modular Arithmetic, in W. Wolfowicz (Ed.): *Proceedings of the 3rd Symposium on State and Progress of Research in Cryptography*, Fondazione Ugo Bordoni, Rome, 1993. Available from: `https://www.cosic.esat.kuleuven.be/publications/article-277.pdf`

[4] Daemen, Joan: *Cipher and Hash Function Design, Strategies based on linear and differential Cryptanalysis*, Ph.D. thesis, KU Leuven, Belgium. Available from: `https://homes.esat.kuleuven.be/~cosicart/ps/JD-9500`, 1995

[5] Lai, Xuejia and James Massey: A Proposal for a New Block Encryption Standard, in Ivan Damgård (Ed.): *Advances in Cryptology - EUROCRYPT '90*, Springer Verlag, Berlin, 1991

[6] Lai, Xuejia, James Massey and Sean Murpy: Markov Ciphers and Differential Cryptanalysis, in Donald Davies (Ed.): *Advances in Cryptology - EUROCRYPT '91*, Springer Verlag, Berlin, 1991

[7] Lai, Xuejia: *On the Design and Security of Block Ciphers*, Ph.D. thesis at ETH Zürich, Switzerland, Hartung-Gorre Verlag, Konstanz, Germany, 1992

[8] Massey, James: SAFER K-64: A Byte-Oriented Block-Cipher Algorithm, in Ross Anderson (Ed.): *Fast Software Encryption*, Springer Verlag, Berlin, 1994

[9] Massey, James: SAFER K-64: One year later, in Bart Preneel (Ed.): *Fast Software Encryption*, Springer Verlag, Berlin, 1995

[10] Mukhopadhyay, Debdeep and Dipanwita RoyChowdhury: *Key Mixing in Block Cipher through Addition modulo* 2^n, available from: `http://eprint.iacr.org/2005/383.pdf`

[11] Schmidt, Dieter: *1024 - A High Security Software Oriented Block Cipher*, ePrint Archive of the IACR, Report 2009/104, available from `http://eprint.iacr.org`

[12] Schmidt, Dieter: *1024XKS - A High Security Software Oriented Block Cipher*, ePrint Archive of the IACR, Report 2010/162, available from `http://eprint.iacr.org`

[13] Schneier, Bruce: Description of a New Variable Length Key, 64-Bit Block Cipher, in Ross Anderson (Ed.): *Fast Software Encryption - Cambrigde Security Workshop*, Springer Verlag, Berlin, 1994

[14] Schneier, Bruce; John Kelsey, Doug Whiting, David Wagner, Chris Hall and Niels Ferguson: *Twofish: A 128-Bit Block Cipher*, 1998. Available from: `http://www.schneier.com/twofish.html`

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

A Reference Implementation of 2048XKS-F

```
#include<stdio.h>

#define NUM_ROUNDS 32
#define INT_LENGTH 32
#define ROL(x,a) (((x)<<(a))|((x)>>(INT_LENGTH-(a))))
```

```
#define ROR(x,a) (((x)<<(INT_LENGTH-(a))|((x)>>(a))))
#define WIDTH 32
#define ROTROUND 455

#define FORWARD
#define BIG_KEY
#define FACTORS
#define PRIMARY
#define PHT
#define TOP

#ifdef FACTORS

void encryption_factors(unsigned long e_factors[WIDTH]){
  unsigned long i;

  e_factors[0]=0x025F1CDB;
  for(i=0;i<(WIDTH/2);i++){
    if(i!=0) e_factors[2*i]=ROL(e_factors[0],2*i);
    e_factors[2*i+1]=ROL(e_factors[0],(WIDTH+7-2*i)%WIDTH);
  }

}

#else

void encryption_factors(unsigned long d_factors[WIDTH]){
  unsigned long i;

  d_factors[0]=229459604;
  for(i=0;i<(WIDTH/2);i++){
    if(i!=0) d_factors[2*i]=ROR(d_factors[0],2*i);
    d_factors[2*i+1]=ROR(d_factors[0],(WIDTH+7-2*i)%WIDTH);
  }
}

#endif

unsigned long modmult(unsigned long factor1,unsigned long factor2){
  unsigned long long f1,f2,ergebnis,k;

  f1=(unsigned long long) factor1;
  f2=(unsigned long long) factor2;
```

```
  ergebnis=f1*f2;
  k=(ergebnis>>INT_LENGTH);
  ergebnis&=0xFFFFFFFF;
  ergebnis+=k;
  ergebnis+=(ergebnis>>INT_LENGTH) & 1;
  return(ergebnis & 0xFFFFFFFF);
}

void invert_keys(unsigned long keys[2*NUM_ROUNDS][WIDTH]){
  unsigned long i,j,intermediate[2*NUM_ROUNDS][WIDTH];

  for(i=0;i<(2*NUM_ROUNDS);i++){
     for(j=0;j<WIDTH;j++){
       intermediate[i][j]=keys[i][j];
     }
  }
  for(i=0;i<NUM_ROUNDS;i++){
    for(j=0;j<WIDTH;j++){
      keys[(2*NUM_ROUNDS-2)-2*i][j]=intermediate[2*i][j];
      keys[(2*NUM_ROUNDS-1)-2*i][j]=intermediate[2*i+1][j];
    }
  }
}



#ifdef BIG_KEY

void key_schedule(unsigned long user_key[8][WIDTH],\
  unsigned long key[2*NUM_ROUNDS][WIDTH]){
  unsigned long i,j;

  for(i=0;i<8;i++){
    for(j=0;j<WIDTH;j++){
      key[i][j]=user_key[i][j];
    }
  }
  for(i=1;i<(NUM_ROUNDS/4);i++){
    for(j=0;j<WIDTH;j++) {
      key[8*i+7][j]=(key[8*(i-1)+((j+((WIDTH*INT_LENGTH-ROTROUND)\
      /INT_LENGTH))/WIDTH*7)]\
      [(j+((WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH))\
      %WIDTH]<<(ROTROUND%INT_LENGTH))|\
```

```
(key[8*(i-1)+((j+(WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH+1)/WIDTH)*7]\
[(j+(WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH+1)%WIDTH]>>\
(INT_LENGTH-ROTROUND%INT_LENGTH));

key[8*i+6][j]=(key[8*(i-1)+7-(j+((WIDTH*INT_LENGTH-ROTROUND)\
/INT_LENGTH))/WIDTH]\
[(j+((WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH))\
%WIDTH]<<(ROTROUND%INT_LENGTH))\
|(key[8*(i-1)+7-(j+(WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH+1)/WIDTH]\
[(j+(WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH+1)%WIDTH]>>\
(INT_LENGTH-ROTROUND%INT_LENGTH));

key[8*i+5][j]=(key[8*(i-1)+6-(j+((WIDTH*INT_LENGTH-ROTROUND)\
/INT_LENGTH))/WIDTH]\
[(j+((WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH))\
%WIDTH]<<(ROTROUND%INT_LENGTH))\
|(key[8*(i-1)+6-(j+(WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH+1)/WIDTH]\
[(j+(WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH+1)%WIDTH]>>\
(INT_LENGTH-ROTROUND%INT_LENGTH));

key[8*i+4][j]=(key[8*(i-1)+5-(j+((WIDTH*INT_LENGTH-ROTROUND)\
/INT_LENGTH))/WIDTH]\
[(j+((WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH))\
%WIDTH]<<(ROTROUND%INT_LENGTH))\
|(key[8*(i-1)+5-(j+(WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH+1)/WIDTH]\
[(j+(WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH+1)%WIDTH]>>\
(INT_LENGTH-ROTROUND%INT_LENGTH));

key[8*i+3][j]=(key[8*(i-1)+4-(j+((WIDTH*INT_LENGTH-ROTROUND)\
/INT_LENGTH))/WIDTH]\
[(j+((WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH))\
%WIDTH]<<(ROTROUND%INT_LENGTH))\
|(key[8*(i-1)+4-(j+(WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH+1)/WIDTH]\
[(j+(WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH+1)%WIDTH]>>\
(INT_LENGTH-ROTROUND%INT_LENGTH));


key[8*i+2][j]=(key[8*(i-1)+3-(j+((WIDTH*INT_LENGTH-ROTROUND)\
/INT_LENGTH))/WIDTH]\
[(j+((WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH))\
%WIDTH]<<(ROTROUND%INT_LENGTH))\
|(key[8*(i-1)+3-(j+(WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH+1)/WIDTH]\
[(j+(WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH+1)%WIDTH]>>\
```

