Differential Cryptanalysis of Snefru, Khafre, REDOC-II, LOKI and Lucifer

Eli Biham Adi Shamir

The Weizmann Institute of Science
Department of Applied Mathematics and Computer Science
Rehovot 76100, Israel

Abstract

In [1,2] we introduced the notion of differential cryptanalysis based on chosen plaintext attacks. In [3,4] we described the application of differential cryptanalysis to Feal[12,11] and extended the method to known plaintext attacks. In this paper differential cryptanalysis methods are applied to the hash function Snefru[9] and to the cryptosystems Khafre[10], REDOC-II[14,6], LOKI[5] and Lucifer[7].

1 Introduction

The notion of differential cryptanalysis was introduced in [1,2,3,4]. In this paper differential cryptanalytic methods are applied to Snefru[9], Khafre[10], REDOC-II[14, 6], LOKI[5] and Lucifer[7].

Snefru[9] is a one way hash function suggested by Merkle as the Xerox secure hash function. In March 1990 a $1000 reward was offered to the first person to break the two-pass variant of Snefru by finding two messages which hash to the same value. A similar reward was later announced for breaking the four-pass variant of Snefru.

Khafre[10] is a fast software oriented cryptosystem suggested by Merkle. Although the number of rounds is not yet determined, the designer expects that almost all applications will use 16, 24 or 32 rounds.

REDOC-II[14,6] is a high speed confusion/diffusion/arithmetic cryptosystem suggested by Cryptech. REDOC-II has ten rounds, but even the one-round variant is claimed to be sufficiently strong since the round-function is very complicated. A reward of \$5000 was offered for the best theoretical attack performed on the one-round variant and a reward of \$20000 was offered for a practical known plaintext attack on the two-round variant.

LOKI[5] is a 64-bit key/64-bit block cryptosystem similar to DES which uses one twelve-bit to eight-bit S box based on irreducible polynomials in four S box entries. Two new modes of operation which convert LOKI into a hash function are defined.

Lucifer[7] is a S-P network cryptosystem designed by IBM prior to the design of DES. The variant of Lucifer with eight rounds has 128-bit blocks and 256-bit keys. Another version of Lucifer is described in [13].

The main results described in this paper are:

Two-pass Snefru is easily breakable within three minutes on a personal computer. Our attack can find many pairs which hash to the same values and can even find several messages hashing to the same hashed value as a given message. The attack is also applicable to three-pass and four-pass Snefru with complexities which are much better than the birthday attack. The attack is independent of the actual choice of the S boxes and one of its variants can even be used as a black box attack in which the choice of the S boxes is not known to the attacker.

Khafre with 16 rounds is breakable by a differential cryptanalytic chosen plaintext attack using about 1500 encryptions within about an hour on a personal computer. By a differential cryptanalytic known plaintext attack it is breakable using about 2^{38} encryptions. Khafre with 24 rounds is breakable by a chosen plaintext attack using about 2^{53} encryptions and using a differential cryptanalytic known plaintext attack it is breakable using about 2^{59} encryptions.

REDOC-II with one round is breakable by a differential cryptanalytic chosen plaintext attack using about 2300 encryptions within less than a minute on a personal computer. For REDOC-II with up to four rounds it is possible to find three bytes of the masks (created by 1280 byte key tables) faster than exhaustive search of the key. The three masks can even be found by a known plaintext attack.

LOKI with up to eleven rounds is breakable faster than exhaustive search by a differential cryptanalytic attack. We further show that every key of LOKI has 15 equivalent keys due to a key complementation property and thus the complexity of a known plaintext attack on the full 16-round version can be reduced to 2^{60}. Another complementation property can reduce the complexity of a chosen plaintext attack by another factor of 16 to 2^{56}. The two hash function modes of LOKI are shown to be insecure.

Lucifer with eight rounds is breakable within 2^{21} steps using 24 ciphertexts pairs. The other version of Lucifer reduced to eight rounds is even weaker.

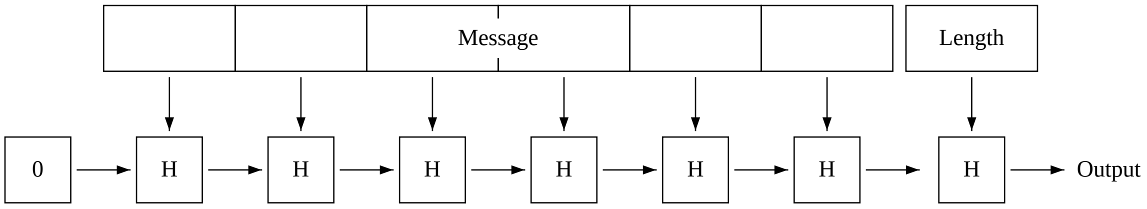


Figure 1. Outline of Snefru.

2 Cryptanalysis of Snefru

Snefru[9] is designed to be a cryptographically strong hash function which hashes messages of arbitrary length into m-bit values (typically 128 bits). The messages are divided into $(512 - m)$-bit chunks and each chunk is mixed with the hashed value computed so far by a randomizing function H. The function H takes a 512-bit input composed of the previous hashed value and the next chunk and calculates an m-bit output. The new hashed value is the output of H. More formally, for any $1 \leq i \leq \#c$:

$$h_i = \mathrm{H}(h_{i-1} \| c_i)$$

where $\#c$ is the number of chunks, '$\|$' is the concatenation operator of bit vectors, c_i is chuck number i and h_0 is an m-bit vector of zeroes. The final output is:

$$\text{output} = \mathrm{H}(h_{\#c} \| \text{length of message in bits}).$$

The process is outlined in figure 1.

The function H is based on a (reversible) 512-bit to 512-bit function E and returns a XOR combination of the first m bits of the input and the last m bits of the output of E. The function E randomizes the data in several passes. Each pass is composed of 64 randomizing rounds, where in each one of them a different byte of the data is used as an input to an S box whose output word is XORed with the two neighboring words. The codes of the functions H and E are given by figures 2 and 3. In the codes the block sizes are measured as 32-bit words and the values of the constants are:

`INPUT_BLOCK_SIZE`	=	16 (i.e., 512-bit block)
`OUTPUT_BLOCK_SIZE`	=	4 (for $m = 128$) or 8 (for $m = 256$)
`NO_OF_PASSES`	=	the number of passes (2, 3 or 4 passes).

A graphic description of the first 18 rounds of the function E is given in figure 4. Each row represents a round. Each column represents a word of data, which is composed of four bytes. The bytes used as inputs to the S boxes are surrounded by a thick rectangle. The words which are affected by the output of the S box in each round are painted in gray. After every group of 16 rounds the values of all the words are rotated.

```
function H (int32 input[INPUT_BLOCK_SIZE])
        returns int32 output[OUTPUT_BLOCK_SIZE]
{
  int32 block[INPUT_BLOCK_SIZE];

  block = E(input);

  for i = 0 to OUTPUT_BLOCK_SIZE-1 do
    output[i] = input[i] ⊕ block[INPUT_BLOCK_SIZE-i-1];

  return(output);
}
```

Figure 2. The function H.

A cryptographically strong hash function is broken if two different messages which hash to the same value are found. In particular, we break Snefru by finding two different chunk-sized messages which hash to the same value, or in other words, finding two inputs of the function H which differ only in the chunk part and have the same output. Unless specified otherwise, we concentrate in the following discussion on two-pass Snefru with $m = 128$ (whose chunks are 384-bit long).

A universal attack on hash functions is based on the birthday paradox. If we hash about $2^{m/2}$ random messages (2^{64} when $m = 128$) then with a high probability we find a pair of messages which hash to the same value. This attack is applicable to any hash function and is independent of its details.

For Snefru we designed a differential cryptanalytic attack which is independent of the choice of the S boxes. Its variants can even be used when the hash function is viewed as a black box with unknown S boxes.

The basic attack is as follows: choose a random chunk-sized message and prepend the 128-bit zero vector (or any previous hashed value calculated from previous chunks) to get the input of the function H. We create a second message from the first one by modifying the two bytes in the eighth and the ninth words which are used as inputs to the S boxes at rounds 56 and 57 (the fourth use of these words). We hash both messages by the function H and compare the outputs of the two executions. A fraction of 2^{-40} of these pairs of messages are hashed to the same value. Therefore, by hashing about 2^{41} messages we can break Snefru. As described later in this section, the number can be greatly reduced by using more structured messages.

In the basic attack we use a characteristic which differentiates only zero XOR values from non-zero XOR values and does not a priori fix the values of the non-zero XORs. In round 56 the byte from word eight is used to garble words seven and

```
function E (int32 input[INPUT_BLOCK_SIZE])
        returns int32 output[INPUT_BLOCK_SIZE]
{
  int32 block[INPUT_BLOCK_SIZE];
  int32 SBoxEntry;
  int shift, i, index, byteInWord;

  int shiftTable[4] = {16, 8, 16, 24};

  block = input;
  for index = 0 to NO_OF_PASSES-1 do {              (for each pass)
    for byteInWord = 0 to 3 do {
      for i = 0 to INPUT_BLOCK_SIZE-1 do {          (for each round)
        SBoxEntry = {fetch entry number block[i] mod 256 of S box
              number 2 · index + (i/2) mod 2};
        block[(i + 1) mod INPUT_BLOCK_SIZE] ⊕= SBoxEntry;
        block[(i - 1) mod INPUT_BLOCK_SIZE] ⊕= SBoxEntry;
      }
      shift = shiftTable[byteInWord];
      for i = 0 to INPUT_BLOCK_SIZE-1 do
        block[i] = {rotate block[i] by shift bits to the right};
    }
  }

  return(output);
}
```

Figure 3. The function E.

nine. In a fraction of about 1/256 of the pairs the garbled version of the byte in the ninth word cancels its chosen XOR between pairs. Therefore, for this fraction the XOR of this byte after round 56 is zero and the same values are XORed to the tenth word in both executions. The same values are used as inputs to the S boxes in both executions till the next time a byte of word seven is used at round 71. Round 71 garbles words six and eight by a different value for each execution and so does round 72 to words seven and nine. In a fraction of about 1/256 of the pairs the garbled version of the byte used as input to round 73 in the ninth word again cancels its previous XOR value. Therefore, for this fraction the XOR of this byte after round 72 is zero and the same values are XORed to the tenth word in both executions. The same values are used as inputs to the S boxes in both executions till the next time a byte of word six is used at round 86. The same cancellation should take place five times in rounds 56, 72, 88, 104 and 120. Therefore, the characteristic's probability is about $(1/256)^5 = 2^{-40}$. Each right pair with respect to this characteristic has zero

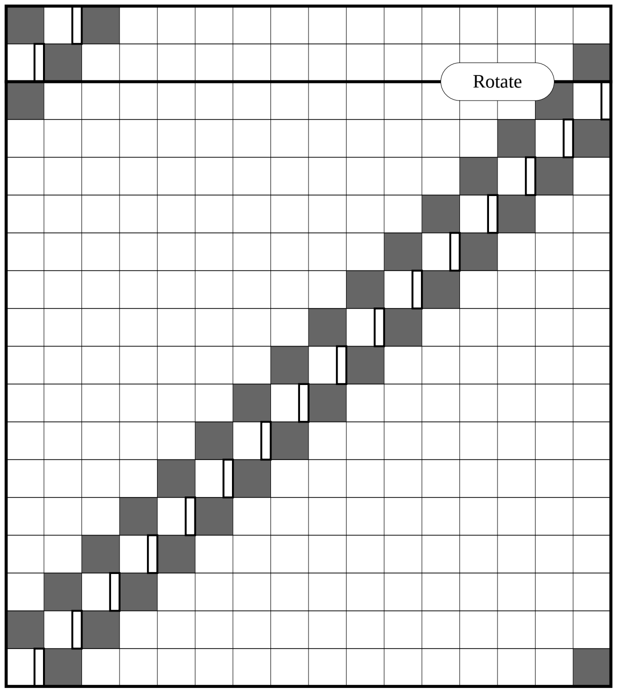


Figure 4. Graphic description of the first 18 rounds of the function E.

