DES Ciphering Algorithm

Mohammed Jasem Reza
University of Al-Mustansiriya

Abstract

The main goal of this work is the implementation of cryptanalysis of DES and a statistical and theoretical analysis of its complexity and success probability. In order to achieve this goal, we implement first a very fast DES routine, resulting in a speed increase of almost 50 % towards the best known classical implementation. The experimental results suggest strongly that the attack is in average about 10 times faster as expected with plaintext-ciphertext at disposal; furthermore, we have achieved a complexity of by using only known pairs. Last, we propose a new analytical expression which approximates success probabilities; it gives slightly better results .

1. Introduction

The objectives of this project are the experiment and the analysis of linear cryptanalysis on DES. This attack was published in 1994, but no statistical analysis was possible at this time because computers were not fast enough. In this project, we first implement an efficient DES function, then run attack and finally make a statistical analysis of its complexity.

DES was an US encryption standard issued by NIST (previously NBS) in 1977 ([16]). In 1997, Biham proposed in [3] a parallel implementation inspired by SIMD (Single Instruction Multiple Data) architectures on regular computers which is the fastest at this time. According to Biham's analysis, one can perform 64 parallel DES computations within 16000 elementary CPU instructions on a 64-bit microprocessor, which leads to 222 DES computations per second with a single microprocessor working at 1 GHz.

So far, the best known attack on DES is linear cryptanalysis ([11, 12]). it is claimed that the complexity should consist in 243 DES computations on average. This leads to a one CPUmonth computation.

2. Data Encryption Standard

The DES has been a worldwide standard for the past 25 years. In 1972, the former American National Bureau of Standards (NBS), now called the National Institute of Standards and Technology (NIST), initiated a project with the goal of protecting computers and digital communications data. As part of this program, they wanted to develop a single, standard cryptographic algorithm. The motivations were the following, a single algorithm could be tested and certified more easily than thousand's; furthermore, it would be easier to let interoperate different cryptographic equipments using it.

The NBS issued a first public request for proposals in 1973; the number of received proposals indicated that there was a huge public interest in the field of cryptography, but very little public expertise. In fact, none of the submissions came only close to meeting the requirements.

A second request in 1974 brought the cipher Lucifer, developed in the IBM laboratories. After a secret review from the NSA (and the reduction of the key size from 128 to 56 bits !), and despite a lot of criticism because of its obscure role, the Data Encryption Standard was adopted as a federal standard in 1976 and authorized for use on all unclassified governmental communications one year later (see [16]).

The standard was recertified in 1983, 1987 and in 1993 without a lot of problems. In 1997, as it was showing some signs of old age and as it can no more be considered as a secure algorithm, the NIST has decided to launch a process in order to find a successor for the next 20 years (see [1]).

We recall here that it was possible in 1997 to build a hardware device which can run an exhaustive search of the key in less than 4 days with a budget of $ 200’000, see [7] for more details and listings. Knowing that agencies (or criminal organizations) have millions of $ at disposal, one can have a good idea of the actual security of DES. However, we have to note that variants of DES, like Triple-DES, are still considered to be very secure.

3. DES Algorithm

DES is a block cipher which encrypts data in 64-bits blocks, i.e. a 64-bits plaintext block goes in one of the end of the algorithm and a 64-bits cipher text block goes out of the other end. Furthermore, DES is a symmetric algorithm, the same algorithm and key being employed for both encryption and decryption (up to a minor modification in the key schedule). The key length is 56 bits, even if it is often expressed as a 64-bits block, the 8 less significant bits of each byte being used for parity checking purposes.

DES has a design related to two general concepts: the one of product cipher and the one of Feistel cipher. A product cipher combines two or more transformations (like substitutions, or permutations) in a manner intending that the result cipher is more secure than the individual components.

A Feistel cipher (see Figure 1 and Definition 1.1) is an iterated block cipher, i.e. involving the sequential repetition of an internal function called the round function.

Feistel Cipher

A Feistel cipher is an iterated cipher mapping a n = 2t bits plaintext (which we denote (L0,R0), for t-bits blocks L0 and R0, to a ciphertext (Rr,Lr), through a r-round process.