```
      (INT_LENGTH-ROTROUND%INT_LENGTH));

      key[8*i+1][j]=(key[8*(i-1)+2-(j+((WIDTH*INT_LENGTH-ROTROUND)\
      /INT_LENGTH))/WIDTH]\
      [(j+((WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH))\
      %WIDTH]<<(ROTROUND%INT_LENGTH))\
      |(key[8*(i-1)+2-(j+(WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH+1)/WIDTH]\
      [(j+(WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH+1)%WIDTH]>>\
      (INT_LENGTH-ROTROUND%INT_LENGTH));


      key[8*i][j]=(key[8*(i-1)+1-(j+((WIDTH*INT_LENGTH-ROTROUND)\
      /INT_LENGTH))/WIDTH]\
      [(j+((WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH))\
      %WIDTH]<<(ROTROUND%INT_LENGTH))\
      |(key[8*(i-1)+1-(j+(WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH+1)/WIDTH]\
      [(j+(WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH+1)%WIDTH]>>\
      (INT_LENGTH-ROTROUND%INT_LENGTH));

    }
  }
}


#else

void key_schedule(unsigned long user_key[4][WIDTH],\
  unsigned long key[2*NUM_ROUNDS][WIDTH]){
  unsigned long i,j;

  for(i=0;i<4;i++){
    for(j=0;j<WIDTH;j++){
      key[i][j]=user_key[i][j];
    }
  }
  for(i=1;i<(NUM_ROUNDS/2);i++){
    for(j=0;j<WIDTH;j++) {
      key[4*i+3][j]=(key[4*(i-1)+((j+((WIDTH*INT_LENGTH-ROTROUND)\
      /INT_LENGTH))/WIDTH*3)]\
      [(j+((WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH))\
      %WIDTH]<<(ROTROUND%INT_LENGTH))|\
      (key[4*(i-1)+((j+(WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH+1)/WIDTH)*3]\
      [(j+(WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH+1)%WIDTH]>>\
```

```
      (INT_LENGTH-ROTROUND%INT_LENGTH));

      key[4*i+2][j]=(key[4*(i-1)+3-(j+((WIDTH*INT_LENGTH-ROTROUND)\
      /INT_LENGTH))/WIDTH]\
      [(j+((WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH))\
      %WIDTH]<<(ROTROUND%INT_LENGTH))\
      |(key[4*(i-1)+3-(j+(WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH+1)/WIDTH]\
      [(j+(WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH+1)%WIDTH]>>\
      (INT_LENGTH-ROTROUND%INT_LENGTH));

      key[4*i+1][j]=(key[4*(i-1)+2-(j+((WIDTH*INT_LENGTH-ROTROUND)\
      /INT_LENGTH))/WIDTH]\
      [(j+((WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH))\
      %WIDTH]<<(ROTROUND%INT_LENGTH))\
      |(key[4*(i-1)+2-(j+(WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH+1)/WIDTH]\
      [(j+(WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH+1)%WIDTH]>>\
      (INT_LENGTH-ROTROUND%INT_LENGTH));


      key[4*i][j]=(key[4*(i-1)+1-(j+((WIDTH*INT_LENGTH-ROTROUND)\
      /INT_LENGTH))/WIDTH]\
      [(j+((WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH))\
      %WIDTH]<<(ROTROUND%INT_LENGTH))\
      |(key[4*(i-1)+1-(j+(WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH+1)/WIDTH]\
      [(j+(WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH+1)%WIDTH]>>\
      (INT_LENGTH-ROTROUND%INT_LENGTH));

    }
  }
}

#endif

#ifdef PHT

void pht(unsigned long a[WIDTH]){
  unsigned long i,b[WIDTH];

  a[1]+=ROL(a[0],1);
  a[0]+=ROL(a[1],2);
  a[3]+=ROL(a[2],7);
  a[2]+=ROL(a[3],16);
  a[5]+=ROL(a[4],13),
```

```
a[4]+=ROL(a[5],30);
a[7]+=ROL(a[6],19);
a[6]+=ROL(a[7],12);
a[9]+=ROL(a[8],25);
a[8]+=ROL(a[9],26);
a[11]+=ROL(a[10],31);
a[10]+=ROL(a[11],8);
a[13]+=ROL(a[12],5);
a[12]+=ROL(a[13],22);
a[15]+=ROL(a[14],11);
a[14]+=ROL(a[15],4);
a[17]+=ROL(a[16],17);
a[16]+=ROL(a[17],18),
a[19]+=ROL(a[18],23);
a[18]+=a[19];
a[21]+=ROL(a[20],29);
a[20]+=ROL(a[21],14);
a[23]+=ROL(a[22],3);
a[22]+=ROL(a[23],28);
a[25]+=ROL(a[24],9);
a[24]+=ROL(a[25],10);
a[27]+=ROL(a[26],15);
a[26]+=ROL(a[27],24);
a[29]+=ROL(a[28],21);
a[28]+=ROL(a[29],6);
a[31]+=ROL(a[30],27);
a[30]+=ROL(a[31],20);
for(i=0;i<(WIDTH/2);i++){
  b[i]=a[2*i];
  b[i+(WIDTH/2)]=a[2*i+1];
}

b[1]+=ROL(b[0],1);
b[0]+=ROL(b[1],2);
b[3]+=ROL(b[2],11);
b[2]+=ROL(b[3],20);
b[5]+=ROL(b[4],21);
b[4]+=ROL(b[5],6);
b[7]+=ROL(b[6],31);
b[6]+=ROL(b[7],24);
b[9]+=ROL(b[8],9);
b[8]+=ROL(b[9],10);
b[11]+=ROL(b[10],19);
```

```
b[10]+=ROL(b[11],28);
b[13]+=ROL(b[12],29);
b[12]+=ROL(b[13],14);
b[15]+=ROL(b[14],7);
b[14]+=b[15];
b[17]+=ROL(b[16],17);
b[16]+=ROL(b[17],18);
b[19]+=ROL(b[18],27);
b[18]+=ROL(b[19],4);
b[21]+=ROL(b[20],5);
b[20]+=ROL(b[21],22);
b[23]+=ROL(b[22],15);
b[22]+=ROL(b[23],8);
b[25]+=ROL(b[24],25);
b[24]+=ROL(b[25],26);
b[27]+=ROL(b[26],3);
b[26]+=ROL(b[27],12);
b[29]+=ROL(b[28],13);
b[28]+=ROL(b[29],30);
b[31]+=ROL(b[30],23);
b[30]+=ROL(b[31],16);
for(i=0;i<(WIDTH/2);i++){
  a[i]=b[2*i];
  a[i+(WIDTH/2)]=b[2*i+1];
}
a[1]+=ROL(a[0],1);
a[0]+=ROL(a[1],2);
a[3]+=ROL(a[2],15);
a[2]+=ROL(a[3],24);
a[5]+=ROL(a[4],29);
a[4]+=ROL(a[5],14);
a[7]+=ROL(a[6],11);
a[6]+=ROL(a[7],4);
a[9]+=ROL(a[8],25);
a[8]+=ROL(a[9],26);
a[11]+=ROL(a[10],7);
a[10]+=ROL(a[11],16);
a[13]+=ROL(a[12],21);
a[12]+=ROL(a[13],6);
a[15]+=ROL(a[14],3);
a[14]+=ROL(a[15],28);
a[17]+=ROL(a[16],17);
a[16]+=ROL(a[17],18);
```

```
a[19]+=ROL(a[18],31);
a[18]+=ROL(a[19],8);
a[21]+=ROL(a[20],13);
a[20]+=ROL(a[21],30);
a[23]+=ROL(a[22],27);
a[22]+=ROL(a[23],20);
a[25]+=ROL(a[24],9);
a[24]+=ROL(a[25],10),
a[27]+=ROL(a[26],23);
a[26]+=a[27];
a[29]+=ROL(a[28],5);
a[28]+=ROL(a[29],22);
a[31]+=ROL(a[30],19);
a[30]+=ROL(a[31],12);
for(i=0;i<(WIDTH/2);i++){
  b[i]=a[2*i];
  b[i+(WIDTH/2)]=a[2*i+1];
}
b[1]+=ROL(b[0],1);
b[0]+=ROL(b[1],2);
b[3]+=ROL(b[2],19);
b[2]+=ROL(b[3],12);
b[5]+=ROL(b[4],5);
b[4]+=ROL(b[5],22);
b[7]+=ROL(b[6],23);
b[6]+=b[7];
b[9]+=ROL(b[8],9);
b[8]+=ROL(b[9],10);
b[11]+=ROL(b[10],27);
b[10]+=ROL(b[11],20);
b[13]+=ROL(b[12],13);
b[12]+=ROL(b[13],30);
b[15]+=ROL(b[14],31);
b[14]+=ROL(b[15],8);
b[17]+=ROL(b[16],17);
b[16]+=ROL(b[17],18);
b[19]+=ROL(b[18],3);
b[18]+=ROL(b[19],28);
b[21]+=ROL(b[20],21);
b[20]+=ROL(b[21],6);
b[23]+=ROL(b[22],7);
b[22]+=ROL(b[23],16);
b[25]+=ROL(b[24],25);
```

```
b[24]+=ROL(b[25],26);
b[27]+=ROL(b[26],11);
b[26]+=ROL(b[27],4);
b[29]+=ROL(b[28],29);
b[28]+=ROL(b[29],14);
b[31]+=ROL(b[30],15);
b[30]+=ROL(b[31],24);
for(i=0;i<(WIDTH/2);i++){
  a[i]=b[2*i];
  a[i+(WIDTH/2)]=b[2*i+1];
}
a[1]+=ROL(a[0],1);
a[0]+=ROL(a[1],2);
a[3]+=ROL(a[2],23);
a[2]+=ROL(a[3],28);
a[5]+=ROL(a[4],13);
a[4]+=ROL(a[5],22);
a[7]+=ROL(a[6],3);
a[6]+=ROL(a[7],16);
a[9]+=ROL(a[8],25);
a[8]+=ROL(a[9],10);
a[11]+=ROL(a[10],15);
a[10]+=ROL(a[11],4);
a[13]+=ROL(a[12],5);
a[12]+=ROL(a[13],30);
a[15]+=ROL(a[14],27);
a[14]+=ROL(a[15],24);
a[17]+=ROL(a[16],17);
a[16]+=ROL(a[17],18);
a[19]+=ROL(a[18],7);
a[18]+=ROL(a[19],12);
a[21]+=ROL(a[20],29);
a[20]+=ROL(a[21],6);