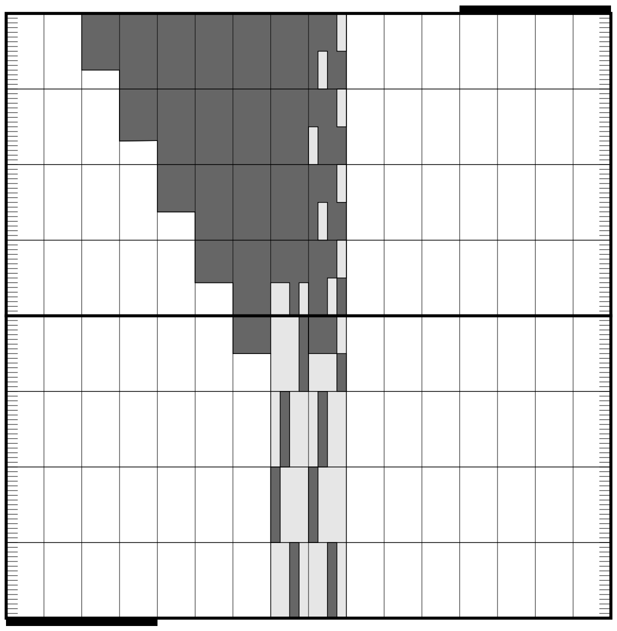


Figure 5. Graphic description of the characteristic.

XORs at the first m bits of the input and at the last m bits of the output and thus both messages are hashed to the same value. Figure 5 is a graphic description of the characteristic. In the figure each column represents a word of data and each row represents 16 rounds (represented by the thin lines along the edges). The gray area in the middle represents the modified words (non-zero XORs) in the characteristic.

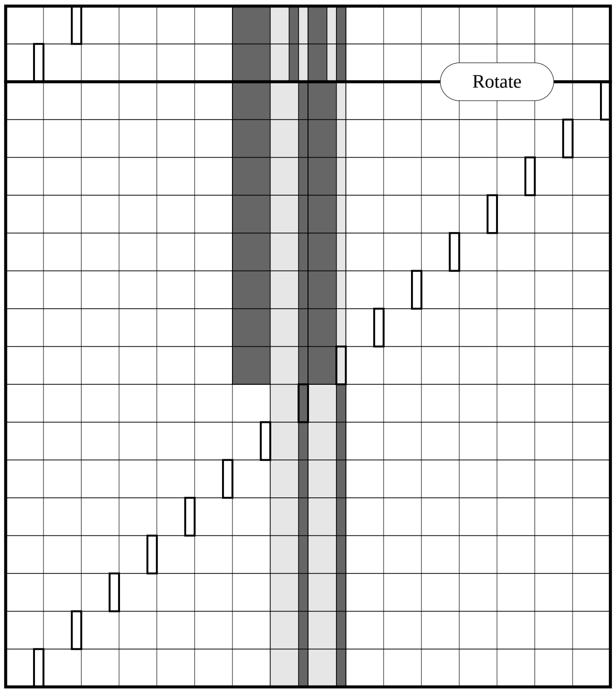


Figure 6. Zoomed part of the characteristic.

The brighter gray area represents the bytes with zero XORs in these words. The two black lines at the top-right and the bottom-left corners point to the words which are used in the calculation of the hash value by the function H (for $m = 128$). Since both of them occur in the white (unmodified) part of the block, the two messages hash to the same value. Figure 6 describes the modified bytes in intermediate rounds of the characteristic. In this figure each row represents a round. This same attack can break two-pass Snefru with m up to 224 bits. Similar attacks with modification of bytes of three to seven consecutive words of the input XOR of the characteristic are possible with the same characteristic's probability. Figure 7 describes a characteristic which modifies seven bytes.

This attack can be enhanced by using structures of messages. If we choose randomly about $2^{20.5}$ messages out of the 2^{24} messages which differ only in three bytes and hash them we get about $\frac{(2^{20.5})^2}{2} = 2^{40}$ legal pairs of messages to the attack. With high probability such a structure contains a right pair, i.e., a pair whose two messages hash to the same value, and such a pair can be easily found by sorting the $2^{20.5}$ hashed values. A variant of this attack can find a pair of messages composed only from ASCII letters or digits by hashing about $2^{20.5}$ messages which differ by the appropriate subset of bits in four bytes. By modifying up to seven bytes (which is the limit of this attack on two-pass Snefru) we can find pairs of messages hashing to the same value which are composed only from ASCII capital letters, only from ASCII digits or even from sets of eight different characters (for example octal digits) with the same complexity (since $\frac{(8^7)^2}{2} = 2^{41} > 2^{40}$). This attack can also be used when Snefru is considered as a black box which hides the choice of the S boxes.

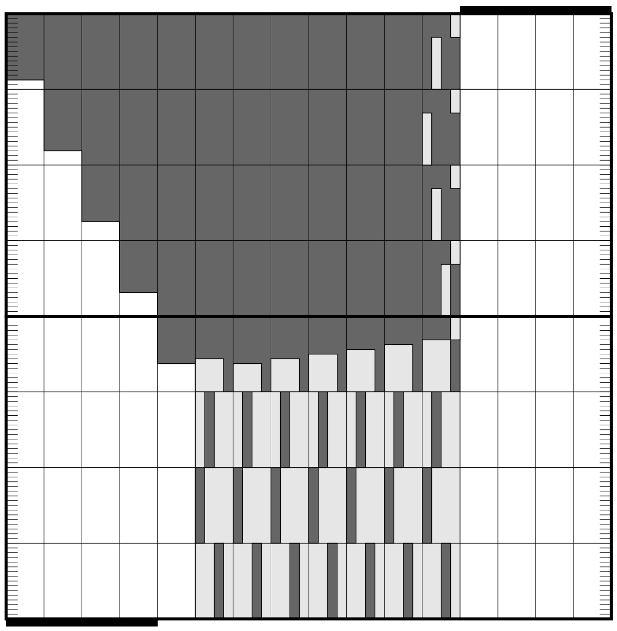


Figure 7. A characteristic with modification of seven bytes.

No. of passes	m	Char prob	#mod bytes	Complexity of attack	Birthday complexity	Comments
2	128–192	2^{-40}	3	$2^{20.5}$	2^{64}–2^{96}	
	224		2	2^{25}	2^{112}	
	128–192	2^{-40}	4	$2^{20.5}$	2^{64}–2^{96}	Alphanumeric
	224		2	2^{29}	2^{112}	messages
3	128	2^{-72}	3	2^{49}	2^{64}	
	160		2	2^{57}	2^{80}	

Table 1. Summary of the results of the black box attacks on Snefru.

In a black box attack on three-pass Snefru with $m = 128$ we can modify only three bytes and the characteristic's probability is 2^{-72}. Using structures of 2^{24} messages we obtain about $\frac{(2^{24})^2}{2} = 2^{47}$ pairs in each structure. Therefore, about $\frac{2^{72}}{2^{47}} \cdot 2^{24} = 2^{49}$ messages should be hashed. For three-pass Snefru with $m = 160$ only two bytes can be modified and the complexity of the attack becomes 2^{57}.

The black box attacks are independent of the (unknown) S boxes. The attack is applicable even if different S boxes are used in different rounds. A summary of the black box attacks on Snefru is given in table 1. Only one byte is modified in each word.

An important observation is that the modification of the bytes may be done at an intermediate round rather than in the message itself. In this case we choose a message and hash it, while remembering the value of the data block in the intermediate round.

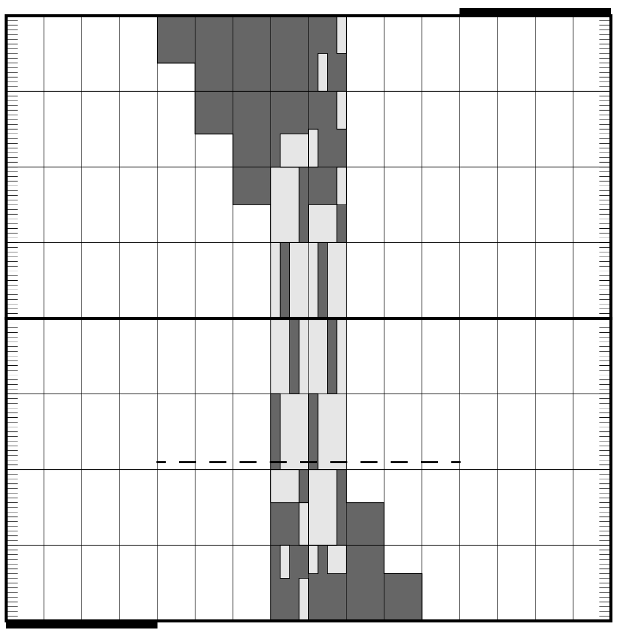


Figure 8. A characteristic with modification at an intermediate round.

We modify the value of bytes of consecutive words that are used in consecutive rounds in the computation. Then, the input of the function E is calculated backwards and its output is calculated forward. From the input and the output of E we calculate the output of H. Figure 8 describes a characteristic which modifies the data at an intermediate round. The intermediate round is denoted by a dashed line.

Another observation is that the values of the last and the first modified bytes can be chosen directly. For each choice of the modifications of all the bytes except the last, there is exactly one possibility for the modified value of the last byte which cancels the difference from the previous word. This value can be easily calculated and thus we can save a factor of 2^8 relative to the characteristic's probability. The first modified bytes can also be chosen (with a small loop) to save another factor of 2^8. Therefore, a total factor of 2^{16} can be saved. Additional choices of bytes do not change the complexity.