(Li,Ri) as follows:

$$\begin{cases} L_i = R_{i-1} \\ R_i = L_{i-1} \oplus f(R_{i-1}, K_i) \end{cases} \qquad (1)$$

where each subkey Ki is derived from the key K.
Usual parameters of an iterated cipher are the number of rounds r, the block bit size n, and the bit size k of the input key K from which r subkeys Ki are derived. For DES, r = 16, n = 64 and k = 56. The subkeys Ki have a size of 48 bits.
The Feistel cipher structure is guaranteed to be reversible (or, in other words, one can use the same function to encrypt and to decrypt the data).
Because XOR is used to combine the left half with the output of the round function, following equality holds:

$$L_{i-1} \oplus f(R_{i-1}, K_i) \oplus f(R_{i-1}, K_i) = L_{i-1} \qquad (2)$$

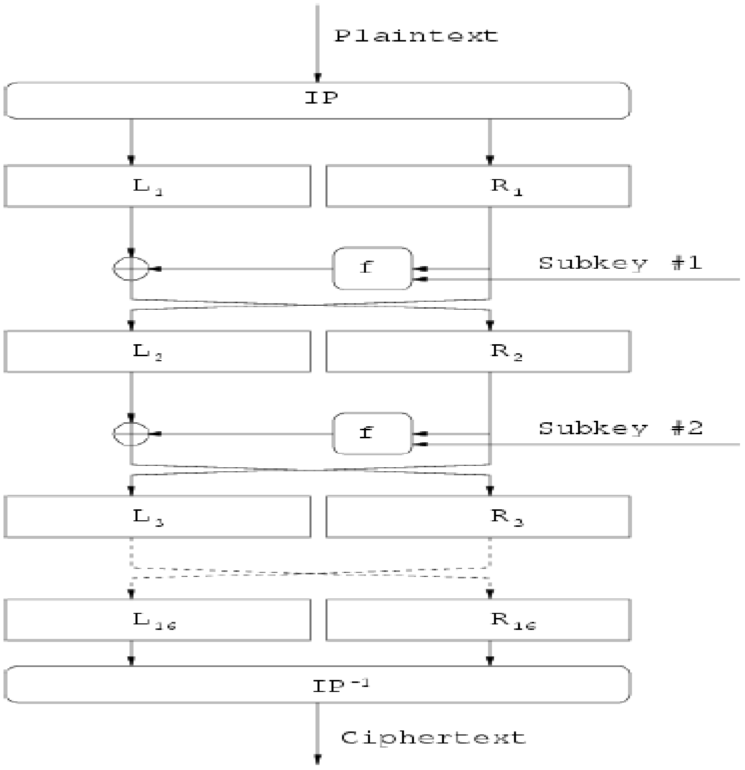

Figure 1: The Feistel cipher structure of DES

We can notice that the design of f doesn’t matter: for example, f don’t need to be invertible. As long as the inputs of f in each round can be reconstructed, one needs to implement only one algorithm for encryption and decryption.

DES operates on a 64-bits block of plaintext. After an initial permutation , the block is split into a right half R and a left half L, each 32-bits long. Then, following the Feistel cipher concept, there are 16 rounds of identical operations, called function f, in which the data are combined with 16 different subkeys Ki, which are derived from the key K using the key scheduling algorithm. At the end of the 16 rounds, the two parts L and R are combined and the inverse of IP finishes the algorithm.