a[23]+=ROL(a[22],19);
a[22]+=a[23];
a[25]+=ROL(a[24],9);
a[24]+=ROL(a[25],26),
a[27]+=ROL(a[26],31);
a[26]+=ROL(a[27],20);
a[29]+=ROL(a[28],21);
a[28]+=ROL(a[29],14);
a[31]+=ROL(a[30],11);
a[30]+=ROL(a[31],8);
```

```
}

#else

void ipht(unsigned long b[WIDTH]){
  unsigned long i,a[WIDTH];

  b[0]-=ROL(b[1],2);
  b[1]-=ROL(b[0],1);
  b[2]-=ROL(b[3],28);
  b[3]-=ROL(b[2],23);
  b[4]-=ROL(b[5],22);
  b[5]-=ROL(b[4],13);
  b[6]-=ROL(b[7],16);
  b[7]-=ROL(b[6],3);
  b[8]-=ROL(b[9],10);
  b[9]-=ROL(b[8],25);
  b[10]-=ROL(b[11],4);
  b[11]-=ROL(b[10],15);
  b[12]-=ROL(b[13],30);
  b[13]-=ROL(b[12],5);
  b[14]-=ROL(b[15],24);
  b[15]-=ROL(b[14],27);
  b[16]-=ROL(b[17],18);
  b[17]-=ROL(b[16],17);
  b[18]-=ROL(b[19],12);
  b[19]-=ROL(b[18],7);
  b[20]-=ROL(b[21],6);
  b[21]-=ROL(b[20],29);
  b[22]-=b[23];
  b[23]-=ROL(b[22],19);
  b[24]-=ROL(b[25],26);
  b[25]-=ROL(b[24],9);
  b[26]-=ROL(b[27],20);
  b[27]-=ROL(b[26],31);
  b[28]-=ROL(b[29],14);
  b[29]-=ROL(b[28],21);
  b[30]-=ROL(b[31],8);
  b[31]-=ROL(b[30],11);
  for(i=0;i<(WIDTH/2);i++){
    a[2*i]=b[i];
    a[2*i+1]=b[i+(WIDTH/2)];
  }
```

```
a[0]-=ROL(a[1],2);
a[1]-=ROL(a[0],1);
a[2]-=ROL(a[3],12);
a[3]-=ROL(a[2],19);
a[4]-=ROL(a[5],22);
a[5]-=ROL(a[4],5);
a[6]-=a[7];
a[7]-=ROL(a[6],23);
a[8]-=ROL(a[9],10),
a[9]-=ROL(a[8],9);
a[10]-=ROL(a[11],20);
a[11]-=ROL(a[10],27);
a[12]-=ROL(a[13],30);
a[13]-=ROL(a[12],13);
a[14]-=ROL(a[15],8);
a[15]-=ROL(a[14],31);
a[16]-=ROL(a[17],18);
a[17]-=ROL(a[16],17);
a[18]-=ROL(a[19],28);
a[19]-=ROL(a[18],3);
a[20]-=ROL(a[21],6);
a[21]-=ROL(a[20],21);
a[22]-=ROL(a[23],16);
a[23]-=ROL(a[22],7);
a[24]-=ROL(a[25],26);
a[25]-=ROL(a[24],25);
a[26]-=ROL(a[27],4);
a[27]-=ROL(a[26],11);
a[28]-=ROL(a[29],14);
a[29]-=ROL(a[28],29);
a[30]-=ROL(a[31],24);
a[31]-=ROL(a[30],15);
for(i=0;i<(WIDTH/2);i++){
  b[2*i]=a[i];
  b[2*i+1]=a[i+(WIDTH/2)];
}
b[0]-=ROL(b[1],2);
b[1]-=ROL(b[0],1);
b[2]-=ROL(b[3],24);
b[3]-=ROL(b[2],15);
b[4]-=ROL(b[5],14);
b[5]-=ROL(b[4],29);
b[6]-=ROL(b[7],4);
```

```
b[7]-=ROL(b[6],11);
b[8]-=ROL(b[9],26);
b[9]-=ROL(b[8],25);
b[10]-=ROL(b[11],16);
b[11]-=ROL(b[10],7);
b[12]-=ROL(b[13],6);
b[13]-=ROL(b[12],21);
b[14]-=ROL(b[15],28);
b[15]-=ROL(b[14],3);
b[16]-=ROL(b[17],18);
b[17]-=ROL(b[16],17);
b[18]-=ROL(b[19],8);
b[19]-=ROL(b[18],31);
b[20]-=ROL(b[21],30);
b[21]-=ROL(b[20],13);
b[22]-=ROL(b[23],20);
b[23]-=ROL(b[22],27);
b[24]-=ROL(b[25],10);
b[25]-=ROL(b[24],9);
b[26]-=b[27];
b[27]-=ROL(b[26],23);
b[28]-=ROL(b[29],22);
b[29]-=ROL(b[28],5);
b[30]-=ROL(b[31],12);
b[31]-=ROL(b[30],19);
for(i=0;i<(WIDTH/2);i++){
  a[2*i]=b[i];
  a[2*i+1]=b[i+(WIDTH/2)];
}
a[0]-=ROL(a[1],2);
a[1]-=ROL(a[0],1);
a[2]-=ROL(a[3],20);
a[3]-=ROL(a[2],11);
a[4]-=ROL(a[5],6);
a[5]-=ROL(a[4],21);
a[6]-=ROL(a[7],24);
a[7]-=ROL(a[6],31);
a[8]-=ROL(a[9],10),
a[9]-=ROL(a[8],9);
a[10]-=ROL(a[11],28);
a[11]-=ROL(a[10],19);
a[12]-=ROL(a[13],14);
a[13]-=ROL(a[12],29);
```

```
a[14]-=a[15];
a[15]-=ROL(a[14],7);
a[16]-=ROL(a[17],18);
a[17]-=ROL(a[16],17);
a[18]-=ROL(a[19],4);
a[19]-=ROL(a[18],27);
a[20]-=ROL(a[21],22);
a[21]-=ROL(a[20],5);
a[22]-=ROL(a[23],8);
a[23]-=ROL(a[22],15);
a[24]-=ROL(a[25],26);
a[25]-=ROL(a[24],25);
a[26]-=ROL(a[27],12);
a[27]-=ROL(a[26],3);
a[28]-=ROL(a[29],30);
a[29]-=ROL(a[28],13);
a[30]-=ROL(a[31],16);
a[31]-=ROL(a[30],23);
for(i=0;i<(WIDTH/2);i++){
  b[2*i]=a[i];
  b[2*i+1]=a[i+(WIDTH/2)];
}
b[0]-=ROL(b[1],2);
b[1]-=ROL(b[0],1);
b[2]-=ROL(b[3],16);
b[3]-=ROL(b[2],7);
b[4]-=ROL(b[5],30);
b[5]-=ROL(b[4],13);
b[6]-=ROL(b[7],12);
b[7]-=ROL(b[6],19);
b[8]-=ROL(b[9],26);
b[9]-=ROL(b[8],25);
b[10]-=ROL(b[11],8),
b[11]-=ROL(b[10],31);
b[12]-=ROL(b[13],22);
b[13]-=ROL(b[12],5);
b[14]-=ROL(b[15],4);
b[15]-=ROL(b[14],11);
b[16]-=ROL(b[17],18);
b[17]-=ROL(b[16],17);
b[18]-=b[19];
b[19]-=ROL(b[18],23);
b[20]-=ROL(b[21],14);
```

```
  b[21]-=ROL(b[20],29);
  b[22]-=ROL(b[23],28);
  b[23]-=ROL(b[22],3);
  b[24]-=ROL(b[25],10);
  b[25]-=ROL(b[24],9);
  b[26]-=ROL(b[27],24);
  b[27]-=ROL(b[26],15);
  b[28]-=ROL(b[29],6);
  b[29]-=ROL(b[28],21);
  b[30]-=ROL(b[31],20);
  b[31]-=ROL(b[30],27);
}

#endif

void f_function(unsigned long key[2*NUM_ROUNDS][WIDTH],\
unsigned long factors[WIDTH],unsigned long max,\
unsigned long left[WIDTH],unsigned long right[WIDTH]){

  unsigned  long i,j,help[WIDTH];
  unsigned long long n,o,carry1;

  for(i=0;i<WIDTH;i++) help[i]=left[i];

#ifdef TOP

#ifdef PRIMARY

  carry1=0;
  for(j=0;j<WIDTH;j++){
    help[j]^=key[max][j];
    help[j]=modmult(help[j],factors[j]);
    n=(unsigned long long) key[max+1][j];
    o=(unsigned long long) help[j];
    n+=o;
    n+=carry1;
    carry1=(n>>INT_LENGTH) & 1;
    help[j]=n & 0xFFFFFFFF;
    if((j & 7)==7) carry1=0;
   }

#else
```

```
  carry1=0;
  for(j=0;j<WIDTH;j++){
    n=(unsigned long long) help[j];
    o=(unsigned long long) key[max][j];
    n+=o;
    n+=carry1;
    carry1=(n>>INT_LENGTH) & 1;
    help[j]=n & 0xFFFFFFFF;
    help[j]=modmult(help[j],factors[j]);
    help[j]^=key[max+1][j];
    if((j & 7)==7) carry1=0;
  }

#endif

#ifdef PHT

    pht(help);

#else

   ipht(help);

#endif

#else

#ifdef PHT

    pht(help);

#else
    ipht(help);

#endif

#ifdef PRIMARY

  carry1=0;
  for(j=0;j<WIDTH;j++){
    help[j]^=key[max][j];
    help[j]=modmult(help[j],factors[j]);
    n=(unsigned long long) key[max+1][j];
```

```
    o=(unsigned long long) help[j];
    n+=o;
    n+=carry1;
    carry1=(n>>INT_LENGTH) & 1;
    help[j]=n & 0xFFFFFFFF;
    if((j & 7)==7) carry1=0;
   }

#else

  carry1=0;
  for(j=0;j<WIDTH;j++){
    n=(unsigned long long) help[j];
    o=(unsigned long long) key[max][j];
    n+=o;
    n+=carry1;
    carry1=(n>>INT_LENGTH) & 1;
    help[j]=n & 0xFFFFFFFF;
    help[j]=modmult(help[j],factors[j]);
    help[j]^=key[max+1][j];
    if((j & 7)==7) carry1=0;
  }

#endif

#endif

  for(i=0;i<WIDTH;i++) right[i]^=help[i];

}

void crypt(unsigned long key[2*NUM_ROUNDS][WIDTH],\
unsigned long factors[WIDTH],unsigned long left[WIDTH],\
unsigned long right[WIDTH]){

unsigned long i,help;

  for(i=0;i<(NUM_ROUNDS/2);i++){
    f_function(key,factors,(4*i),left,right);
    f_function(key,factors,(4*i+2),right,left);
  }
  for(i=0;i<WIDTH;i++){
    help=right[i];
```

```
    right[i]=left[i];
    left[i]=help;
  }
}



void encrypt(unsigned long userkey[][WIDTH],unsigned long long size,\
  unsigned long left[][WIDTH], unsigned long right[][WIDTH]){

  unsigned long key[2*NUM_ROUNDS][WIDTH];
  unsigned long factors[WIDTH];
  unsigned long i,j,help1[WIDTH],help2[WIDTH];