An extension of these observations makes it possible to modify more than one byte (up to four bytes) in each word and to choose up to twice the number of modified bytes in a word plus one (depending on the exact characteristic). A characteristic which modifies only one byte in each word is called a *simple characteristic.* A characteristic which modifies more than one byte in a word is called a *complex characteristic.* Note that all the black box attacks described above use simple characteristics (although it is not necessary).

The probability of the simple characteristics of two-pass Snefru described earlier in this section is 2^{-40}. By modifying four bytes at an intermediate round and choosing directly the last and the first of them we get 2^{16} possible data blocks from which we

No. of passes	m	Char prob	#mod bytes	Complexity of attack	Birthday complexity	Comments
2	128–192	2^{-40}	4·1 (2)	$2^{12.5}$	2^{64}–2^{96}	
	224		2·1 (2)	2^{25}	2^{112}	
	224	2^{-56}	2·3 (4)	$2^{12.5}$	2^{112}	
	128–192	2^{-40}	4·1 (1)	2^{17}	2^{64}–2^{96}	Alphanumeric
	224		2·1 (1)	2^{29}	2^{112}	messages
3	128–160	2^{-72}	6·1 (2)	$2^{28.5}$	2^{64}–2^{80}	
	192	2^{-80}	4·2 (3)	$2^{28.5}$	2^{96}	
	224	2^{-96}	2·4 (5)	2^{33}	2^{112}	
4	128–192	2^{-160}	4·4 (9)	$2^{44.5}$	2^{64}–2^{96}	
	224	2^{-112}	2·2 (3)	2^{81}	2^{112}	

Table 2. Summary of the results of the attacks on Snefru with known S boxes.

choose and hash about $2^{12.5}$. The number of possible pairs is $\frac{(2^{12.5})^2}{2} = 2^{24}$. Each pair has probability of $2^{-40} \cdot 2^{16} = 2^{-24}$ to be a right pair. Therefore, hashing $2^{12.5}$ messages we can find a right pair with a high probability. This attack can be used for m up to 192 bits. Using a complex characteristic we can attack the $m = 224$ with the same complexity.

The probability of the simple characteristics of three-pass Snefru is 2^{-72}. By modifying six bytes in an intermediate round and choosing directly the last and the first of them we get 2^{32} possible data blocks, from which we choose and hash about $2^{28.5}$. The number of possible pairs is about $\frac{(2^{28.5})^2}{2} = 2^{56}$. Each pair has a probability of $2^{-72} \cdot 2^{16} = 2^{-56}$ to be a right pair. Therefore, hashing $2^{28.5}$ messages we can find a right pair with a high probability. Modification of six bytes makes it possible to use this attack up to $m \leq 160$. The attacks on three-pass Snefru with $m = 192$ and $m = 224$ hash about $2^{28.5}$ and 2^{33} messages respectively using complex characteristics.

The probability of the simple characteristics of four-pass Snefru is 2^{-104}. Using simple characteristics we can only break the variants with $m = 192$ and $m = 224$ with complexities 2^{81} and 2^{89} respectively. Using the complex characteristic with probability 2^{-160} described in figure 9 we can break four-pass Snefru with up to $m = 192$ with complexity $2^{44.5}$.

A summary of the attacks on Snefru with known S boxes is given in table 2. The number of modified bytes is denoted by the number of the modified words times the number of modified bytes in each modified word. The number in parentheses is the number of bytes chosen directly. The S boxes should be known but the attack is independent of their choice. The attack is applicable even if different S boxes are used in different rounds.

This attack can also find many partners which hash to the same value as a given

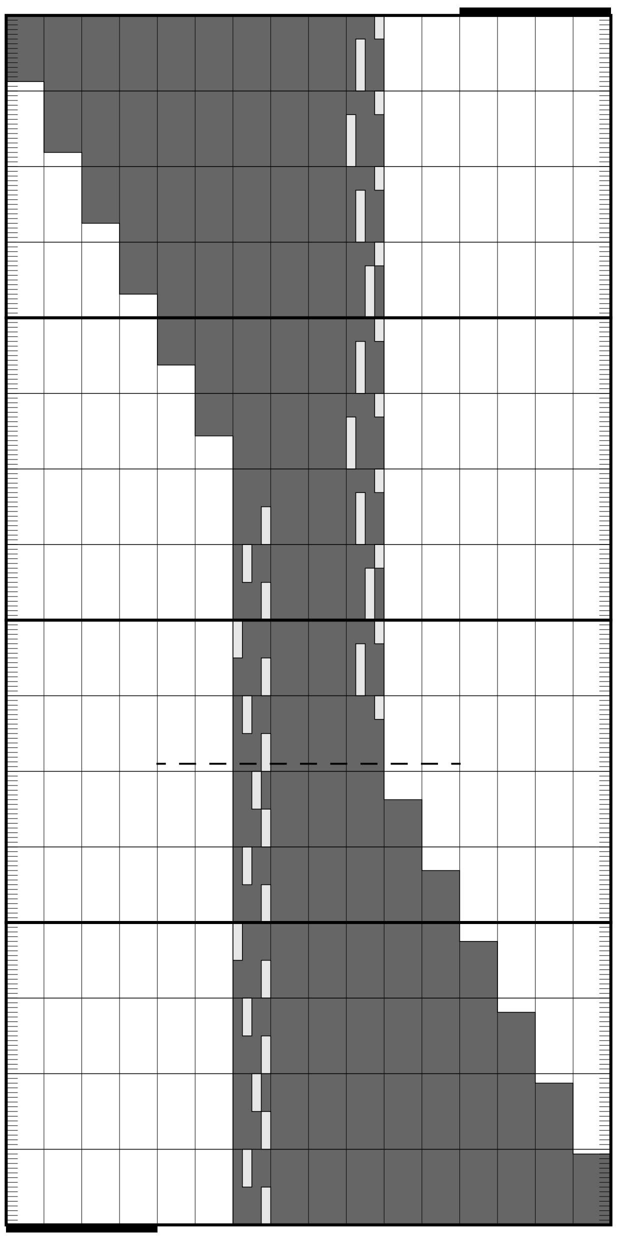


Figure 9. A complex four-pass characteristic.

message. For two-pass Snefru, given a message we create new messages by modifying the value of seven bytes by the characteristic in figure 7. By trying about 2^{40} such messages we can find with a high probability a second message which hashes to the same value as the given message. Moreover, the modification of the last modified byte (typically in word 12) may be chosen after the garbling from the previous bytes is known. Therefore, the value of this modified byte can be chosen directly to cancel the garbling, and can decrease the complexity of this attack by a factor of 2^8. If the modification is in a middle round it is possible to verify the value of the first modified byte after choosing the last one directly and decrease the complexity by a total factor of 2^{16} to about 2^{24} hash calculations. This variant can be applied to three-pass and

No. of passes	m	Char prob	#mod bytes	Complexity of attack	Brute force	Comments
2	128–160	2^{-40}	6·1 (2)	2^{24}	2^{128}–2^{160}	
	128–160		6·1 (0)	2^{40}	2^{128}–2^{160}	Black box
	128–224	2^{-64}	2·4 (5)	2^{24}	2^{128}–2^{224}	
	128–160	2^{-40}	7·1 (1)	2^{32}	2^{128}–2^{160}	Alphanumeric
	128–160		7·1 (0)	2^{40}	2^{128}–2^{160}	Alphanumeric, black box
3	128–224	2^{-96}	2·4 (5)	2^{56}	2^{128}–2^{224}	
4	128–192	2^{-160}	4·4 (9)	2^{88}	2^{128}–2^{192}	

Table 3. Summary of the results of the attacks which find partners of given messages.

four-pass Snefru as well. A summary of the attacks on Snefru which can find many partners of given messages is given in table 3.

A personal computer implementation of this attack on two-pass Snefru finds a pair of messages within three minutes. It finds a partner of a given message in about an hour.

The following two messages hash to the same value by two-pass Snefru. The messages are 48-byte long and are denoted as 12 words. The messages and the hashed value are given in hexadecimal.

- Message 1: `3fe15e26 23b7c030 c7089999 90efc48f a04d87ee 16493392 00046085 00003415 00000000 00000000 00000000 00000000`.
- Message 2: `3fe15e26 23b7c030 c7089999 90efc48f a9a09fee d74af7ae 096c7885 c19ef029 00000000 00000000 00000000 00000000`.
- Common hash value: `c8ff5e2c 8f9cf7c7 f08ddaa7 e4f9b44e`.

The following four messages hash to the same value as the (chosen) zero message:

- Message 1: `00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000`.
- Message 2: `00000000 f1301600 13dfc53e 4cc3b093 37461661 ccd8b94d 24d9d35f 71471fde 00000000 00000000 00000000 00000000`.
- Message 3: `00000000 1d197f00 2abd3f6f cf33f3d1 8674966a 816e5d51 acd9a905 53c1d180 00000000 00000000 00000000 00000000`.
- Message 4: `00000000 e98c8300 1e777a47 b5271f34 a04974bb 44cc8b62 be4b0efc 18131756 00000000 00000000 00000000 00000000`.

- Common hash value: `2e88e244 e9d4a208 b2d02fbb 72d0eee6`.

The following 36-byte messages hash to the same value by two-pass Snefru with $m = 224$:

- Message 1: `5bcc4d9b e1da3df2 a6fb6db0 002eef3f 00000007 00000000 00000000 00000000 00000000`.
- Message 2: `eb11879b e1da3d07 1626a76e 002eef3f 00000007 00000000 00000000 00000000 00000000`.
- Common hash value: `70c0577c 3feb6c47 42edcd49 a28241e3 b5e9fc88 1968f18f 1d712965`.

3 Cryptanalysis of Khafre

Khafre[10] is a software oriented cryptosystem with 64-bit blocks whose number of rounds (which should be a multiple of eight) is not yet determined. Each block is divided into two halves called the right half and the left half. In each round the lowest byte of the right half is used as input to an S box with 32-bit output. The left half is XORed with the output of the S box. The right half is rotated and the two halves are exchanged. The rotation is such that every byte is used once every eight rounds as an input to an S box. Before the first round and after every eighth round the data is XORed with 64-bit subkeys. These subkeys are the only way the key is involved in the cryptosystem.

The differential cryptanalysis of Khafre is based upon the observation that the number of output bits of an S box is more than twice the number of input bits. Therefore, given an output XOR of an S box in a pair, the input pair is (usually) unique and it is easy to find the two inputs. Moreover, there are about $\frac{(2^8)^2}{2} = 2^{15}$ possible input pairs for each S box, and thus only about 2^{-17} of the 32-bit values are output XORs of some pair.