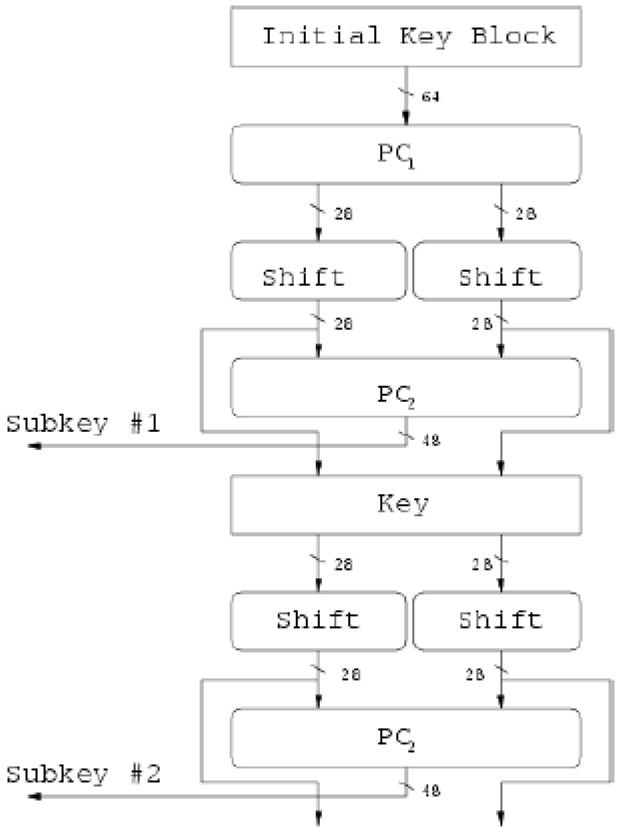

Figure 2: The key scheduling algorithm

3.1 The Key Schedule

As said before, the DES key is often expressed as a 64-bits block, where the least significant bits of each bytes are ignored and used as parity check to ensure that the key is error-free. This operation is implemented by the so-called permuted choice, denoted PC1, which eliminates the superfluous bits and permutes the remaining ones.

After this operation, a different 48-bits subkey is generated for each of the 16 rounds of DES in the following manner: first, the 56-bits key is divided into two 28-bits halves. Then, the halves are circularly shifted left by either one or two bits, depending of the round. After being shifted, 48 out of the 56 bits are selected by a compression permutation, often denoted PC2.
Because of the shifting, a different subset of key bits is used in each subkey.
Each bit is used in approximately 14 of the 16 rounds, but not all bits are used exactly the same number of times. The key scheduling algorithm is illustrated in Figure 2.

3.2 The f-function

The f-function processing is illustrated in Figure 3. One can find the detailed descriptions of DES, together with the exact description of EP, PP, PC1, PC2, IP, IP−1 and the parameters of the circular shifts in the key scheduling algorithm in several good books on cryptography ([15, 18]).

One round consists of the following operations: first, an expansion permutation, denoted EP, expands the 32 bits of the right half of the data Ri to 48 bits, which are XORed with the corresponding subkey; this sum will be the input of the substitution stage.

This operation changes the order of the bits as well as repeating certain bits. The goals of EP are multiple: it makes the right half the same size as the key for the XOR operation, it provides a longer results that can be compressed during the substitution operation. Furthermore, it allows one bit to affect two substitutions, so the dependency of the output bits on the input bits spreads faster. One calls this effect the avalanche effect.