  unsigned long long m;

  key_schedule(userkey,key);
  encryption_factors(factors);
  for(i=0;i<WIDTH;i++){
    help1[i]=0;help2[i]=0;
  }

#if defined(FORWARD)

  for(i=0;i<(NUM_ROUNDS);i++){
    crypt(key,factors,help1,help2);
    for(j=0;j<WIDTH;j++){
      key[2*i][j]=help2[j];
      key[2*i+1][j]=help1[j];
    }
  }

#else

  for(i=(NUM_ROUNDS);i>0;i--){
    crypt(key,factors,help1,help2);
    for(j=0;j<WIDTH;j++){
      key[2*i-1][j]=help1[j];
      key[2*i-2][j]=help2[j];
    }
  }
  #endif

  for(m=0;m<size;m++){
```

```
    crypt(key,factors,&(left[m][0]),&(right[m][0]));
  }

}

void decrypt(unsigned long userkey[][WIDTH],unsigned long long size,\
  unsigned long left[][WIDTH], unsigned long right[][WIDTH]){

  unsigned long key[2*NUM_ROUNDS][WIDTH];
  unsigned long factors[WIDTH];
  unsigned long i,j,help1[WIDTH],help2[WIDTH];
  unsigned long long m;

  key_schedule(userkey,key);
  encryption_factors(factors);
  for(i=0;i<WIDTH;i++){
    help1[i]=0;help2[i]=0;
  }

#if defined(FORWARD)

  for(i=0;i<(NUM_ROUNDS);i++){
    crypt(key,factors,help1,help2);
    for(j=0;j<WIDTH;j++){
      key[2*i][j]=help2[j];
      key[2*i+1][j]=help1[j];
    }
  }

#else

  for(i=(NUM_ROUNDS);i>0;i--){
    crypt(key,factors,help1,help2);
    for(j=0;j<WIDTH;j++){
      key[2*i-1][j]=help1[j];
      key[2*i-2][j]=help2[j];
    }
  }
  #endif

  invert_keys(key);
  for(m=0;m<size;m++){
    crypt(key,factors,&(left[m][0]),&(right[m][0]));
```

```
  }

}



int main(){
  unsigned long i,j;
  unsigned long left[1][WIDTH],right[1][WIDTH];

#if defined(BIG_KEY)
  unsigned long userkey[8][WIDTH];
#else
  unsigned long userkey[4][WIDTH];
#endif

  for(i=0;i<WIDTH;i++){
   left[0][i]=i;
   right[0][i]=i;
  }
#if defined(BIG_KEY)
  for(i=0;i<WIDTH;i++){
    for(j=0;j<8;j++) userkey[j][i]=WIDTH*j+i;
  }
#else
  for(i=0;i<WIDTH;i++){
    for(j=0;j<4;j++) userkey[j][i]=WIDTH*j+i;
  }
#endif

  encrypt(userkey,1ULL,left,right);
  for(i=0;i<WIDTH;i++) printf("%lx    %lx\n",left[0][i], right[0][i]);
  scanf("%ld",&j);
  decrypt(userkey,1ULL,left,right);
  for(i=0;i<WIDTH;i++) printf("%lx        %lx\n",left[0][i],right[0][i]);
  scanf("%ld",&j);
return(0);
}
```

B Reference Implementation of 4096XKS-F

```
#include<stdio.h>

#define NUM_ROUNDS 32
#define INT_LENGTH 32
#define ROL(x,a) (((x)<<(a))|((x)>>(INT_LENGTH-(a))))
#define ROR(x,a) (((x)<<(INT_LENGTH-(a))|((x)>>(a))))
#define WIDTH 64
#define ROTROUND 455

#define FORWARD
#define BIG_KEY
#define FACTORS
#define PRIMARY
#define PHT
#define TOP

#ifdef FACTORS

void encryption_factors(unsigned long e_factors[WIDTH]){
  unsigned long i;

  e_factors[0]=0x025F1CDB;
  for(i=0;i<(WIDTH/4);i++){
    if(i!=0) e_factors[2*i]=ROL(e_factors[0],2*i);
    e_factors[2*i+1]=\
    ROL(e_factors[0],((WIDTH/2)+7-2*i)%(WIDTH/2));
  }
  e_factors[WIDTH/2]=229459604;
  for(i=0;i<(WIDTH/4);i++){
    if(i!=0) e_factors[WIDTH/2+2*i]=ROL(e_factors[WIDTH/2],2*i);
    e_factors[WIDTH/2+2*i+1]=\
    ROL(e_factors[WIDTH/2],((WIDTH/2)+17-2*i)%(WIDTH/2));
  }
}

#else

void encryption_factors(unsigned long d_factors[WIDTH]){
  unsigned long i;
```

```
  d_factors[0]=229459604;
  for(i=0;i<(WIDTH/4);i++){
    if(i!=0) d_factors[2*i]=ROR(d_factors[0],2*i);
    d_factors[2*i+1]=\
    ROR(d_factors[0],((WIDTH/2)+7-2*i)%(WIDTH/2));
  }
  d_factors[WIDTH/2]=0x025F1CDB;
  for(i=0;i<(WIDTH/4);i++){
    if(i!=0) d_factors[WIDTH/2+2*i]=ROR(d_factors[WIDTH/2],2*i);
    d_factors[WIDTH/2+2*i+1]=\
    ROR(d_factors[WIDTH/2],((WIDTH/2)+17-2*i)%(WIDTH/2));
  }
}

#endif

unsigned long modmult(unsigned long factor1,unsigned long factor2){
  unsigned long long f1,f2,ergebnis,k;

  f1=(unsigned long long) factor1;
  f2=(unsigned long long) factor2;
  ergebnis=f1*f2;
  k=(ergebnis>>INT_LENGTH);
  ergebnis&=0xFFFFFFFF;
  ergebnis+=k;
  ergebnis+=(ergebnis>>INT_LENGTH) & 1;
  return(ergebnis & 0xFFFFFFFF);
}

void invert_keys(unsigned long keys[2*NUM_ROUNDS][WIDTH]){
  unsigned long i,j,intermediate[2*NUM_ROUNDS][WIDTH];

  for(i=0;i<(2*NUM_ROUNDS);i++){
     for(j=0;j<WIDTH;j++){
       intermediate[i][j]=keys[i][j];
     }
  }
  for(i=0;i<NUM_ROUNDS;i++){
    for(j=0;j<WIDTH;j++){
      keys[(2*NUM_ROUNDS-2)-2*i][j]=intermediate[2*i][j];
      keys[(2*NUM_ROUNDS-1)-2*i][j]=intermediate[2*i+1][j];
    }
  }
```

```
}



#ifdef BIG_KEY

void key_schedule(unsigned long user_key[8][WIDTH],\
  unsigned long key[2*NUM_ROUNDS][WIDTH]){
  unsigned long i,j;

  for(i=0;i<8;i++){
    for(j=0;j<WIDTH;j++){
      key[i][j]=user_key[i][j];
    }
  }
  for(i=1;i<(NUM_ROUNDS/4);i++){
    for(j=0;j<WIDTH;j++) {
      key[8*i+7][j]=(key[8*(i-1)+((j+((WIDTH*INT_LENGTH-ROTROUND)\
      /INT_LENGTH))/WIDTH*7)]\
      [(j+((WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH))\
      %WIDTH]<<(ROTROUND%INT_LENGTH))|\
      (key[8*(i-1)+((j+(WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH+1)/WIDTH)*7]\
      [(j+(WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH+1)%WIDTH]>>\
      (INT_LENGTH-ROTROUND%INT_LENGTH));

      key[8*i+6][j]=(key[8*(i-1)+7-(j+((WIDTH*INT_LENGTH-ROTROUND)\
      /INT_LENGTH))/WIDTH]\
      [(j+((WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH))\
      %WIDTH]<<(ROTROUND%INT_LENGTH))\
      |(key[8*(i-1)+7-(j+(WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH+1)/WIDTH]\
      [(j+(WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH+1)%WIDTH]>>\
      (INT_LENGTH-ROTROUND%INT_LENGTH));

      key[8*i+5][j]=(key[8*(i-1)+6-(j+((WIDTH*INT_LENGTH-ROTROUND)\
      /INT_LENGTH))/WIDTH]\
      [(j+((WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH))\
      %WIDTH]<<(ROTROUND%INT_LENGTH))\
      |(key[8*(i-1)+6-(j+(WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH+1)/WIDTH]\
      [(j+(WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH+1)%WIDTH]>>\
      (INT_LENGTH-ROTROUND%INT_LENGTH));

      key[8*i+4][j]=(key[8*(i-1)+5-(j+((WIDTH*INT_LENGTH-ROTROUND)\
      /INT_LENGTH))/WIDTH]\
```

```
        [(j+((WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH))\
        %WIDTH]<<(ROTROUND%INT_LENGTH))\
        |(key[8*(i-1)+5-(j+(WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH+1)/WIDTH]\
        [(j+(WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH+1)%WIDTH]>>\
        (INT_LENGTH-ROTROUND%INT_LENGTH));

        key[8*i+3][j]=(key[8*(i-1)+4-(j+((WIDTH*INT_LENGTH-ROTROUND)\
        /INT_LENGTH))/WIDTH]\
        [(j+((WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH))\