A second observation is that there are characteristics in which only one even round (or only one odd round) has non-zero input XOR to the S box. The output XOR of this round in a right pair is easily derivable from the plaintext XOR and the ciphertext XOR. Given this output XOR we can discard most of the wrong pairs by the first observation, leaving only a small fraction of about 2^{-17} of them.

The characteristics of Khafre are described by templates which choose between zero XORs and non-zero XORs. Each right pair may have its own value of the non-zero XORs. The following characteristic is used as an example of the cryptanalysis of Khafre with 16 rounds. This characteristic is described as the first characteristic of Khafre due to its simplicity. Better characteristics are described later in this section.

Rnd	Left Half				Right Half					Output XOR			
Ω_P	0	0	A	0	0	0	B	0					
1	0	0	A	0	0	0	B	0	$\rightarrow$	0	0	0	0
2	B	0	0	0	0	0	A	0	$\rightarrow$	0	0	0	0
3	A	0	0	0	B	0	0	0	$\rightarrow$	0	0	0	0
4	0	B	0	0	A	0	0	0	$\rightarrow$	0	0	0	0
5	0	A	0	0	0	B	0	0	$\rightarrow$	0	0	0	0
6	0	0	0	B	0	A	0	0	$\rightarrow$	0	0	0	0
7	0	0	0	A	0	0	0	B	$\rightarrow$	C	D	E	$A^{\dagger}$
8	0	0	B	0	C	D	E	0	$\rightarrow$	0	0	0	0
9	D	E	0	C	0	0	B	0	$\rightarrow$	0	0	0	0
10	B	0	0	0	D	E	0	C	$\rightarrow$	$F \oplus B^{\ddagger}$	G	H	I
11	0	C	D	E	F	G	H	I	$\rightarrow$	J	K	$D \oplus L^{\ddagger}$	$E^{\dagger}$
12	I	F	G	H	J	M^0	L	0	$\rightarrow$	0	0	0	0
13	0	J	M^0	L	I	F	G	H	$\rightarrow$	N	$P \oplus J^{\ddagger}$	Q	$L^{\dagger}$
14	G	H	I	F	N	P	R^0	0	$\rightarrow$	0	0	0	0
15	R^0	0	N	P	G	H	I	F	$\rightarrow$	S	T	U	$P^{\dagger}$
16	H	I	F	G	V	T	W^0	0	$\rightarrow$	0	0	0	0
Ω_T	T	W^0	0	V	H	I	F	G					

Each value 0 describes a byte which has equal values in both executions of the encryptions of the pair (zero XOR). Each letter denotes a XOR value which is not zero. A letter with a superscript ^{0} denotes a XOR value which can be either zero or non-zero. The exact values of the non-zero XOR values vary for every right pair. The superscript † means that the byte of the output XOR must be equal to the corresponding byte of the left half in order to cause the input XOR byte of the S box in the next round to be zero. Each occurrence of † causes a reduction of the probability of the characteristic by $\frac{1}{255}$. The superscript ‡ means that the byte of the output XOR must not be equal to the corresponding byte of the left half in order to prevent a zero value in the corresponding byte in the next round, so that it can become zero in one of the following rounds, after XORing with another non-zero value. Each occurrence of ‡ causes a reduction of the probability of the characteristic by $\frac{254}{255}$. Therefore, the probability of this characteristic is $\left(\frac{1}{255}\right)^4 \cdot \left(\frac{254}{255}\right)^3 \approx 2^{-32}$. The input XOR Ω_P of the characteristic has two degrees of freedom: A and B can have 255 possible values. Therefore, the characteristic has $255^2 \approx 2^{16}$ possible plaintext XORs.

Given a sufficient number of pairs, we can discard most of the wrong pairs using the byte in the ciphertext XOR with value zero. Only about 2^{-8} of the wrong pairs remain.

Looking at the characteristic we can see that the output XOR of the tenth round is easily extracted by XORing the right half of the plaintext XOR with the right half of the ciphertext XOR and rotating the result by 16 bits (ROT16($R' \oplus r'$)). This happens since the tenth round is the only even round whose output XOR is not zero. There are 2^{32} possibilities for the value of ROT16$(R' \oplus r')$. However, there are only about 2^{15} possible input pairs of the S box itself. Therefore, there are at most about 2^{15} possible output XORs in the tenth round. As a consequence, most of the remaining wrong pairs can be easily discarded, leaving only about 2^{-17} of the 2^{-8} of the wrong pairs left by the previous test. In addition, the two input values of the S box and the two output values can be identified uniquely.

The input XOR value C of the S box in the tenth round equals the upper byte of the output XOR in the seventh round. The input XOR B and the lower byte of the output XOR A of the S box in the seventh round are known from the plaintext XOR. There are only 128 possible pairs of inputs (with that input XOR) to the S box in the seventh round. 16 bits of the output XOR of this S box are known. Therefore, we can discard each pair whose corresponding 16 bit value is not as expected. The probability of a wrong pair to pass this test is about $2^7 \cdot 2^{-16} = 2^{-9}$.

For each of the remaining pairs, we can find the actual values of the inputs to the S box in the fifteenth round since we know its eight bit input XOR and eight bits of its output XOR. There are only 2^7 pairs with this input XOR and therefore about half of the wrong pairs can be discarded. Then, we can calculate the input values to the S box in the thirteenth round by a similar calculation and discard about half of the remaining wrong pairs. The input values to the S box in the eleventh round can be found with much better identification, since all the 32 bits of the output XOR are known at this stage. We can discard most of the remaining wrong pairs leaving only about $2^7 \cdot 2^{-32} = 2^{-25}$ of them.

Up to now, we discarded almost all the wrong pairs, leaving only a negligible fraction of about $2^{-8} \cdot 2^{-17} \cdot 2^{-9} \cdot 2^{-1} \cdot 2^{-1} \cdot 2^{-25} = 2^{-61}$ of them (since the tests are independent). For the right pairs, we found the pairs of actual input values of the S boxes in all the five rounds with non-zero input XORs. We do not know which value belongs to which encryption in the pair, thus, we have two possible relations for each of these five values. We can find 16 possibilities for the lower byte of the left half of the last subkey by XORing through a trail from the tenth round forward to the ciphertext (two possible values of the input XOR of the tenth round and two possible values of the output XOR of each one of the eleventh, the thirteenth and the fifteenth rounds). Using the counting method with three right pairs among $3 \cdot 2^{32}$ pairs, we can identify the right pairs themselves. Once the right pairs are identified, we can uniquely identify the value of this byte of the subkey, and identify the exact choice of inputs to the S boxes in the five rounds for each encryption in the right pairs. Identification of the values of the input to the S box of the last round is possible using the counting method which identifies two more bytes of the last subkey. A similar identification may be done for the fourteenth round and then to the twelfth round, each finding two more bytes of the subkey. In total we find seven bytes of the last subkey. We can complete the value of the last subkey using another characteristic in which the first non-zero input XOR to an S box is in the eighth round, and reduce the cryptosystem to eight rounds (since in Khafre the subkeys are XORed into the data only once every eight rounds). The eight-round cryptosystem is already known to be breakable even if the S boxes themselves are unknown (see [10]).

This attack needs about three right pairs obtained from about $3 \cdot 2^{32}$ pairs ($3 \cdot 2^{33}$ ciphertexts). This number of ciphertexts can be drastically reduced by using a compact structure of 2^{16} encryptions which contains about 2^{31} pairs. Therefore, the structure has probability about half to contain a right pair. The structure is simple: choose a constant random value for six of the bytes of the plaintexts, excluding the second and the sixth bytes. Choose all the 2^{16} possible values for the second and sixth bytes of the plaintexts, resulting with a structure of 2^{16} plaintexts, and encrypt all the plaintexts. Note that this structure also contains pairs with the additional characteristic needed to complete the last subkey. In order to have about three right pairs, we have to choose about six such structures, with a total of about $6 \cdot 2^{16} \approx 400000$ plaintexts.

The attacking program finds the last subkey in less than 45 minutes on a personal computer using 400000 encryptions with 90% success rate. Using about 590000 encryptions the success rate is enlarged to more than 99% in about an hour execution time. The program uses about 500K bytes of memory most of which is used to store the plaintexts and the ciphertexts.

Differential cryptanalytic attacks can be converted to known plaintext attacks. This attack can be converted to a known plaintext attack using about $2^{41.5}$ plaintext/ciphertext pairs. In such an attack, the $2^{41.5}$ plaintexts can form $(2^{41.5})^2/2 = 2^{82}$ pairs. Since there are only 2^{64} possible plaintext XORs, about $2^{82}/2^{64} = 2^{18}$ pairs occur with each plaintext XOR. There are about 2^{16} usable input XORs of the characteristic and thus we get about $2^{16} \cdot 2^{18} = 2^{34}$ candidate pairs which can be used to break khafre with 16 rounds.

Characteristics with improved probability of about 2^{-24} also exist. One such characteristic is:

Rnd	Left Half				Right Half					Output XOR			
Ω_P	0	0	A	0	0	0	0	0					
1	0	0	A	0	0	0	0	0	$\rightarrow$	0	0	0	0
2	0	0	0	0	0	0	A	0	$\rightarrow$	0	0	0	0
3	A	0	0	0	0	0	0	0	$\rightarrow$	0	0	0	0
4	0	0	0	0	A	0	0	0	$\rightarrow$	0	0	0	0
5	0	A	0	0	0	0	0	0	$\rightarrow$	0	0	0	0
6	0	0	0	0	0	A	0	0	$\rightarrow$	0	0	0	0
7	0	0	0	A	0	0	0	0	$\rightarrow$	0	0	0	0
8	0	0	0	0	0	0	0	A	$\rightarrow$	B	C	D	E
9	0	0	A	0	B	C	D	E	$\rightarrow$	F	G	$H^0 \oplus A$	I
10	D	E	B	C	F	G	H^0	I	$\rightarrow$	$J^0 \oplus D$	$K^0 \oplus E$	$L \oplus B^{\ddagger}$	$C^{\dagger}$
11	H^0	I	F	G	J^0	K^0	L	0	$\rightarrow$	0	0	0	0
12	0	J^0	K^0	L	H^0	I	F	G	$\rightarrow$	M	$N \oplus J^{0\ddagger}$	$P^0 \oplus K^0$	$L^{\dagger}$
13	G	H^0	I	F	M	N	P^0	0	$\rightarrow$	0	0	0	0
14	P^0	0	M	N	G	H^0	I	F	$\rightarrow$	$Q^0 \oplus P^0$	R	$S^0 \oplus M$	$N^{\dagger}$
15	I	F	G	H^0	Q^0	R	S^0	0	$\rightarrow$	0	0	0	0
16	R	S^0	0	Q^0	I	F	G	H^0	$\rightarrow$	$T^0 \oplus R$	$U^0 \oplus S^0$	V^0	$W^0 \oplus Q^0$
Ω_T	F	G	H^0	I	T^0	U^0	V^0	W^0					

Using characteristics with probability about 2^{-24} we need about $3 \cdot 2^{24}$ pairs which are formed by $3 \cdot 2^{25}$ encryptions. Using structures of 2^8 encryptions which contain 2^{15} pairs the attack needs about $3 \cdot 2^{17}$ encryptions (same as with the characteristic with probability about 2^{-32}). Known plaintext differential cryptanalytic attacks based on this characteristic needs about $2^{41.5}$ encryptions (since $\frac{(2^{41.5})^2}{2 \cdot 2^{64}} \cdot 2^8 = 2^{26} > 3 \cdot 2^{24}$). The above characteristic can be extended to a 24-round characteristic with probability about 2^{-56}. Attacks on 24-round Khafre based on this characteristic needs about 2^{60} pairs. Using structures of 2^8 encryptions with 2^{15} pairs it needs about 2^{53} encryptions. The differential cryptanalytic known plaintext attack on 24-round Khafre based on this characteristic needs about $2^{58.5}$ encryptions (since $\frac{(2^{58.5})^2}{2 \cdot 2^{64}} \cdot 2^8 = 2^{60}$).