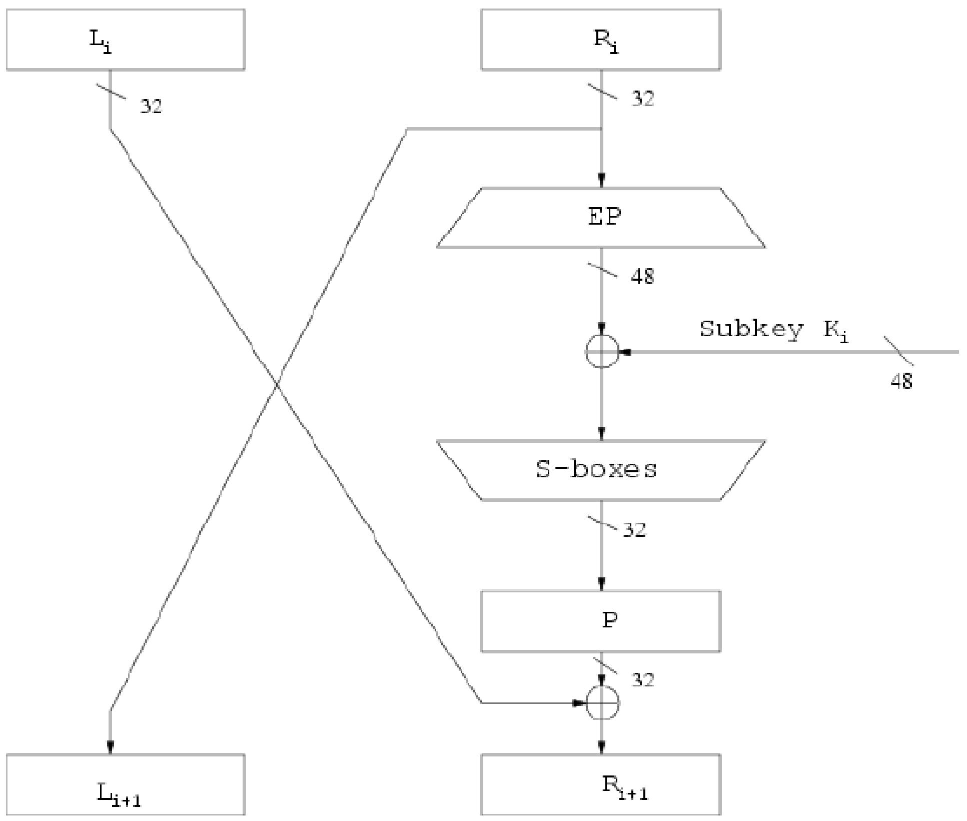

Figure 3: The f-function

The substitution stage is composed of eight different S-boxes. Each S-box has an input of 6 bits and a 4 bits output. The 48 bits are divided into eight 6-bits subblocks. Each separate block is operated on by a separate S-box.

A S-box is a table of 4 rows and 16 columns. The first and the last bit of the 6 input bits specify which row is used and the four inner bits specify the corresponding column.

The S-box substitution is the critical step in DES, regarding as well its implementation or its security. The algorithm’s other operations are all linear and easy to analyse, while the S-boxes are the only non-linear steps.
The end of the f-function consists of a straight permutation, the P-box permutation PP. This permutation maps each output bit of the substitution stage to an output position, i.e. no bits are used twice and no bits are ignored. Finally, the output of PP is XORed with the left half of the initial 64-bits block. Then, the left and right halves are permuted, following the Feistel cipher concept, and another round can begin.

4. Bitsliced Implementation

Implementing DES in software can be a very painful task in terms of speed.
In this section, we present first the classical way to implement DES in software, then we introduce the concept of bitslicing. In a next part, we present the Intel Pentium MMX architecture, which was the one we used to implement a fast DES routine, then we discuss some issues in our optimization work and finally we present the speed measures of our routine.

4.1 The Classical Way to Implement DES in Software

The main problem which arises while implementing DES in the classical way is to deal effectively with the permutations. One have to consider each bit in a register separately, which costs a lot of time. It is possible to use lookup-tables and streamlined operations, but these techniques are memory intensive, and the data quickly don’t fit anymore in the cache, which causes a severe slowing down of the code. An known advanced technique, which requires no memory, is the SWAPMOVE one. One of the quickest DES implementation available freely from the Internet, Eric Young’s one (see [20]), utilises it. The SWAPMOVE technique is described in Algorithm 1.1.

In this process, the bits in B, masked by M, are swapped with the bits in A, masked by M << N. It is possible, for example, to implement the initial permutation IP using five SWAPMOVE operations, i.e. a total of 30 logical operations.

As discussed in [17], it is straightforward to notice that a classical implementation of DES on modern 64-bits processors makes a very poor use of the computing power; for example, the XOR operation involving 32-bits values doesn’t use the full potential of the logical unit, and much time is wasted in dealing with the permutations, which can be seen as not calculating parts of the algorithm (this is more a data routing problem than a data transforming one !).
The bitslicing technique was first used in the cryptography field by Biham in[3]. In fact, this a known implementation trick among the electronicians.

The idea behind the bitslicing concept is quite simple: one allocates one register for each bit of data, instead of storing all the bits in an unique register.

This allows to process in a parallel way a number of bits which is equal to the size of the available registers.

Let's consider the DES algorithm: it is mainly built with permutations and substitutions. If we assign a register to a single bit, the permutations are dealt at compile-time, it is in fact an addressing problem. We don't have to isolate a special bit, which costs a lot of time, because we have the bit ready in a register, or in a memory location hard-coded in the program.