        %WIDTH]<<(ROTROUND%INT_LENGTH))\
        |(key[8*(i-1)+4-(j+(WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH+1)/WIDTH]\
        [(j+(WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH+1)%WIDTH]>>\
        (INT_LENGTH-ROTROUND%INT_LENGTH));


        key[8*i+2][j]=(key[8*(i-1)+3-(j+((WIDTH*INT_LENGTH-ROTROUND)\
        /INT_LENGTH))/WIDTH]\
        [(j+((WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH))\
        %WIDTH]<<(ROTROUND%INT_LENGTH))\
        |(key[8*(i-1)+3-(j+(WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH+1)/WIDTH]\
        [(j+(WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH+1)%WIDTH]>>\
        (INT_LENGTH-ROTROUND%INT_LENGTH));

        key[8*i+1][j]=(key[8*(i-1)+2-(j+((WIDTH*INT_LENGTH-ROTROUND)\
        /INT_LENGTH))/WIDTH]\
        [(j+((WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH))\
        %WIDTH]<<(ROTROUND%INT_LENGTH))\
        |(key[8*(i-1)+2-(j+(WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH+1)/WIDTH]\
        [(j+(WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH+1)%WIDTH]>>\
        (INT_LENGTH-ROTROUND%INT_LENGTH));


        key[8*i][j]=(key[8*(i-1)+1-(j+((WIDTH*INT_LENGTH-ROTROUND)\
        /INT_LENGTH))/WIDTH]\
        [(j+((WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH))\
        %WIDTH]<<(ROTROUND%INT_LENGTH))\
        |(key[8*(i-1)+1-(j+(WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH+1)/WIDTH]\
        [(j+(WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH+1)%WIDTH]>>\
        (INT_LENGTH-ROTROUND%INT_LENGTH));

    }
  }
}
```

```
#else

void key_schedule(unsigned long user_key[4][WIDTH],\
  unsigned long key[2*NUM_ROUNDS][WIDTH]){
  unsigned long i,j;

  for(i=0;i<4;i++){
    for(j=0;j<WIDTH;j++){
      key[i][j]=user_key[i][j];
    }
  }
  for(i=1;i<(NUM_ROUNDS/2);i++){
    for(j=0;j<WIDTH;j++) {
      key[4*i+3][j]=(key[4*(i-1)+((j+((WIDTH*INT_LENGTH-ROTROUND)\
      /INT_LENGTH))/WIDTH*3)]\
      [(j+((WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH))\
      %WIDTH]<<(ROTROUND%INT_LENGTH))|\
      (key[4*(i-1)+((j+(WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH+1)/WIDTH)*3]\
      [(j+(WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH+1)%WIDTH]>>\
      (INT_LENGTH-ROTROUND%INT_LENGTH));

      key[4*i+2][j]=(key[4*(i-1)+3-(j+((WIDTH*INT_LENGTH-ROTROUND)\
      /INT_LENGTH))/WIDTH]\
      [(j+((WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH))\
      %WIDTH]<<(ROTROUND%INT_LENGTH))\
      |(key[4*(i-1)+3-(j+(WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH+1)/WIDTH]\
      [(j+(WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH+1)%WIDTH]>>\
      (INT_LENGTH-ROTROUND%INT_LENGTH));

      key[4*i+1][j]=(key[4*(i-1)+2-(j+((WIDTH*INT_LENGTH-ROTROUND)\
      /INT_LENGTH))/WIDTH]\
      [(j+((WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH))\
      %WIDTH]<<(ROTROUND%INT_LENGTH))\
      |(key[4*(i-1)+2-(j+(WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH+1)/WIDTH]\
      [(j+(WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH+1)%WIDTH]>>\
      (INT_LENGTH-ROTROUND%INT_LENGTH));


      key[4*i][j]=(key[4*(i-1)+1-(j+((WIDTH*INT_LENGTH-ROTROUND)\
      /INT_LENGTH))/WIDTH]\
      [(j+((WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH))\
```

```
      %WIDTH]<<(ROTROUND%INT_LENGTH))\
      |(key[4*(i-1)+1-(j+(WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH+1)/WIDTH]\
      [(j+(WIDTH*INT_LENGTH-ROTROUND)/INT_LENGTH+1)%WIDTH]>>\
      (INT_LENGTH-ROTROUND%INT_LENGTH));

    }
  }
}

#endif

#ifdef PHT

void pht(unsigned long a[WIDTH]){
  unsigned long i,b[WIDTH];

  a[1]+=ROL(a[0],1);
  a[0]+=ROL(a[1],2);
  a[3]+=ROL(a[2],7);
  a[2]+=ROL(a[3],16);
  a[5]+=ROL(a[4],13),
  a[4]+=ROL(a[5],30);
  a[7]+=ROL(a[6],19);
  a[6]+=ROL(a[7],12);
  a[9]+=ROL(a[8],25);
  a[8]+=ROL(a[9],26);
  a[11]+=ROL(a[10],31);
  a[10]+=ROL(a[11],8);
  a[13]+=ROL(a[12],5);
  a[12]+=ROL(a[13],22);
  a[15]+=ROL(a[14],11);
  a[14]+=ROL(a[15],4);
  a[17]+=ROL(a[16],17);
  a[16]+=ROL(a[17],18),
  a[19]+=ROL(a[18],23);
  a[18]+=a[19];
  a[21]+=ROL(a[20],29);
  a[20]+=ROL(a[21],14);
  a[23]+=ROL(a[22],3);
  a[22]+=ROL(a[23],28);
  a[25]+=ROL(a[24],9);
  a[24]+=ROL(a[25],10);
  a[27]+=ROL(a[26],15);
```

```
a[26]+=ROL(a[27],24);
a[29]+=ROL(a[28],21);
a[28]+=ROL(a[29],6);
a[31]+=ROL(a[30],27);
a[30]+=ROL(a[31],20);

a[33]+=ROL(a[32],1);
a[32]+=ROL(a[33],2);
a[35]+=ROL(a[34],7);
a[34]+=ROL(a[35],16);
a[37]+=ROL(a[36],13),
a[36]+=ROL(a[37],30);
a[39]+=ROL(a[38],19);
a[38]+=ROL(a[39],12);
a[41]+=ROL(a[40],25);
a[40]+=ROL(a[41],26);
a[43]+=ROL(a[42],31);
a[42]+=ROL(a[43],8);
a[45]+=ROL(a[44],5);
a[44]+=ROL(a[45],22);
a[47]+=ROL(a[46],11);
a[46]+=ROL(a[47],4);
a[49]+=ROL(a[48],17);
a[48]+=ROL(a[49],18),
a[51]+=ROL(a[50],23);
a[50]+=a[51];
a[53]+=ROL(a[52],29);
a[52]+=ROL(a[53],14);
a[55]+=ROL(a[54],3);
a[54]+=ROL(a[55],28);
a[57]+=ROL(a[56],9);
a[56]+=ROL(a[57],10);
a[59]+=ROL(a[58],15);
a[58]+=ROL(a[59],24);
a[61]+=ROL(a[60],21);
a[60]+=ROL(a[61],6);
a[63]+=ROL(a[62],27);
a[62]+=ROL(a[63],20);

for(i=0;i<(WIDTH/2);i++){
  b[i]=a[2*i];
  b[i+(WIDTH/2)]=a[2*i+1];
}
```

```
b[1]+=ROL(b[0],1);
b[0]+=ROL(b[1],2);
b[3]+=ROL(b[2],11);
b[2]+=ROL(b[3],20);
b[5]+=ROL(b[4],21);
b[4]+=ROL(b[5],6);
b[7]+=ROL(b[6],31);
b[6]+=ROL(b[7],24);
b[9]+=ROL(b[8],9);
b[8]+=ROL(b[9],10);
b[11]+=ROL(b[10],19);
b[10]+=ROL(b[11],28);
b[13]+=ROL(b[12],29);
b[12]+=ROL(b[13],14);
b[15]+=ROL(b[14],7);
b[14]+=b[15];
b[17]+=ROL(b[16],17);
b[16]+=ROL(b[17],18);
b[19]+=ROL(b[18],27);
b[18]+=ROL(b[19],4);
b[21]+=ROL(b[20],5);
b[20]+=ROL(b[21],22);
b[23]+=ROL(b[22],15);
b[22]+=ROL(b[23],8);
b[25]+=ROL(b[24],25);
b[24]+=ROL(b[25],26);
b[27]+=ROL(b[26],3);
b[26]+=ROL(b[27],12);
b[29]+=ROL(b[28],13);
b[28]+=ROL(b[29],30);
b[31]+=ROL(b[30],23);
b[30]+=ROL(b[31],16);

b[33]+=ROL(b[32],1);
b[32]+=ROL(b[33],2);
b[35]+=ROL(b[34],11);
b[34]+=ROL(b[35],20);
b[37]+=ROL(b[36],21);
b[36]+=ROL(b[37],6);
b[39]+=ROL(b[38],31);
b[38]+=ROL(b[39],24);
b[41]+=ROL(b[40],9);
```

```
b[40]+=ROL(b[41],10);
b[43]+=ROL(b[42],19);
b[42]+=ROL(b[43],28);
b[45]+=ROL(b[44],29);
b[44]+=ROL(b[45],14);
b[47]+=ROL(b[46],7);
b[46]+=b[47];
b[49]+=ROL(b[48],17);
b[48]+=ROL(b[49],18);
b[51]+=ROL(b[50],27);
b[50]+=ROL(b[51],4);
b[53]+=ROL(b[52],5);
b[52]+=ROL(b[53],22);
b[55]+=ROL(b[54],15);
b[54]+=ROL(b[55],8);
b[57]+=ROL(b[56],25);
b[56]+=ROL(b[57],26);
b[59]+=ROL(b[58],3);