The best characteristic we have found is the following 16-round characteristic, which has probability about 2^{-16}:

Rnd	Left Half				Right Half					Output XOR			
Ω_P	0	0	A	0	0	0	0	0					
1	0	0	A	0	0	0	0	0	$\rightarrow$	0	0	0	0
2	0	0	0	0	0	0	A	0	$\rightarrow$	0	0	0	0
3	A	0	0	0	0	0	0	0	$\rightarrow$	0	0	0	0
4	0	0	0	0	A	0	0	0	$\rightarrow$	0	0	0	0
5	0	A	0	0	0	0	0	0	$\rightarrow$	0	0	0	0
6	0	0	0	0	0	A	0	0	$\rightarrow$	0	0	0	0
7	0	0	0	A	0	0	0	0	$\rightarrow$	0	0	0	0
8	0	0	0	0	0	0	0	A	$\rightarrow$	B	C	D	E
9	0	0	A	0	B	C	D	E	$\rightarrow$	F	G	$H^0 \oplus A$	I
10	D	E	B	C	F	G	H^0	I	$\rightarrow$	$J^0 \oplus D$	$K^0 \oplus E$	$L \oplus B^\ddagger$	$M \oplus C^\ddagger$
11	H^0	I	F	G	J^0	K^0	L	M	$\rightarrow$	$N^0 \oplus H$	$P^0 \oplus I$	$Q \oplus F^\ddagger$	$R \oplus G^\ddagger$
12	M	J^0	K^0	L	N^0	P^0	Q	R	$\rightarrow$	$S^0 \oplus M$	$T \oplus J^{0\ddagger}$	$U^0 \oplus K^0$	$L^\dagger$
13	R	N^0	P^0	Q	S^0	T	U^0	0	$\rightarrow$	0	0	0	0
14	U^0	0	S^0	T	R	N^0	P^0	Q	$\rightarrow$	$V^0 \oplus U^0$	W	$X^0 \oplus S^0$	$T^\dagger$
15	P^0	Q	R	N^0	V^0	W	X^0	0	$\rightarrow$	0	0	0	0
16	W	X^0	0	V^0	P^0	Q	R	N^0	$\rightarrow$	$Y^0 \oplus W$	$Z^0 \oplus X^0$	α^0	$\beta^0 \oplus V^0$
Ω_T	Q	R	N^0	P^0	Y^0	Z^0	α^0	β^0					

Two of the odd rounds (the ninth and the eleventh rounds) have non-zero output XORs. The XOR of these two output XORs (with a rotation of one of them) can be easily extracted for right pairs. Since this XOR is a combination of four outputs (rather than two in the previous characteristics), the identification of the right pairs is much more complex, but still possible. The differential cryptanalytic chosen plaintext attack based on this characteristic needs three right pairs which are found in $3 \cdot 2^{16}$ pairs. Using structures of 2^8 encryptions which contain 2^{15} pairs about $\frac{2^8}{2^{15}} \cdot 3 \cdot 2^{16} = 1536$ encryptions are needed. The application of this chosen plaintext attack takes about an hour on a personal computer. The known plaintext differential cryptanalytic attack based on this characteristic needs about $2^{37.5}$ encryptions (since $\frac{(2^{37.5})^2}{2 \cdot 2^{64}} \cdot 2^8 = 2^{18} > 3 \cdot 2^{16}$).

A summary of the results for 16-round Khafre and 24-round Khafre is given in table 4 which describes the number of pairs needed for the attack, the number of chosen plaintexts needed, and the number of known plaintexts needed. Note that these complexities are independent of the actual choice of the S boxes as long as the S boxes themselves are known to the attacker, and remain valid even if different S boxes are used in different rounds.

Rounds	Char prob	Pairs needed	Chosen plaintexts	Known plaintexts
16	2^{-16}	$3 \cdot 2^{16}$	1536	$2^{37.5}$
24	2^{-56}	2^{60}	2^{53}	$2^{58.5}$

Table 4. Summary of the results of the attacks on Khafre.

4 Cryptanalysis of REDOC-II

REDOC-II[14,6] is a ten-round cryptosystem with 70-bit blocks (arranged as ten bytes of seven bits). Each round contains six phases: (1) First variable substitution, (2) Second variable substitution, (3) First variable key XOR, (4) Variable enclave, (5) Second variable key XOR and (6) Variable permutation. Each phase modifies the data using tables. There are 16 predefined substitution tables which are used by the variable substitutions. There are 128 predefined permutation tables used by the variable permutation. There are 128 predefined enclave tables used by the variable enclave. All these tables are fixed and are given as part of the definition of REDOC-II. In addition, 128 ten-byte key tables and nine mask tables are calculated for each key by a key processing algorithm. In each variable key XOR phase one table is chosen by XORing the value of a specific byte in the data with a specific byte in the mask tables. The resulting value is the table number. All the bytes of the data except the choosing byte are XORed with the corresponding bytes in the chosen key table. In each variable substitution phase one table is chosen by XORing the value of a specific byte in the data with a specific byte in the mask tables. The table number is the resulting value modulo 16. All the bytes of the data except the choosing byte are substituted by the chosen substitution table. In each variable permutation phase one table is chosen by adding (modulo 128) all the ten bytes of the data and XORing the result with a specific byte in the mask tables. The resulting value is the table number. All the bytes of the data are permuted by the chosen permutation.

The variable enclave phase is more complicated. The predefined enclave tables have five rows and three columns. Each entry contains a number between one and five. There are two properties which an enclave table should satisfy: each column should be a permutation of the numbers 1–5 and each row should contain three different numbers. Processing an enclave table on a half-block is as follows:

1. Each entry in the table is the index of a byte in the half-block.
2. Add the values of the three bytes pointed by the numbers in the first row of the table and keep the result in the byte pointed by the first column in this row.
3. Add the resultant values of the three bytes pointed by the numbers in the second row of the table and keep the result in the byte pointed by the first column in the row.
4. Similarly add according to the third, fourth and fifth rows.

Each variable enclave phase uses four enclave tables as follows:

1. Divide the block into two half-blocks of five bytes each. The half-blocks are called the left half and the right half.
2. XOR the values of two particular bytes in the right half (in the first round: the first two bytes) with two particular mask bytes. The resultant two bytes are indexes of two enclave tables.
3. Process the left half by the first enclave table indexed by the above two bytes.
4. Process the resultant left half by the second enclave table indexed by the above two bytes.
5. XOR the values of two particular bytes in the resultant left half (in the first round: the first two bytes) with two particular mask bytes. The resultant two bytes are indexes of two enclave tables.
6. XOR the left half to the right half.
7. Process the resultant right half by the first enclave table indexed by the above two bytes.
8. Process the resultant right half by the second enclave table indexed by the above two bytes.
9. XOR the right half to the left half.

An important property of the enclave tables is that they are linear operations in terms of addition which can be simulated by a matrix-vector product. By modifying only upper bits in the input, only upper bits in the output are modified. Moreover, the linear modification table of the upper output bits by the upper input bits uniquely identifies the enclave table used. This property can even be used in the variable enclave phase. The left half of the input with two of the bytes of the right half affect the choice of the enclave tables used in this phase. However, three of the bytes of the right half do not affect the choice of the enclave tables (in the first round they are the eighth, ninth and tenth bytes) and thus the modifications of the upper bits of the output are linear functions of the modifications of the upper bits of these input bytes. Note that since we XOR the right half to the left half as the last step in the variable enclave phase we get a symmetric modification in both halves and therefore, an even number of modified upper bits.

In this attack we use the following one-round characteristic:

After Phase	Data XOR										
Ω_P	0	0	0	0	0	0	0	A	0	0	
First Subst	0	0	0	0	0	0	0	B	0	0	For some B
Second Subst	0	0	0	0	0	0	0	64	0	0	with probability about 1/128
Key XOR	0	0	0	0	0	0	0	64	0	0	
Enclave	C	0	D	E	F	C	0	D	E	F	with probability about 1/2
Key XOR	C	0	D	E	F	C	0	D	E	F	
Permutation	Some permutation of C,0,D,E,F,C,0,D,E,F										
Ω_T	Some permutation of C,0,D,E,F,C,0,D,E,F										

where $A, B \in \{1, \ldots, 127\}$ and $C, D, E, F \in \{0, 64\}$ (not all of them zero). In total this characteristic has probability about $\frac{1}{256}$. The ciphertext XOR has 60 zero bits (six in each byte) and the XORed value of the uppermost bits is zero as well. Similar characteristics exist in which the difference is at the ninth and tenth bytes rather than at the eighth byte. Differences in more than one of these three bytes is also possible with smaller probabilities, but if the difference is the same in all the differing bytes and the values of all the differing bytes in the plaintexts are equal then the probability remains about $\frac{1}{256}$.

Given sufficiently many pairs encrypted by one-round REDOC-II with the plaintext differences specified in the characteristics we can discard (almost) all the wrong pairs by verifying that the 61 bits of the ciphertext XORs $(60 + 1)$ are really zero. Only a negligible fraction of 2^{-61} of the wrong pairs may remain. In practice, only right pairs remain.

For each of the $16 \cdot 16 = 256$ possible values of the masks of the substitution phases we count the number of pairs whose differing byte after the two substitutions resulting from the masks differ only by the uppermost bit. For each one of the 128 possible values of the mask of the permutation phase we count the number of pairs whose ciphertext XOR permuted by the resulting inverse permutation is symmetric and has zeroes in the second and the seventh bytes. The right values of these mask bytes are likely to be the ones counted most frequently and thus can be identified. This attack needs about 1000 pairs and finds three masks of the processed key.