The only problem which remains to be solved is the substitutions one. Instead of using lookup tables, one have to express the S-boxes, which are the core of the substitution stage in DES, as their gate circuit, i.e. as a big boolean expression. Fortunately, as there is no loop in a S-box, we have no problem of conditional behaviour.

The evaluation implementation of the boolean expressions is typically more expensive than a lookup-table implementation, but the fact that we can evaluate 32 or 64 bits in parallel decreases the costs per S-box a lot.

Following [17], we recall here the advantages and drawbacks of a bitsliced Algorithm 1.1 The SWAPMOVE technique

Algorithm 1.1 The SWAPMOVE technique

```
SWAPMOVE(A, B, N, M)

T = ((A >> N) ^ B) & M;
B = B ^ T;
A = A ^ (T << N);
```

<u>4.2 implementation:</u>

The permutations costs nothing at execution time; they are hard-coded in the implementation.

The processor's logical unit is used at full rate.

The data are usually not available nor usable when they are spread
over a bunch of registers; this implies some conversion stages that
my be rather slow. This problem is known as the orthogonalisation
problem.

Table lookups are not possible anymore and have to be replaced by some logical computation, which may be rather painful to calculate and slow to execute. Furthermore, finding the optimal boolean evaluation of the S-boxes is not a trivial task, it is not even known if the current best schemes are optimal.

The resulting code is big and it is possible to loose some speed if the processor's code cache is not big enough.

In order to get some benefit from this technique, many registers are needed, as memory is slow.

This technique is very painful when implemented by hand.

Although the number of drawbacks seems bigger than the number of benefits, Biham showed in [3] that it is possible to gain a speedup of three towards the best classical implementation on a Compaq (former Digital) 64-bits processor.

5. Performance Results

We give here the results of the performance results of our DES routine.

We first give in Figure 5 the number of CPU cycles measurements of the raw code of the S-boxes. One have to notice that the input values of the S-boxes are not prefetched, and thus not available in the L1 cache.

S-box	1	2	3	4	5	6	7	8
CPU-cycles	283	271	276	262	279	272	275	269

Figure 5: CPU cycles for the S-boxes.

This gives in average 4.5 clock cycles per S-box per 64 bits encrypted block.

We give now the clock cycles measurement values of the whole DES computation; these measures take place in two different situations: in the first one, the data to be encrypted are not in the L1 cache while they are in the second situation.

We recall that the first situation corresponds to typical use of a DES routine, while the second one corresponds to our situation, where the data to be encrypted are in the cache at the beginning of the computation.

der to give a comparison with classical implementations, the DES library considered to be the fastest at this day, Eric Young's one (see [20]), written in assembly and optimized for a 32 bits architecture has a speed of 122 Mbps on an Intel PIII 666 MHz. One can compare this value with the one corresponding to the situation where the data aren't available in the cache.

Furthermore, one have not to forget that our implementation needs the data to be already spread, i.e. the orthogonalisation operation has to be done before encrypting. However, the goal was not to implement the fastest DES routine usable in the real world, but the fastest possible routine which runs under specific conditions, i.e. in the context of a linear cryptanalysis.

6. Conclusion

The aim of this research was to implement Matsui's linear cryptanalysis of DES. In order to achieve this goal, we have implemented a very fast DES routine which runs on the Intel Pentium III MMX architecture. We managed to run eight times the attack, breaking thus eight DES keys.

The experimental results have shown that the linear cryptanalysis of DES has a far lower complexity as expected by Matsui. This confirms what has been suggested several times in the literature: Matsui's estimations are pessimistic.

Furthermore, we have proposed an analytical expression which approximates the maximal rank probability of the right subkey in the list of candidates. This expression gives slightly better results than Matsui's experimental ones.

A lot of theoretical work is still necessary in order to give a really accurate measure of the average complexity of the attack. Anyway, it was very impressive and exciting to break a DES key several times with few computer resources. We would like here to thank Prof. Serge Vaudenay once again for having proposed such a wonderful diploma thesis subject, which allowed me programming and experimenting at a very low-level with a modern computer architecture as well as doing mathematics and trying to explain in a theoretical way the experimental results.