b[58]+=ROL(b[59],12);
b[61]+=ROL(b[60],13);
b[60]+=ROL(b[61],30);
b[63]+=ROL(b[62],23);
b[62]+=ROL(b[63],16);

for(i=0;i<(WIDTH/2);i++){
  a[i]=b[2*i];
  a[i+(WIDTH/2)]=b[2*i+1];
}

a[1]+=ROL(a[0],1);
a[0]+=ROL(a[1],2);
a[3]+=ROL(a[2],15);
a[2]+=ROL(a[3],24);
a[5]+=ROL(a[4],29);
a[4]+=ROL(a[5],14);
a[7]+=ROL(a[6],11);
a[6]+=ROL(a[7],4);
a[9]+=ROL(a[8],25);
a[8]+=ROL(a[9],26);
a[11]+=ROL(a[10],7);
a[10]+=ROL(a[11],16);
a[13]+=ROL(a[12],21);
a[12]+=ROL(a[13],6);
```

```
a[15]+=ROL(a[14],3);
a[14]+=ROL(a[15],28);
a[17]+=ROL(a[16],17);
a[16]+=ROL(a[17],18);
a[19]+=ROL(a[18],31);
a[18]+=ROL(a[19],8);
a[21]+=ROL(a[20],13);
a[20]+=ROL(a[21],30);
a[23]+=ROL(a[22],27);
a[22]+=ROL(a[23],20);
a[25]+=ROL(a[24],9);
a[24]+=ROL(a[25],10),
a[27]+=ROL(a[26],23);
a[26]+=a[27];
a[29]+=ROL(a[28],5);
a[28]+=ROL(a[29],22);
a[31]+=ROL(a[30],19);
a[30]+=ROL(a[31],12);

a[33]+=ROL(a[32],1);
a[32]+=ROL(a[33],2);
a[35]+=ROL(a[34],15);
a[34]+=ROL(a[35],24);
a[37]+=ROL(a[36],29);
a[36]+=ROL(a[37],14);
a[39]+=ROL(a[38],11);
a[38]+=ROL(a[39],4);
a[41]+=ROL(a[40],25);
a[40]+=ROL(a[41],26);
a[43]+=ROL(a[42],7);
a[42]+=ROL(a[43],16);
a[45]+=ROL(a[44],21);
a[44]+=ROL(a[45],6);
a[47]+=ROL(a[46],3);
a[46]+=ROL(a[47],28);
a[49]+=ROL(a[48],17);
a[48]+=ROL(a[49],18);
a[51]+=ROL(a[50],31);
a[50]+=ROL(a[51],8);
a[53]+=ROL(a[52],13);
a[52]+=ROL(a[53],30);
a[55]+=ROL(a[54],27);
a[54]+=ROL(a[55],20);
```

```
a[57]+=ROL(a[56],9);
a[56]+=ROL(a[57],10),
a[59]+=ROL(a[58],23);
a[58]+=a[59];
a[61]+=ROL(a[60],5);
a[60]+=ROL(a[61],22);
a[63]+=ROL(a[62],19);
a[62]+=ROL(a[63],12);

for(i=0;i<(WIDTH/2);i++){
  b[i]=a[2*i];
  b[i+(WIDTH/2)]=a[2*i+1];
}

b[1]+=ROL(b[0],1);
b[0]+=ROL(b[1],2);
b[3]+=ROL(b[2],19);
b[2]+=ROL(b[3],12);
b[5]+=ROL(b[4],5);
b[4]+=ROL(b[5],22);
b[7]+=ROL(b[6],23);
b[6]+=b[7];
b[9]+=ROL(b[8],9);
b[8]+=ROL(b[9],10);
b[11]+=ROL(b[10],27);
b[10]+=ROL(b[11],20);
b[13]+=ROL(b[12],13);
b[12]+=ROL(b[13],30);
b[15]+=ROL(b[14],31);
b[14]+=ROL(b[15],8);
b[17]+=ROL(b[16],17);
b[16]+=ROL(b[17],18);
b[19]+=ROL(b[18],3);
b[18]+=ROL(b[19],28);
b[21]+=ROL(b[20],21);
b[20]+=ROL(b[21],6);
b[23]+=ROL(b[22],7);
b[22]+=ROL(b[23],16);
b[25]+=ROL(b[24],25);
b[24]+=ROL(b[25],26);
b[27]+=ROL(b[26],11);
b[26]+=ROL(b[27],4);
b[29]+=ROL(b[28],29);
```

```
b[28]+=ROL(b[29],14);
b[31]+=ROL(b[30],15);
b[30]+=ROL(b[31],24);

b[33]+=ROL(b[32],1);
b[32]+=ROL(b[33],2);
b[35]+=ROL(b[34],19);
b[34]+=ROL(b[35],12);
b[37]+=ROL(b[36],5);
b[36]+=ROL(b[37],22);
b[39]+=ROL(b[38],23);
b[38]+=b[39];
b[41]+=ROL(b[40],9);
b[40]+=ROL(b[41],10);
b[43]+=ROL(b[42],27);
b[42]+=ROL(b[43],20);
b[45]+=ROL(b[44],13);
b[44]+=ROL(b[45],30);
b[47]+=ROL(b[46],31);
b[46]+=ROL(b[47],8);
b[49]+=ROL(b[48],17);
b[48]+=ROL(b[49],18);
b[51]+=ROL(b[50],3);
b[50]+=ROL(b[51],28);
b[53]+=ROL(b[52],21);
b[52]+=ROL(b[53],6);
b[55]+=ROL(b[54],7);
b[54]+=ROL(b[55],16);
b[57]+=ROL(b[56],25);
b[56]+=ROL(b[57],26);
b[59]+=ROL(b[58],11);
b[58]+=ROL(b[59],4);
b[61]+=ROL(b[60],29);
b[60]+=ROL(b[61],14);
b[63]+=ROL(b[62],15);
b[62]+=ROL(b[63],24);


for(i=0;i<(WIDTH/2);i++){
  a[i]=b[2*i];
  a[i+(WIDTH/2)]=b[2*i+1];
}
```

```
a[1]+=ROL(a[0],1);
a[0]+=ROL(a[1],2);
a[3]+=ROL(a[2],23);
a[2]+=ROL(a[3],28);
a[5]+=ROL(a[4],13);
a[4]+=ROL(a[5],22);
a[7]+=ROL(a[6],3);
a[6]+=ROL(a[7],16);
a[9]+=ROL(a[8],25);
a[8]+=ROL(a[9],10);
a[11]+=ROL(a[10],15);
a[10]+=ROL(a[11],4);
a[13]+=ROL(a[12],5);
a[12]+=ROL(a[13],30);
a[15]+=ROL(a[14],27);
a[14]+=ROL(a[15],24);
a[17]+=ROL(a[16],17);
a[16]+=ROL(a[17],18);
a[19]+=ROL(a[18],7);
a[18]+=ROL(a[19],12);
a[21]+=ROL(a[20],29);
a[20]+=ROL(a[21],6);
a[23]+=ROL(a[22],19);
a[22]+=a[23];
a[25]+=ROL(a[24],9);
a[24]+=ROL(a[25],26),
a[27]+=ROL(a[26],31);
a[26]+=ROL(a[27],20);
a[29]+=ROL(a[28],21);
a[28]+=ROL(a[29],14);
a[31]+=ROL(a[30],11);
a[30]+=ROL(a[31],8);

a[33]+=ROL(a[32],1);
a[32]+=ROL(a[33],2);
a[35]+=ROL(a[34],23);
a[34]+=ROL(a[35],28);
a[37]+=ROL(a[36],13);
a[36]+=ROL(a[37],22);
a[39]+=ROL(a[38],3);
a[38]+=ROL(a[39],16);
a[41]+=ROL(a[40],25);
a[40]+=ROL(a[41],10);
```

```
a[43]+=ROL(a[42],15);
a[42]+=ROL(a[43],4);
a[45]+=ROL(a[44],5);
a[44]+=ROL(a[45],30);
a[47]+=ROL(a[46],27);
a[46]+=ROL(a[47],24);
a[49]+=ROL(a[48],17);
a[48]+=ROL(a[49],18);
a[51]+=ROL(a[50],7);
a[50]+=ROL(a[51],12);
a[53]+=ROL(a[52],29);
a[52]+=ROL(a[53],6);
a[55]+=ROL(a[54],19);
a[54]+=a[55];
a[57]+=ROL(a[56],9);
a[56]+=ROL(a[57],26),
a[59]+=ROL(a[58],31);
a[58]+=ROL(a[59],20);
a[61]+=ROL(a[60],21);
a[60]+=ROL(a[61],14);
a[63]+=ROL(a[62],11);
a[62]+=ROL(a[63],8);

for(i=0;i<(WIDTH/2);i++){
  b[i]=a[2*i];
  b[i+(WIDTH/2)]=a[2*i+1];
}

b[1]+=ROL(b[0],1);
b[0]+=ROL(b[1],2);
b[3]+=ROL(b[2],19);
b[2]+=ROL(b[3],12);
b[5]+=ROL(b[4],5);
b[4]+=ROL(b[5],22);
b[7]+=ROL(b[6],23);
b[6]+=b[7];
b[9]+=ROL(b[8],9);
b[8]+=ROL(b[9],10);
b[11]+=ROL(b[10],27);
b[10]+=ROL(b[11],20);
b[13]+=ROL(b[12],13);
b[12]+=ROL(b[13],30);
b[15]+=ROL(b[14],31);
```

```
b[14]+=ROL(b[15],8);
b[17]+=ROL(b[16],17);
b[16]+=ROL(b[17],18);
b[19]+=ROL(b[18],3);
b[18]+=ROL(b[19],28);
b[21]+=ROL(b[20],21);
b[20]+=ROL(b[21],6);
b[23]+=ROL(b[22],7);
b[22]+=ROL(b[23],16);
b[25]+=ROL(b[24],25);
b[24]+=ROL(b[25],26);
b[27]+=ROL(b[26],11);
b[26]+=ROL(b[27],4);
b[29]+=ROL(b[28],29);
b[28]+=ROL(b[29],14);
b[31]+=ROL(b[30],15);
b[30]+=ROL(b[31],24);

b[33]+=ROL(b[32],1);
b[32]+=ROL(b[33],2);
b[35]+=ROL(b[34],19);
b[34]+=ROL(b[35],12);
b[37]+=ROL(b[36],5);
b[36]+=ROL(b[37],22);
b[39]+=ROL(b[38],23);
b[38]+=b[39];
b[41]+=ROL(b[40],9);
b[40]+=ROL(b[41],10);
b[43]+=ROL(b[42],27);
b[42]+=ROL(b[43],20);
b[45]+=ROL(b[44],13);
b[44]+=ROL(b[45],30);
b[47]+=ROL(b[46],31);
b[46]+=ROL(b[47],8);
b[49]+=ROL(b[48],17);
b[48]+=ROL(b[49],18);
b[51]+=ROL(b[50],3);
b[50]+=ROL(b[51],28);
b[53]+=ROL(b[52],21);
b[52]+=ROL(b[53],6);
b[55]+=ROL(b[54],7);
b[54]+=ROL(b[55],16);
b[57]+=ROL(b[56],25);
```

```
  b[56]+=ROL(b[57],26);
  b[59]+=ROL(b[58],11);
  b[58]+=ROL(b[59],4);
  b[61]+=ROL(b[60],29);
  b[60]+=ROL(b[61],14);
  b[63]+=ROL(b[62],15);
  b[62]+=ROL(b[63],24);

  for(i=0;i<WIDTH;i++) a[i]=b[i];

}

#else

void ipht(unsigned long a[WIDTH]){
  unsigned long i,b[WIDTH];

  a[0]-=ROL(a[1],2);
  a[1]-=ROL(a[0],1);
  a[2]-=ROL(a[3],12);
  a[3]-=ROL(a[2],19);
  a[4]-=ROL(a[5],22);
  a[5]-=ROL(a[4],5);
  a[6]-=a[7];
  a[7]-=ROL(a[6],23);
  a[8]-=ROL(a[9],10),
  a[9]-=ROL(a[8],9);
  a[10]-=ROL(a[11],20);
  a[11]-=ROL(a[10],27);
  a[12]-=ROL(a[13],30);
  a[13]-=ROL(a[12],13);
  a[14]-=ROL(a[15],8);
  a[15]-=ROL(a[14],31);
  a[16]-=ROL(a[17],18);
  a[17]-=ROL(a[16],17);
  a[18]-=ROL(a[19],28);