The attack can be enhanced by using structures of 32 encryptions with identical nine bytes and whose tenth byte has 32 different values. In such a structure there are 496 pairs. There are only 128 possible differences after the second substitution and thus there are about four pairs which differ only by one uppermost bit after the substitution phases. These four pairs use the same enclave tables and thus with probability about half the structure contains four right pairs, and with probability about half does not contain any right pair. Using three such structures with identical eight bytes, where 32 plaintexts differ by the ninth byte, 32 differ by the tenth byte and 32 differ by both the ninth and the tenth bytes with equal values in both bytes in each plaintext, we are guaranteed to have at least one structure whose choosing byte of the second key XOR has no difference and thus to have about four right pairs. This enhanced attack needs only 96 chosen plaintexts.

REDOC-II with more than one round is also vulnerable to this attack. The following characteristic is a two-round extension of the above characteristic (for simplicity we use in the second round the same choosing bytes as in the first round, rather than the new choosing bytes of the second round).

After Phase	Data XOR										
Ω_P	0	0	0	0	0	0	0	A	0	0	
First Subst	0	0	0	0	0	0	0	B	0	0	For some B
Second Subst	0	0	0	0	0	0	0	64	0	0	with probability about 1/128
Key XOR	0	0	0	0	0	0	0	64	0	0	
Enclave	C	0	D	E	F	C	0	D	E	F	with probability about 4/31 (see note †)
Key XOR	C	0	D	E	F	C	0	D	E	F	
Permutation	0	0	0	0	0	0	0	G	H	I	with probability about 1/15 (see note ‡)
First Subst	0	0	0	0	0	0	0	J	K	0	For some J and K
Second Subst	0	0	0	0	0	0	0	64	64	0	with probability about $(1/128)^2$
Key XOR	0	0	0	0	0	0	0	64	64	0	
Enclave	L	0	M	N	P	L	0	M	N	P	with probability about 1/2
Key XOR	L	0	M	N	P	L	0	M	N	P	(see note •)
Permutation	Some permutation of L,0,M,N,P,L,0,M,N,P										
Ω_T	Some permutation of L,0,M,N,P,L,0,M,N,P										

† One of C, D, E and F is 64 and the others are zero.

‡ Two of G, H and I are 64 and the third is zero. The probability that the permutation takes the two 64's into G, H and I is $\frac{\binom{3}{2}}{\binom{10}{2}} = \frac{3}{45} = \frac{1}{15}$. Assuming without loss of generality that $I = 0$.

• L, M, N and P are either zero or 64.

This characteristic has probability about $\frac{1}{128} \cdot \frac{4}{31} \cdot \frac{3}{45} \cdot \left(\frac{1}{128}\right)^2 \cdot \frac{1}{2} \approx 2^{-29}$ and the attack needs about 2^{31} pairs. Using structures of 128 encryptions whose differences are restricted to a single byte (either the eighth, ninth or the tenth byte) we are guaranteed to have 64 pairs whose difference after the first two substitution phases is only in one uppermost bit, and each of them has a probability of about 2^{-22} to be a right pair. Therefore, there is a right pair in such a structure with probability about 2^{-16} and the attack needs about $4 \cdot 2^{16} \cdot 128 = 2^{25}$ encryptions to find four right pairs and to deduce three masks. The extended three-round characteristic has probability about 2^{-50} and thus the attack needs about 2^{52} pairs. Using structures of 128 encryptions the attack needs about 2^{46} encryptions. The extended four-round characteristic has probability about 2^{-71} and thus the attack needs about 2^{73} pairs. Using structures of 128 encryptions the attack needs about 2^{67} encryptions. About $2^{73} \cdot 2^{-61} = 2^{12}$ wrong pairs may not be discarded, but the right values of the three masks can still be identified using the counting scheme which counts all the 15 bits simultaneously.

These chosen plaintext attacks can be converted to known plaintext attacks. Given $2^{35} \cdot \sqrt{2m}$ encryptions, there are about $\frac{\left(2^{35} \cdot \sqrt{2m}\right)^2}{2 \cdot 2^{70}} = m$ pairs with each plaintext XOR value. There are $3 \cdot 2^7$ possible plaintext XORs of pairs differing by one of the three bytes and therefore about $3 \cdot 2^7 \cdot m$ pairs with plaintext XORs appropriate to the attack are likely to exist in the data. Using the plaintext XORs which differ by more than one byte, this complexity changes to about $7 \cdot 2^7 \cdot m$. Since the attack on one-round REDOC-II needs about 1000 pairs, $7 \cdot 2^7 \cdot m = 1000$ and therefore $m \approx 1$. The number of encryptions needed for the known plaintext attack on one-round REDOC-II is about $2^{35} \cdot \sqrt{2m} \approx 2^{35.5}$. The attacks on REDOC-II with two, three and four rounds need about 2^{46}, $2^{56.5}$ and 2^{67} known plaintexts, respectively.

An extension of the attack on one-round REDOC-II can find all the masks and the key tables. We assume here that the three masks were already found and that the cryptosystem is reduced to three phases. In order to find all the key tables we use several structures of 128 encryptions which differ by one of the three bytes as above, plus several encryptions which differ also by the first two bytes.

The extension starts by calculating the matrix which describes the double enclave of the right half of the enclave phase. In the first step we look for the value of the entry which corresponds to the influence of the eighth input byte on the second output byte by trying the triplets of the value XORed with the input byte before it is multiplied, the multiplication factor and the value added after the multiplication from the other four input bytes. For each such triplet we check whether all the pairs in the structure suggest the same value to be XORed with the sum to make the output byte. The right value of the triplet should be suggested by all the pairs in the structure. Usually several triplets remain undiscarded, and all of them have the same factor. This factor should be the value of the corresponding entry in the matrix. The two entries which correspond to the ninth and to the tenth input bytes can be found similarly. Using the values of these three entries we can find more bits of the twelve entries of the matrix which correspond to the same three input bytes and to the four other output bytes. These values usually suffice to uniquely identify the pair of enclave tables used in the double enclave and to complete the matrix.

The following steps are described briefly. Find the values which are XORed with the inputs of the right half of the data (by the first key XOR phase and by the left half of the data after its double enclave). Find the values which are XORed with the output of the right double enclave to make the outputs. Derive the relationship between the values XORed with the inputs and the values XORed with the outputs, derive some entries of the key tables and calculate the masks of the right double enclave and the XOR of the masks of the two key XOR phases. Find more entries of the key tables by reversing the left double enclave and finding its masks. Complete the missing entries of the key tables using the additional encryptions (especially the second bytes of the key tables which cannot be found otherwise). Derive the actual indexes of the key tables and calculate the actual values of the missing masks from the key tables.

In addition to the chosen plaintext attacks, there are chosen ciphertext attacks which use characteristics based on the differences in the ciphertexts and show their evolution towards the plaintexts (i.e., in the reverse direction). One such characteristic of the one-round variant is:

Before Phase	Data XOR	
Ω_T	Some permutation of two 64's and eight 0's	
Permutation	Same values in both half blocks where one 64 is at bytes $i \in \{1,3,4,5\}$ and the other at byte $i+5$	with probability about 4/45
Key XOR	The same	
Enclave	0 0 0 0 0 0 0 A B C	with probability about 1/4
Key XOR	0 0 0 0 0 0 0 A B C	
Second Subst	0 0 0 0 0 0 0 ? ? ?	
First Subst	0 0 0 0 0 0 0 D E F	$(D,E,F) \neq (0,0,0)$
Ω_P	0 0 0 0 0 0 0 D E F	

This characteristic has probability about $\frac{1}{45}$. Similar characteristics with four differing bytes in the ciphertexts, six differing bytes and eight differing bytes have probabilities about $\frac{1}{140}$, $\frac{1}{210}$ and $\frac{1}{180}$ respectively. Using special structures, we can attack one-round REDOC-II using 40 chosen ciphertexts in order to find the three mask bytes. The variants with two, three and four rounds can be attacked using 2^{24}, 2^{45} and 2^{66} chosen ciphertexts respectively. The conversion of these attacks to known plaintext attacks gives approximately the same complexities as the attacks based on the chosen plaintext attacks.

The three masks of the substitution and the permutation phases of the one-round variant can be found within less than a second on a personal computer by a chosen plaintext attack. The attacking program on one-round REDOC-II finds all the masks and the key tables in about a minute using about 2300 encryptions with more than 90% success rate. Using about 3900 encryptions the success rate becomes better than 99%. The program uses about 150K bytes of memory. A summary of the results on REDOC-II is given in table 5.

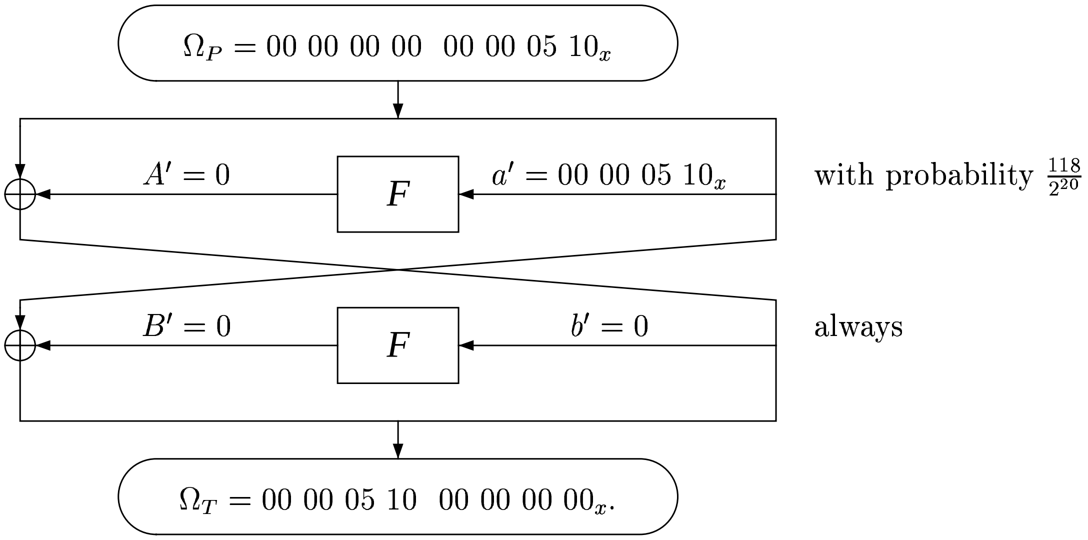
Rnds	Char prob	Pairs needed	Chosen plains	Chosen ciphers	Known plains	Comments
1	2^{-8}	–	2300	–	–	All masks + key tables
1	2^{-8}	1000	96	40	$2^{35.5}$	Three masks
2	2^{-29}	2^{31}	2^{25}	2^{24}	2^{46}	Three masks
3	2^{-50}	2^{52}	2^{46}	2^{45}	$2^{56.5}$	Three masks
4	2^{-71}	2^{73}	2^{67}	2^{66}	2^{67}	Three masks

Table 5. Summary of the results of the attacks on REDOC-II.