7.References

[1] AES Homepage. http://csrc.nist.gov/encryption/aes/.

[2] K. Aoki and H. Lipmaa. Fast implementation of AES candidates. In The Second AES Candidate Conference. National Institute for Standards and Technology, 1999.

[3] E. Biham. A fast new DES implementation in software. In FSE’97, volume 1267 of LNCS, pages 260–272. Springer-Verlag, 1997.

[4] E. Biham and A. Shamir. Differential Cryptanalysis of the Data Encryption Standard. Springer-Verlag, 1993.

[5] Intel Corporation. Intel architecture optimization reference manual,1999.

[6] Intel Corporation. Intel architecture software developer’s manual, 1999.

[7] Electronic Frontier Fondation. Cracking DES. O’Reilly, 1998.

[8] W. Feller. An introduction to probability theory and its applications.Wiley Series in Probability and Mathematical Statistics. Wiley, 1968.

[9] C. Harpes, G. Kramer, and J.L. Massey. A generalization of linear cryptanalysis and the applicability of Matsui’s piling-up lemma. In Advances in Cryptology - EUROCRYPT’95, volume 921 of LNCS, pages 24–38. Springer-Verlag, 1995.

[10] M. Kwan. Reducing the gate count of bitslice DES. http://eprint.iacr.org/2000/051.ps, 2000.

[11] M. Matsui. Linear cryptanalysis method for DES cipher. In Advances in Cryptology - EUROCRYPT’93, volume 765 of LNCS, pages 386–397. Springer-Verlag, 1993.

[12] M. Matsui. The first experimental cryptanalysis of the Data Encryption Standard. In Advances in Cryptology - CRYPTO’94, volume 839 of LNCS, pages 1–11. Springer-Verlag, 1994.

[13] M. Matsui. On correlation between the order of S-boxes and the strength of DES. In Advances in Cryptology - EUROCRYPT’94, volume 950 of LNCS, pages 366–375. Springer-Verlag, 1995.

[14] L. May, L. Penna, and A. Clark. An implementation of bitsliced DES on the pentium MMX TM processor. In Information Security and Privacy: 5th Australasian Conference, ACISP 2000, volume 1841 of LNCS. Springer-Verlag, 2000.

[15] A.J. Menezes, P.C. Van Oorschot, and S.A. Vanstone. Handbook of applied cryptography. The CRC Press series on discrete mathematics and its applications. CRC-Press, 1997.

[16] National Bureau of Standards. Data Encryption Standard. U. S. Department of Commerce, 1977.

[17] T. Pornin. Automatic software optimization of block ciphers using bitslicing techniques, 1999. ´Ecole Normale Sup´erieure, Paris.

[18] B. Schneier. Applied cryptography: protocols, algorithms and source code in C. John Wiley and Sons, 1994.

[19] S. Vaudenay. An experiment on DES statistical cryptanalysis. In 3rd ACM Conference on Computer and Communications Security, pages 139–147. ACM Press, 1996.

[20] E.A. Young. LibDES. http://www.openssl.org.

الخلاصه :

م. م. محمد جاسم رضا

الجامعه المستنصريه/ كلية التربيه الاساسيه

قسم الرياضيات

ان الهدف الاساس من هذه الدراسه هو تطبيق تحليل التشفير في (DES) وتطبيق تحليل نظري وأحصائي لتعقيده وزيادة أحتمالية نجاحه . ومن أجل تحقيق هذا الهدف نطبق أولا دورة (DES) السريعه جدا مما يؤدي الى زياده في السرعه بنسبة (50%) تقريبا نحو التطبيق الكلاسيكي المعروف أكثر من غيره . النتائج التطبيقيه تقترح وبشده بأن الخرق (الهجوم) يكون بمعدل عشر مرات أسرع مما هو متوقع مع وجود النص الواضح_ الجاهز للاستخدام . ثم نكون قد حققنا تعقيدا من خلال أستخدام الازواج المعروفه . وأخيرا نقترح صيغه تحليليه جديده تقرب من أحتمالية النجاح وتعطي نتائج أفضل قليلا .