  a[19]-=ROL(a[18],3);
  a[20]-=ROL(a[21],6);
  a[21]-=ROL(a[20],21);
  a[22]-=ROL(a[23],16);
  a[23]-=ROL(a[22],7);
  a[24]-=ROL(a[25],26);
  a[25]-=ROL(a[24],25);
```

```
a[26]-=ROL(a[27],4);
a[27]-=ROL(a[26],11);
a[28]-=ROL(a[29],14);
a[29]-=ROL(a[28],29);
a[30]-=ROL(a[31],24);
a[31]-=ROL(a[30],15);

a[32]-=ROL(a[33],2);
a[33]-=ROL(a[32],1);
a[34]-=ROL(a[35],12);
a[35]-=ROL(a[34],19);
a[36]-=ROL(a[37],22);
a[37]-=ROL(a[36],5);
a[38]-=a[39];
a[39]-=ROL(a[38],23);
a[40]-=ROL(a[41],10),
a[41]-=ROL(a[40],9);
a[42]-=ROL(a[43],20);
a[43]-=ROL(a[42],27);
a[44]-=ROL(a[45],30);
a[45]-=ROL(a[44],13);
a[46]-=ROL(a[47],8);
a[47]-=ROL(a[46],31);
a[48]-=ROL(a[49],18);
a[49]-=ROL(a[48],17);
a[50]-=ROL(a[51],28);
a[51]-=ROL(a[50],3);
a[52]-=ROL(a[53],6);
a[53]-=ROL(a[52],21);
a[54]-=ROL(a[55],16);
a[55]-=ROL(a[54],7);
a[56]-=ROL(a[57],26);
a[57]-=ROL(a[56],25);
a[58]-=ROL(a[59],4);
a[59]-=ROL(a[58],11);
a[60]-=ROL(a[61],14);
a[61]-=ROL(a[60],29);
a[62]-=ROL(a[63],24);
a[63]-=ROL(a[62],15);


for(i=0;i<(WIDTH/2);i++){
  b[2*i]=a[i];
```

```
  b[2*i+1]=a[i+(WIDTH/2)];
}


b[0]-=ROL(b[1],2);
b[1]-=ROL(b[0],1);
b[2]-=ROL(b[3],28);
b[3]-=ROL(b[2],23);
b[4]-=ROL(b[5],22);
b[5]-=ROL(b[4],13);
b[6]-=ROL(b[7],16);
b[7]-=ROL(b[6],3);
b[8]-=ROL(b[9],10);
b[9]-=ROL(b[8],25);
b[10]-=ROL(b[11],4);
b[11]-=ROL(b[10],15);
b[12]-=ROL(b[13],30);
b[13]-=ROL(b[12],5);
b[14]-=ROL(b[15],24);
b[15]-=ROL(b[14],27);
b[16]-=ROL(b[17],18);
b[17]-=ROL(b[16],17);
b[18]-=ROL(b[19],12);
b[19]-=ROL(b[18],7);
b[20]-=ROL(b[21],6);
b[21]-=ROL(b[20],29);
b[22]-=b[23];
b[23]-=ROL(b[22],19);
b[24]-=ROL(b[25],26);
b[25]-=ROL(b[24],9);
b[26]-=ROL(b[27],20);
b[27]-=ROL(b[26],31);
b[28]-=ROL(b[29],14);
b[29]-=ROL(b[28],21);
b[30]-=ROL(b[31],8);
b[31]-=ROL(b[30],11);

b[32]-=ROL(b[33],2);
b[33]-=ROL(b[32],1);
b[34]-=ROL(b[35],28);
b[35]-=ROL(b[34],23);
b[36]-=ROL(b[37],22);
b[37]-=ROL(b[36],13);
```

```
b[38]-=ROL(b[39],16);
b[39]-=ROL(b[38],3);
b[40]-=ROL(b[41],10);
b[41]-=ROL(b[40],25);
b[42]-=ROL(b[43],4);
b[43]-=ROL(b[42],15);
b[44]-=ROL(b[45],30);
b[45]-=ROL(b[44],5);
b[46]-=ROL(b[47],24);
b[47]-=ROL(b[46],27);
b[48]-=ROL(b[49],18);
b[49]-=ROL(b[48],17);
b[50]-=ROL(b[51],12);
b[51]-=ROL(b[50],7);
b[52]-=ROL(b[53],6);
b[53]-=ROL(b[52],29);
b[54]-=b[55];
b[55]-=ROL(b[54],19);
b[56]-=ROL(b[57],26);
b[57]-=ROL(b[56],9);
b[58]-=ROL(b[59],20);
b[59]-=ROL(b[58],31);
b[60]-=ROL(b[61],14);
b[61]-=ROL(b[60],21);
b[62]-=ROL(b[63],8);
b[63]-=ROL(b[62],11);


for(i=0;i<(WIDTH/2);i++){
  a[2*i]=b[i];
  a[2*i+1]=b[i+(WIDTH/2)];
}

a[0]-=ROL(a[1],2);
a[1]-=ROL(a[0],1);
a[2]-=ROL(a[3],12);
a[3]-=ROL(a[2],19);
a[4]-=ROL(a[5],22);
a[5]-=ROL(a[4],5);
a[6]-=a[7];
a[7]-=ROL(a[6],23);
a[8]-=ROL(a[9],10),
a[9]-=ROL(a[8],9);
```

```
a[10]-=ROL(a[11],20);
a[11]-=ROL(a[10],27);
a[12]-=ROL(a[13],30);
a[13]-=ROL(a[12],13);
a[14]-=ROL(a[15],8);
a[15]-=ROL(a[14],31);
a[16]-=ROL(a[17],18);
a[17]-=ROL(a[16],17);
a[18]-=ROL(a[19],28);
a[19]-=ROL(a[18],3);
a[20]-=ROL(a[21],6);
a[21]-=ROL(a[20],21);
a[22]-=ROL(a[23],16);
a[23]-=ROL(a[22],7);
a[24]-=ROL(a[25],26);
a[25]-=ROL(a[24],25);
a[26]-=ROL(a[27],4);
a[27]-=ROL(a[26],11);
a[28]-=ROL(a[29],14);
a[29]-=ROL(a[28],29);
a[30]-=ROL(a[31],24);
a[31]-=ROL(a[30],15);

a[32]-=ROL(a[33],2);
a[33]-=ROL(a[32],1);
a[34]-=ROL(a[35],12);
a[35]-=ROL(a[34],19);
a[36]-=ROL(a[37],22);
a[37]-=ROL(a[36],5);
a[38]-=a[39];
a[39]-=ROL(a[38],23);
a[40]-=ROL(a[41],10),
a[41]-=ROL(a[40],9);
a[42]-=ROL(a[43],20);
a[43]-=ROL(a[42],27);
a[44]-=ROL(a[45],30);
a[45]-=ROL(a[44],13);
a[46]-=ROL(a[47],8);
a[47]-=ROL(a[46],31);
a[48]-=ROL(a[49],18);
a[49]-=ROL(a[48],17);
a[50]-=ROL(a[51],28);
a[51]-=ROL(a[50],3);
```

```
a[52]-=ROL(a[53],6);
a[53]-=ROL(a[52],21);
a[54]-=ROL(a[55],16);
a[55]-=ROL(a[54],7);
a[56]-=ROL(a[57],26);
a[57]-=ROL(a[56],25);
a[58]-=ROL(a[59],4);
a[59]-=ROL(a[58],11);
a[60]-=ROL(a[61],14);
a[61]-=ROL(a[60],29);
a[62]-=ROL(a[63],24);
a[63]-=ROL(a[62],15);


for(i=0;i<(WIDTH/2);i++){
  b[2*i]=a[i];
  b[2*i+1]=a[i+(WIDTH/2)];
}

b[0]-=ROL(b[1],2);
b[1]-=ROL(b[0],1);
b[2]-=ROL(b[3],24);
b[3]-=ROL(b[2],15);
b[4]-=ROL(b[5],14);
b[5]-=ROL(b[4],29);
b[6]-=ROL(b[7],4);
b[7]-=ROL(b[6],11);
b[8]-=ROL(b[9],26);
b[9]-=ROL(b[8],25);
b[10]-=ROL(b[11],16);
b[11]-=ROL(b[10],7);
b[12]-=ROL(b[13],6);
b[13]-=ROL(b[12],21);
b[14]-=ROL(b[15],28);
b[15]-=ROL(b[14],3);
b[16]-=ROL(b[17],18);
b[17]-=ROL(b[16],17);
b[18]-=ROL(b[19],8);
b[19]-=ROL(b[18],31);
b[20]-=ROL(b[21],30);
b[21]-=ROL(b[20],13);
b[22]-=ROL(b[23],20);
b[23]-=ROL(b[22],27);
```

```
b[24]-=ROL(b[25],10);
b[25]-=ROL(b[24],9);
b[26]-=b[27];
b[27]-=ROL(b[26],23);
b[28]-=ROL(b[29],22);
b[29]-=ROL(b[28],5);
b[30]-=ROL(b[31],12);
b[31]-=ROL(b[30],19);

b[32]-=ROL(b[33],2);
b[33]-=ROL(b[32],1);
b[34]-=ROL(b[35],24);
b[35]-=ROL(b[34],15);
b[36]-=ROL(b[37],14);
b[37]-=ROL(b[36],29);
b[38]-=ROL(b[39],4);
b[39]-=ROL(b[38],11);
b[40]-=ROL(b[41],26);
b[41]-=ROL(b[40],25);
b[42]-=ROL(b[43],16);
b[43]-=ROL(b[42],7);
b[44]-=ROL(b[45],6);
b[45]-=ROL(b[44],21);
b[46]-=ROL(b[47],28);
b[47]-=ROL(b[46],3);
b[48]-=ROL(b[49],18);
b[49]-=ROL(b[48],17);
b[50]-=ROL(b[51],8);
b[51]-=ROL(b[50],31);
b[52]-=ROL(b[53],30);
b[53]-=ROL(b[52],13);
b[54]-=ROL(b[55],20);
b[55]-=ROL(b[54],27);
b[56]-=ROL(b[57],10);
b[57]-=ROL(b[56],9);
b[58]-=b[59];
b[59]-=ROL(b[58],23);
b[60]-=ROL(b[61],22);
b[61]-=ROL(b[60],5);
b[62]-=ROL(b[63],12);
b[63]-=ROL(b[62],19);

for(i=0;i<(WIDTH/2);i++){
```

```
  a[2*i]=b[i];
  a[2*i+1]=b[i+(WIDTH/2)];
}

a[0]-=ROL(a[1],2);
a[1]-=ROL(a[0],1);
a[2]-=ROL(a[3],20);
a[3]-=ROL(a[2],11);
a[4]-=ROL(a[5],6);
a[5]-=ROL(a[4],21);
a[6]-=ROL(a[7],24);
a[7]-=ROL(a[6],31);
a[8]-=ROL(a[9],10),
a[9]-=ROL(a[8],9);
a[10]-=ROL(a[11],28);
a[11]-=ROL(a[10],19);
a[12]-=ROL(a[13],14);
a[13]-=ROL(a[12],29);
a[14]-=a[15];
a[15]-=ROL(a[14],7);
a[16]-=ROL(a[17],18);
a[17]-=ROL(a[16],17);
a[18]-=ROL(a[19],4);
a[19]-=ROL(a[18],27);
a[20]-=ROL(a[21],22);
a[21]-=ROL(a[20],5);
a[22]-=ROL(a[23],8);
a[23]-=ROL(a[22],15);
a[24]-=ROL(a[25],26);
a[25]-=ROL(a[24],25);
a[26]-=ROL(a[27],12);
a[27]-=ROL(a[26],3);
a[28]-=ROL(a[29],30);
a[29]-=ROL(a[28],13);
a[30]-=ROL(a[31],16);
a[31]-=ROL(a[30],23);

a[32]-=ROL(a[33],2);
a[33]-=ROL(a[32],1);
a[34]-=ROL(a[35],20);
a[35]-=ROL(a[34],11);
a[36]-=ROL(a[37],6);
a[37]-=ROL(a[36],21);
```

```
a[38]-=ROL(a[39],24);
a[39]-=ROL(a[38],31);
a[40]-=ROL(a[41],10),
a[41]-=ROL(a[40],9);
a[42]-=ROL(a[43],28);
a[43]-=ROL(a[42],19);
a[44]-=ROL(a[45],14);
a[45]-=ROL(a[44],29);
a[46]-=a[47];
a[47]-=ROL(a[46],7);