5 Cryptanalysis of LOKI

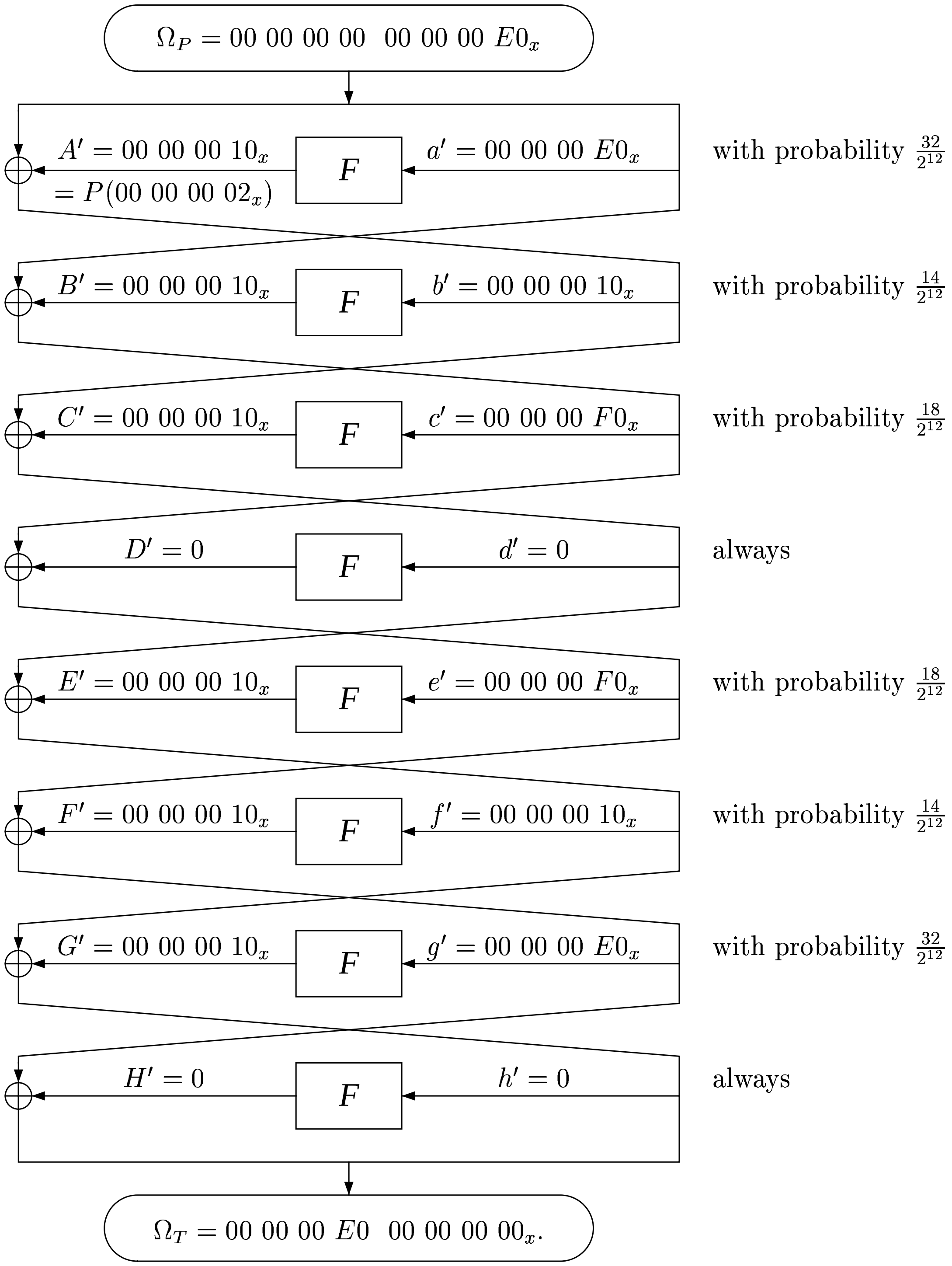
LOKI[5] is a 64-bit key/64-bit block cryptosystem similar to DES which uses one twelve-bit to eight-bit S box (based on irreducible polynomials) replicated four times in each round. The expansion and the permutation are replaced by new choices and the initial and final transformations are replaced by XORs with the key. The bit permutations in the key scheduling are replaced by rotations and the subkeys become 32-bit long. The XOR of the input of the F function with the key is done before the expansion and therefore neighboring S boxes receive common bits. Two new modes of operation which convert LOKI into a hash function are defined.

The pairs XORs distribution table of the larger S box of LOKI has much smaller probabilities than the ones of DES (average $\frac{1}{256}$ and maximum $\frac{1}{64}$). However, it is possible to have non-zero XORs in the inputs of two S boxes resulting with the same output, whereas in DES this requires at least three S boxes. We have found the following two-round iterative characteristic with probability $\frac{118}{2^{20}} \approx 2^{-13.12}$ (this probability is calculated using the observation that two neighboring S boxes have four common input bits, otherwise we get slightly smaller probability):



$\Omega_P = 00\ 00\ 00\ 00\ \ 00\ 00\ 05\ 10_x$
$A' = 0$
F
$a' = 00\ 00\ 05\ 10_x$
with probability $\frac{118}{2^{20}}$
$B' = 0$
F
$b' = 0$
always
$\Omega_T = 00\ 00\ 05\ 10\ \ 00\ 00\ 00\ 00_x$.

This characteristic can be iterated to nine rounds with probability about $2^{-52.5}$ and to eleven rounds with probability about $2^{-65.5}$. Since all the four S boxes of LOKI are the same and all the output XORs in this characteristic are zero, there are three similar characteristics in which the XOR pattern is rotated by units of eight bits. There is another eight-round iterative characteristic in which only non-replicated bits of some S box are different and the outputs differ only by one bit. This characteristic is:



This iterative characteristic has probability about 2^{-46} and its extension to nine rounds has the same probability. Using this characteristic it is possible to break LOKI with up to eleven rounds with less than 2^{64} chosen or known plaintexts.

Careful analysis of the structure of LOKI has revealed that any key has 15 equivalent keys which encrypt any plaintext to the same ciphertext due to a key complementation property. These 15 keys are the key XORed with the 15 possible 64-bit hexadecimal numbers whose digits are equal (i.e., $hhhhhhhhhhhhhhhh_x$ where $h \in \{1_x, \ldots, F_x\}$). Encryption with these keys results with the same inputs to the F functions in all the 16 executions. Therefore, most of the keys are redundant and a known plaintext attack can have complexity 2^{60} rather than 2^{64}.

Another complementation property is due to the observation that XORing the key with an hexadecimal value $gggggggghhhhhhhh_x$ (or with $hhhhhhhhgggggggg_x$) and XORing the plaintext by $iiiiiiiiiiiiiiii_x$ where $g \in \{0_x, \ldots, F_x\}$, $h \in \{0_x, \ldots, F_x\}$ and $i = g \oplus h$ results in XORing the ciphertext by $iiiiiiiiiiiiiiii_x$. This property can be used to reduce the complexity of a chosen plaintext attack by a further factor of 16 to 2^{56}.

These observations result in major weaknesses when LOKI is used as a hash function. For any message it is easy to find 15 additional messages which hash to the same value by the Single Block Hash (SBH) mode of LOKI: the other messages are the given message XORed with each of the 15 hexadecimal values $hhhhhhhhhhhhhhhh_x$. Since the messages are used as the key of the LOKI primitive (XORed with the previous hash value which can be viewed as a fixed value) and the plaintext of LOKI is fixed, the outputs of all the executions are the same by the first complementation property.

For any message it is easy to find 255 other messages which hash to the same value by the Double Block Hash (DBH) mode of LOKI provided the initial value is changed. This is done by XORing H_{-1} and M_2 by $gggggggghhhhhhhh_x$ and XORing M_1 by $hhhhhhhhgggggggg_x$ without changing H_0 (where $g \in \{0_x, \ldots, F_x\}$ and $h \in \{0_x, \ldots, F_x\}$).

LOKI has 256 simple fixpoints $\mathrm{LOKI}_K(X) = X$ where $K = gggggggghhhhhhhh_x$, $X = iiiiiiiiiiiiiiii_x$, $g, h \in \{0_x, \ldots, F_x\}$ and $i = g \oplus h$. In particular, LOKI encrypts the plaintext zero by the key zero to the ciphertext zero: $\mathrm{LOKI}_0(0) = 0$. Therefore, the two hash function modes hash the zero messages with the zero initial value to zero. This observation shows that the zero initial value should be avoided since any number of zero-blocks (or any even number in the DBH mode) can be prepended to the message without modifying the hash value. Moreover, in the SBH mode all the 16 initial values $H_0 = hhhhhhhhhhhhhhhh_x$ should be avoided since the message $00000000hhhhhhhh_x$ and 15 others hash to the initial value $H_1 = hhhhhhhhhhhhhhhh_x$. In the DBH mode all the 256 initial values $H_{-1} = 0$ and $H_0 = gggggggghhhhhhhh_x$ should be avoided since the messages $M_1 = hhhhhhhhgggggggg_x$ and $M_2 = iiiiiiiiiiiiiiii_x$ where $i = g \oplus h$ hash to the initial value and can be prepended any number of times without affecting the hash value.

After this research had been done, Matthew Kwan[8] found a three-round iterative characteristic of LOKI with probability $2^{-14.4}$ which can be use to break LOKI with up to 14 rounds. He also found many more fixpoints of LOKI.

6 Cryptanalysis of Lucifer

Lucifer[7] is a substitution/permutation network cryptosystem designed by IBM prior to the design of DES. In DES the output of the F function is XORed with the input of the previous round to form the input of the next round. This value is XORed (in turn) with key bits to form the input of the S boxes. In Lucifer the input of the S boxes is the bit permuted output of the S boxes of the previous round while the input of the S boxes of the first round is the plaintext itself. A key bit is used to choose the actual S box at each entry out of two possible S boxes. The other variant of Lucifer[13] is similar to DES, but is weaker than the variant attacked in this section.

Given an input of an S box, the outputs of the two possible S boxes are known. Each output bit may be equal in the two S boxes or may differ. Usually only one or two output bits are equal in the two S boxes. In few cases one output bit is equal in all the four output values obtained when two input values differing by one bit (for example 8_x and A_x) enter the two possible S boxes. There are pairs of inputs for which the same output bits stay fixed for both values and the same bits differ using either one of the two S boxes. In particular, there are pairs for which three output bits are equal and the fourth bit differ using either S box.

