Linear Cryptanalysis of DES

Diploma Thesis

Pascal Junod

Diploma Professor:	Prof. Dr. Ueli Maurer	ETH Zürich
Supervisor:	Prof. Dr. Serge Vaudenay	EPF Lausanne

to Mimi

Abstract

The main goal of this diploma work is the implementation of Matsui's linear 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 on the Intel Pentium III MMX architecture which is fully optimised for linear cryptanalysis. New implementation concepts are applied, 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 ($\mathcal{O}\left(2^{39}\right)$ DES computations) as expected with 2^{43} known plaintext-ciphertext at disposal; furthermore, we have achieved a complexity of $\mathcal{O}\left(2^{43}\right)$ by using only $2^{42.5}$ known pairs. Last, we propose a new analytical expression which approximates success probabilities; it gives slightly better results than Matsui's experimental ones.

Résumé

Le but principal de ce projet est l'implémentation de la cryptanalyse linéaire de DES, technique inventée par Matsui, et d'effectuer une analyse statistique de sa complexité. Dans ce but, nous avons implémenté une routine DES extrêmement rapide sur une architecture Intel Pentium III MMX. Des concepts très modernes ont été utilisés, ainsi que des optimisations rendues possibles par l'attaque, ce qui permet une augmentation de rapidité de 50 % par rapport à l'implémentation classique la plus rapide à ce jour. Les résultats expérimentaux suggèrent clairement que la complexité de l'attaque est en moyenne 10 fois moindre (à savoir $\mathcal{O}\left(2^{39}\right)$ évaluations de DES) par rapport à celle estimée par Matsui quand 2^{43} couples de textes clairs-chiffrés sont disponibles; de plus, nous obtenons une complexité de $\mathcal{O}\left(2^{43}\right)$ moyennant une quantité plus faible ($2^{42.5}$) de couples. Nous proposons enfin une expression analytique qui donne une approximation légèrement meilleure par rapport aux valeurs expérimentales de Matsui pour ce qui concerne la probabilité de succès de l'attaque.

Contents

Acknowledgements

First of all, I wish to express my profound gratitude to Professor Serge Vaudenay for accepting to supervise my diploma thesis and for having received me in his new laboratory.

I thank Professor Ueli Maurer for accepting to be my diploma professor, for allowing me to do it in Lausanne, and most important, for having shown me the beauty of cryptology during his lectures.

In no special order, I would now thank specially Dr. John Pliam, Dr. Stefan Wolf, Reto Kohlas, Eric Debes, Aslan Tchamkerten, Cyril Measson, Changyan Di, Nenad Buncic, and Sophie Vitali for their role in this work, or more generally in my studies. They all know why!

Last but no least, I would thank my parents for their support and Myriam for everything.

Subject

The objectives of this project are the experiment and the analysis of Matsui's 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 Matsui's attack and finally make a statistical analysis of its complexity.

DES was an US encryption standard issued by NIST (previously NBS) in 1977 ([15]). 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 2^{22} DES computations per second with a single microprocessor working at 1 GHz.

So far, the best known attack on DES is Matsui's linear cryptanalysis ([13, 12]). In the original paper, it is claimed that the complexity should consist in 2^{43} DES computations on average. This leads to a one CPU-month computation. The experiment however suggests a lower complexity.

The project consists in three phases which are proposed here in incremental difficulty levels:

- Implement a fast DES function by using Biham's technique.
- Run Matsui's attack and perform an experimental complexity analysis.
- Make a better theoretical complexity analysis.

1 The DES Cipher: Implementation and Optimisation

In this chapter, we make first a brief formal description of the DES cipher; in a second part, we give a detailed description of our bitsliced implementation of this algorithm, as well as the results of the performance measurements.

1.1 Historical Overview

The DES (Data Encryption Standard) 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 authorised for use on all unclassified governmental communications one year later (see [15]).

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 [16]).

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 [5] for more details and listings. Knowing that agencies (or criminal organisations) 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.

1.2 Definition

In this part, we give a detailed description of the DES algorithm. First, general explanations are given, then the key scheduling algorithm and finally the f-function are discussed.

1.2.1 General Outline

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 ciphertext 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*.