a[48]-=ROL(a[49],18);
a[49]-=ROL(a[48],17);
a[50]-=ROL(a[51],4);
a[51]-=ROL(a[50],27);
a[52]-=ROL(a[53],22);
a[53]-=ROL(a[52],5);
a[54]-=ROL(a[55],8);
a[55]-=ROL(a[54],15);
a[56]-=ROL(a[57],26);
a[57]-=ROL(a[56],25);
a[58]-=ROL(a[59],12);
a[59]-=ROL(a[58],3);
a[60]-=ROL(a[61],30);
a[61]-=ROL(a[60],13);
a[62]-=ROL(a[63],16);
a[63]-=ROL(a[62],23);


for(i=0;i<(WIDTH/2);i++){
  b[2*i]=a[i];
  b[2*i+1]=a[i+(WIDTH/2)];
}

b[0]-=ROL(b[1],2);
b[1]-=ROL(b[0],1);
b[2]-=ROL(b[3],16);
b[3]-=ROL(b[2],7);
b[4]-=ROL(b[5],30);
b[5]-=ROL(b[4],13);
b[6]-=ROL(b[7],12);
b[7]-=ROL(b[6],19);
b[8]-=ROL(b[9],26);
b[9]-=ROL(b[8],25);
```

```
b[10]-=ROL(b[11],8),
b[11]-=ROL(b[10],31);
b[12]-=ROL(b[13],22);
b[13]-=ROL(b[12],5);
b[14]-=ROL(b[15],4);
b[15]-=ROL(b[14],11);
b[16]-=ROL(b[17],18);
b[17]-=ROL(b[16],17);
b[18]-=b[19];
b[19]-=ROL(b[18],23);
b[20]-=ROL(b[21],14);
b[21]-=ROL(b[20],29);
b[22]-=ROL(b[23],28);
b[23]-=ROL(b[22],3);
b[24]-=ROL(b[25],10);
b[25]-=ROL(b[24],9);
b[26]-=ROL(b[27],24);
b[27]-=ROL(b[26],15);
b[28]-=ROL(b[29],6);
b[29]-=ROL(b[28],21);
b[30]-=ROL(b[31],20);
b[31]-=ROL(b[30],27);

b[32]-=ROL(b[33],2);
b[33]-=ROL(b[32],1);
b[34]-=ROL(b[35],16);
b[35]-=ROL(b[34],7);
b[36]-=ROL(b[37],30);
b[37]-=ROL(b[36],13);
b[38]-=ROL(b[39],12);
b[39]-=ROL(b[38],19);
b[40]-=ROL(b[41],26);
b[41]-=ROL(b[40],25);
b[42]-=ROL(b[43],8),
b[43]-=ROL(b[42],31);
b[44]-=ROL(b[45],22);
b[45]-=ROL(b[44],5);
b[46]-=ROL(b[47],4);
b[47]-=ROL(b[46],11);
b[48]-=ROL(b[49],18);
b[49]-=ROL(b[48],17);
b[50]-=b[51];
b[51]-=ROL(b[50],23);
```

```
  b[52]-=ROL(b[53],14);
  b[53]-=ROL(b[52],29);
  b[54]-=ROL(b[55],28);
  b[55]-=ROL(b[54],3);
  b[56]-=ROL(b[57],10);
  b[57]-=ROL(b[56],9);
  b[58]-=ROL(b[59],24);
  b[59]-=ROL(b[58],15);
  b[60]-=ROL(b[61],6);
  b[61]-=ROL(b[60],21);
  b[62]-=ROL(b[63],20);
  b[63]-=ROL(b[62],27);

  for(i=0;i<WIDTH;i++) a[i]=b[i];


}

#endif

void f_function(unsigned long key[2*NUM_ROUNDS][WIDTH],\
unsigned long factors[WIDTH],unsigned long max,\
unsigned long left[WIDTH],unsigned long right[WIDTH]){

  unsigned  long i,j,help[WIDTH];
  unsigned long long n,o,carry1;

  for(i=0;i<WIDTH;i++) help[i]=left[i];

#ifdef TOP

#ifdef PRIMARY

  carry1=0;
  for(j=0;j<WIDTH;j++){
    help[j]^=key[max][j];
    help[j]=modmult(help[j],factors[j]);
    n=(unsigned long long) key[max+1][j];
    o=(unsigned long long) help[j];
    n+=o;
    n+=carry1;
    carry1=(n>>INT_LENGTH) & 1;
    help[j]=n & 0xFFFFFFFF;
```

```
    if((j & 7)==7) carry1=0;
   }

#else

  carry1=0;
  for(j=0;j<WIDTH;j++){
    n=(unsigned long long) help[j];
    o=(unsigned long long) key[max][j];
    n+=o;
    n+=carry1;
    carry1=(n>>INT_LENGTH) & 1;
    help[j]=n & 0xFFFFFFFF;
    help[j]=modmult(help[j],factors[j]);
    help[j]^=key[max+1][j];
    if((j & 7)==7) carry1=0;
  }

#endif

#ifdef PHT

    pht(help);

#else

   ipht(help);

#endif

#else

#ifdef PHT

    pht(help);

#else
    ipht(help);

#endif

#ifdef PRIMARY
```

```
  carry1=0;
  for(j=0;j<WIDTH;j++){
    help[j]^=key[max][j];
    help[j]=modmult(help[j],factors[j]);
    n=(unsigned long long) key[max+1][j];
    o=(unsigned long long) help[j];
    n+=o;
    n+=carry1;
    carry1=(n>>INT_LENGTH) & 1;
    help[j]=n & 0xFFFFFFFF;
    if((j & 7)==7) carry1=0;
   }

#else

  carry1=0;
  for(j=0;j<WIDTH;j++){
    n=(unsigned long long) help[j];
    o=(unsigned long long) key[max][j];
    n+=o;
    n+=carry1;
    carry1=(n>>INT_LENGTH) & 1;
    help[j]=n & 0xFFFFFFFF;
    help[j]=modmult(help[j],factors[j]);
    help[j]^=key[max+1][j];
    if((j & 7)==7) carry1=0;
  }

#endif

#endif

  for(i=0;i<WIDTH;i++) right[i]^=help[i];

}

void crypt(unsigned long key[2*NUM_ROUNDS][WIDTH],\
unsigned long factors[WIDTH],unsigned long left[WIDTH],\
unsigned long right[WIDTH]){

unsigned long i,help;

  for(i=0;i<(NUM_ROUNDS/2);i++){
```

```
    f_function(key,factors,(4*i),left,right);
    f_function(key,factors,(4*i+2),right,left);
  }
  for(i=0;i<WIDTH;i++){
    help=right[i];
    right[i]=left[i];
    left[i]=help;
  }
}



void encrypt(unsigned long userkey[][WIDTH],unsigned long long size,\
  unsigned long left[][WIDTH], unsigned long right[][WIDTH]){

  unsigned long key[2*NUM_ROUNDS][WIDTH];
  unsigned long factors[WIDTH];
  unsigned long i,j,help1[WIDTH],help2[WIDTH];
  unsigned long long m;

  key_schedule(userkey,key);
  encryption_factors(factors);
  for(i=0;i<WIDTH;i++){
    help1[i]=0;help2[i]=0;
  }

#if defined(FORWARD)

  for(i=0;i<(NUM_ROUNDS);i++){
    crypt(key,factors,help1,help2);
    for(j=0;j<WIDTH;j++){
      key[2*i][j]=help2[j];
      key[2*i+1][j]=help1[j];
    }
  }

#else

  for(i=(NUM_ROUNDS);i>0;i--){
    crypt(key,factors,help1,help2);
    for(j=0;j<WIDTH;j++){
      key[2*i-1][j]=help1[j];
      key[2*i-2][j]=help2[j];
```

```
    }
  }
  #endif

  for(m=0;m<size;m++){
    crypt(key,factors,&(left[m][0]),&(right[m][0]));
  }

}

void decrypt(unsigned long userkey[][WIDTH],unsigned long long size,\
  unsigned long left[][WIDTH], unsigned long right[][WIDTH]){

  unsigned long key[2*NUM_ROUNDS][WIDTH];
  unsigned long factors[WIDTH];
  unsigned long i,j,help1[WIDTH],help2[WIDTH];
  unsigned long long m;

  key_schedule(userkey,key);
  encryption_factors(factors);
  for(i=0;i<WIDTH;i++){
    help1[i]=0;help2[i]=0;
  }

#if defined(FORWARD)

  for(i=0;i<(NUM_ROUNDS);i++){
    crypt(key,factors,help1,help2);
    for(j=0;j<WIDTH;j++){
      key[2*i][j]=help2[j];
      key[2*i+1][j]=help1[j];
    }
  }

#else

  for(i=(NUM_ROUNDS);i>0;i--){
    crypt(key,factors,help1,help2);
    for(j=0;j<WIDTH;j++){
      key[2*i-1][j]=help1[j];
      key[2*i-2][j]=help2[j];
    }
  }
```

```
  #endif

  invert_keys(key);
  for(m=0;m<size;m++){
    crypt(key,factors,&(left[m][0]),&(right[m][0]));
  }

}



int main(){
  unsigned long i,j;
  unsigned long left[1][WIDTH],right[1][WIDTH];

#if defined(BIG_KEY)
  unsigned long userkey[8][WIDTH];
#else
  unsigned long userkey[4][WIDTH];
#endif

  for(i=0;i<WIDTH;i++){
   left[0][i]=i;
   right[0][i]=i;
  }
#if defined(BIG_KEY)
  for(i=0;i<WIDTH;i++){
    for(j=0;j<8;j++) userkey[j][i]=WIDTH*j+i;
  }
#else
  for(i=0;i<WIDTH;i++){
    for(j=0;j<4;j++) userkey[j][i]=WIDTH*j+i;
  }
#endif

  encrypt(userkey,1ULL,left,right);
  for(i=0;i<WIDTH;i++) printf("%lx    %lx\n",left[0][i], right[0][i]);
  scanf("%ld",&j);
  decrypt(userkey,1ULL,left,right);
  for(i=0;i<WIDTH;i++) printf("%lx        %lx\n",left[0][i],right[0][i]);
  scanf("%ld",&j);
return(0);
}
```