The published description of this variant of Lucifer does not specify the particular choice of S boxes. For the sake of concreteness, we use the third and fourth lines of S1 of DES as the S boxes S_0 and S_1 of Lucifer. Other choices of the S boxes give similar results. Table 6 describes the S boxes and the equal bits of the outputs of the two S boxes. We see that 11 inputs have two equal bits in the outputs, four inputs have one equal bit and for one input all the output bits differ. Table 7 describes the equal bits of two input values that differ by one bit using both S boxes. A binary notation is used in the tables.

Table 8 describes pairs that have many equal bits, s.t. the replacement of one input with the other in the pair leaves those output bits unchanged using either S box. In this table '`0`' and ' `1`' means that the output bit is '0' or '1' respectively at all the cases. '`+`' means that at either S box, the output bit is equal for both inputs of the pairs. '`-`' means the output bit value is different for the inputs of the pairs for either S box. '`.`' means that neither of the above cases holds.

By consulting these tables we can create many plaintexts whose particular (chosen) bit at an interior round has a chosen fixed value, regardless of the choice of the key. We can also create pairs of plaintexts which differ in a later round only at a particular bit. Lucifer with eight rounds can be attacked using the encryptions of such plaintexts.

Input	Output of S_0	Output of S_1	Equal bits
0000	0100	1111	.1..
0001	0001	1100	..0.
0010	1110	1000	1..0
0011	1000	0010	.0.0
0100	1101	0100	.10.
0101	0110	1001	....
0110	0010	0001	00..
0111	1011	0111	..11
1000	1111	0101	.1.1
1001	1100	1011	1...
1010	1001	0011	.0.1
1011	0111	1110	.11.
1100	0011	1010	.01.
1101	1010	0000	.0.0
1110	0101	0110	01..
1111	0000	1101	..0.

Table 6. Output bits that are equal for both S boxes.

Input	Equal bits
.000	.1..
0.00	.1..
001.	...0
.110	0...
10.0	...1
110.	.0..

Table 7. Output bits that are equal for both S boxes and two input values.

6.1 Attack 1

The following attack breaks eight-round Lucifer with 32-bit blocks, with the DES P permutation and with S boxes based on the third and fourth lines of S1 of DES. Other choices of the S boxes, permutation and the block size are breakable similarly.

Table 9 describes 450 plaintexts as a cartesian product of the specified inputs to the S boxes of the first round. These plaintexts cause the 17th bit in the input of the fourth round to be zero. The input and output data of the first few rounds is given

Input #1	Input #2	Common in S_0	Common in S_1	Common bits in Both S boxes
0001	1111	000-	110-	++0-
0010	1001	11-0	10--	1+-.
0011	1101	10-0	00-0	+0-0
1000	1010	1--1	0--1	+--1
1000	1101	1-1-	0-0-	+-+-
1010	1101	10--	00--	+0--
1011	1100	0-11	1-10	+-1+

Table 8. Output bits that are equal in pairs for either S box.

S1:	3_x, 6_x, A_x, C_x, D_x
S2:	2_x
S3:	A_x
S4:	0_x, 4_x, 8_x, B_x, E_x
S5:	6_x
S6:	7_x, 8_x, A_x
S7:	2_x, 3_x, D_x
S8:	2_x, 9_x

Table 9. Input values that cause zero at the fourth round.

in table 10. $\mathtt{I}_1$ is the plaintext. $\mathtt{O}_i$ are the output of the S boxes for the input $\mathtt{I}_i$. $\mathtt{I}_{i+1}$ is the input of round $i+1$ which is the bit permuted value of $\mathtt{O}_i$.

The key bits of the following rounds can be found by the following algorithm:

1. Try all the possible values of the key bits of the eighth, seventh and sixth rounds with the key bits of the four S boxes in the fifth round that are affected by the output of S5 in the fourth round, and the key bit of S5 in the fourth round

$\mathtt{I}_1$:	0011	0010	1010	0000	0110	0111	0010	0010	The first plaintext.
$\mathtt{O}_1$:	.0..	1..0	.0.1	.1..	00..	...1	...0	1...	For all the 450 plaintexts.
$\mathtt{I}_2$:	....	1100	....	10.0	0011	....	....	....	
$\mathtt{O}_2$:	....	.01.	....	...1	.0.0	....	....	....	
$\mathtt{I}_3$:	110.	....	....	.0..	....	....	...0	....	
$\mathtt{O}_3$:	.0..	....	....	....	....	....	....	....	
$\mathtt{I}_4$:	....	....	....	....	0...	....	....	....	

Table 10. Input and output of the separate rounds.

(total of 29 bits).

2. For each of them partially decrypt the ciphertexts to get the input bits of S5 in the fourth round. If any of them does not have zero at the proper bit then the tried key is wrong.

3. Using 40 encryptions we get a probability of 2^{-40} for a wrong key to survive, i.e., probability of about $2^{-40} \cdot 2^{29} = 2^{-11}$ that any wrong key remains. The real key must have zero for all the pairs and thus we find 29 key bits (out of $8 \times 8 = 64$).

4. Once these key bits are known, the other key bits can be found by a similar method with the same ciphertexts.

This algorithm has a time complexity of 2^{29} and needs 40 chosen plaintext encryptions.

There are similar attacks on Lucifer with 128-bit blocks with a chosen fixed bit in the fourth round (or possibly even the fifth round depending on the exact choice of the P permutation and the S boxes). In these attacks the above algorithm starts by finding 53 out of the $8 \times 32 = 256$ key bits, uses about 60–70 ciphertexts, and has a time complexity of about 2^{53}.

6.2 Attack 2

The following attack breaks eight-round Lucifer with 128-bit blocks. This attack is described in general terms to allow any choice of the P permutation.

In the preparation phase of the attack we choose an S box in the second round which will have inputs 8_x and A_x by the two members of the pairs. If its third bit (with value 2_x) comes from an S box in the first round from the output bit 1 (with value 8_x) then we try another S box (only about three quarters of the S boxes in the second round can be chosen). All the other inputs of the S boxes in the second round should be equal in the pair. At the first round we choose the following values for the bits of the two plaintexts:

1. One S box in the first round has an output bit which enters the third bit of the chosen S box in the second round. If this output bit is:

bit 2:	choose `1011` and `1100` as the inputs.
bit 3:	choose `0011` and `1101` as the inputs.
bit 4:	choose `0001` and `1111` as the inputs.

   In these cases the outputs differ only by this bit.

Rounds	Block size	Chosen plaintexts	Operations	Comments
8	128	60	2^{53}	Attack 1
8	128	24	2^{21}	Attack 2

Table 11. Summary of the results of the attacks on Lucifer.

2. For the other three S boxes which enter the chosen S box in the second round, choose input values (using table 6) whose output bit which enter the chosen S box in the second round is one which guarantee the values of $8_x/A_x$. These inputs are chosen to have the same values for both plaintexts in the pair.

3. For the other S boxes choose any value which should be the same for both plaintexts in the pair.

After the first round the partially encrypted values differ only by one bit (the output of the S box from clause 1 above). Thus, in the second round only one S box has different input values (1000 and 1010, respectively). In the output two bits differ. In the third round two S boxes have different inputs. Their outputs enter seven S boxes in the fourth round (they may enter eight S boxes but with a proper choice seven are possible). The output bits of the seven S boxes enter about 20–28 S boxes in the fifth round. Therefore, the outputs of at least four S boxes do not differ. In the sixth round we choose an S box with one of these bits as its input. We try all the possible values of the key bits of this S box, of the four affected S boxes in the seventh round and of the 16 affected S boxes in the eighth round. For each of their choices we verify the equality of the input bit in the sixth round. Since we try 2^{21} choices and each text has probability half to succeed we need about 24–30 pairs to find the value of the 21 key bits. Once these key bits are found, the other key bits can be found with a similar method using the known key bits.

This method can be applied to Lucifer with 32-bit blocks with 2^{21} operations and 12 encryptions.

A summary of the results on Lucifer is given in table 11.

References

[1] Eli Biham, Adi Shamir, *Differential Cryptanalysis of DES-like Cryptosystems (extended abstract)*, Lecture Notes in Computer Science, Advances in Cryptology, proceedings of CRYPTO'90, pp. 2–21, 1990.

[2] Eli Biham, Adi Shamir, *Differential Cryptanalysis of DES-like Cryptosystems*, Journal of Cryptology, Vol. 4, No. 1, pp. 3–72, 1991.

[3] Eli Biham, Adi Shamir, *Differential Cryptanalysis of FEAL and N-Hash (extended abstract)*, Lecture Notes in Computer Science, Advances in Cryptology, proceedings of EUROCRYPT'91, pp. 1–16, 1991.

[4] Eli Biham, Adi Shamir, *Differential Cryptanalysis of FEAL and N-Hash*, technical report CS91-17, Department of Applied Mathematics and Computer Science, The Weizmann Institute of Science, 1991.

[5] Lawrence Brown, Josef Pieprzyk, Jennifer Seberry, *LOKI - A Cryptographic Primitive for Authentication and Secrecy Applications*, Lecture Notes in Computer Science, Advances in Cryptology, proceedings of AUSCRYPT'90, pp. 229–236, 1990.

[6] Thomas W. Cusick, Michael C. Wood, *The REDOC-II Cryptosystem*, Lecture Notes in Computer Science, Advances in Cryptology, proceedings of CRYPTO'90, pp. 545–563, 1990.

[7] H. Feistel, *Cryptography and Data Security*, Scientific American, Vol. 228, No. 5, pp. 15–23, May 1973.

[8] Matthew Kwan, private communications.

[9] Ralph C. Merkle, *A Fast Software One-Way Hash Function*, Journal of Cryptology, Vol. 3, No. 1, pp. 43–58, 1990.

[10] Ralph C. Merkle, *Fast Software Encryption Functions*, Lecture Notes in Computer Science, Advances in Cryptology, proceedings of CRYPTO'90, pp. 476–501, 1990.

[11] Shoji Miyaguchi, Akira Shiraishi, Akihiro Shimizu, *Fast Data Encryption Algorithm FEAL-8*, Review of electrical communications laboratories, Vol. 36, No. 4, pp. 433–437, 1988.

[12] Akihiro Shimizu, Shoji Miyaguchi, *Fast Data Encryption Algorithm FEAL*, Lecture Notes in Computer Science, Advances in Cryptology, proceedings of EUROCRYPT'87, pp. 267–278, 1987.

[13] Arthur Sorkin, *Lucifer, a Cryptographic Algorithm*, Cryptologia, Vol. 8, No. 1, pp. 22–41, January 1984.

[14] Michael C. Wood, technical report, Cryptech Inc., Jamestown, NY, July 1990.