Definition 1.1 (Feistel Cipher)
A Feistel cipher *is an iterated cipher mapping a $n = 2t$ bits plaintext (which we denote (L_0, R_0), for t-bits blocks L_0 and R_0, to a ciphertext (R_r, L_r), through a r-round process, where $r \geq 1$. For $1 \leq i \leq r$, round i maps $(L_{i-1}, R_{i-1}) \underset{K_i}{\longrightarrow} (L_i, R_i)$ as follows:*

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

where each subkey K_i 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 K_i are derived. For DES, $r = 16$, $n = 64$ and $k = 56$. The subkeys K_i 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:

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

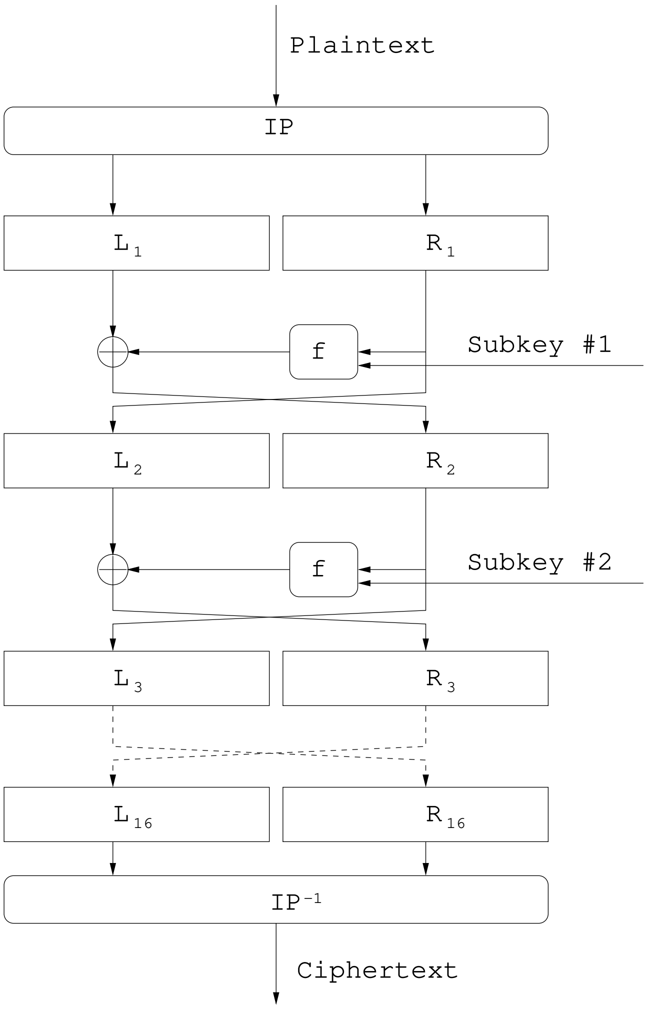

Figure 1: The Feistel cipher structure of DES

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 (denoted $\mathcal{IP}$), 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 K_i, 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 $\mathcal{IP}$ (denoted $\mathcal{IP}^{-1}$) finishes the algorithm.

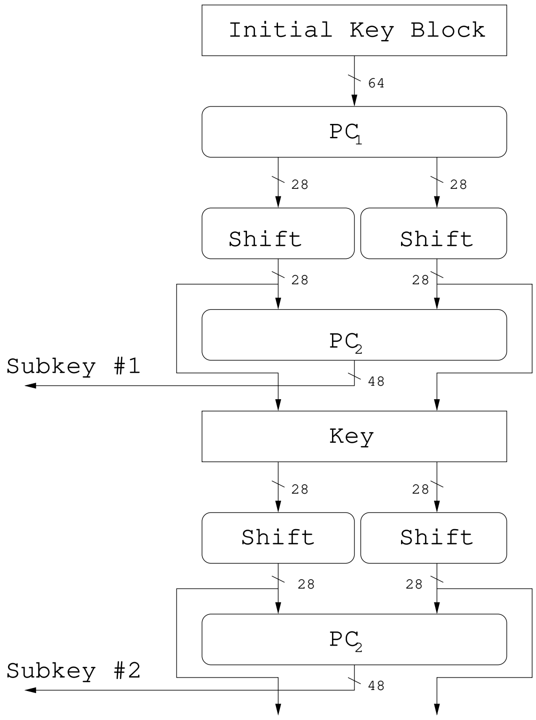

Figure 2: The key scheduling algorithm

1.2.2 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 $\mathcal{PC}_1$, 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 $\mathcal{PC}_2$.

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.

1.2.3 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 $\mathcal{EP}$, $\mathcal{PP}$, $\mathcal{PC}_1$, $\mathcal{PC}_2$, $\mathcal{IP}$, $\mathcal{IP}^{-1}$ and the parameters of the circular shifts in the key scheduling algorithm in several good books on cryptography ([1, 18]).

One round consists of the following operations: first, an expansion permutation, denoted $\mathcal{EP}$, expands the 32 bits of the right half of the data R_i to 48 bits, which are `XOR`ed 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 $\mathcal{EP}$ 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*.

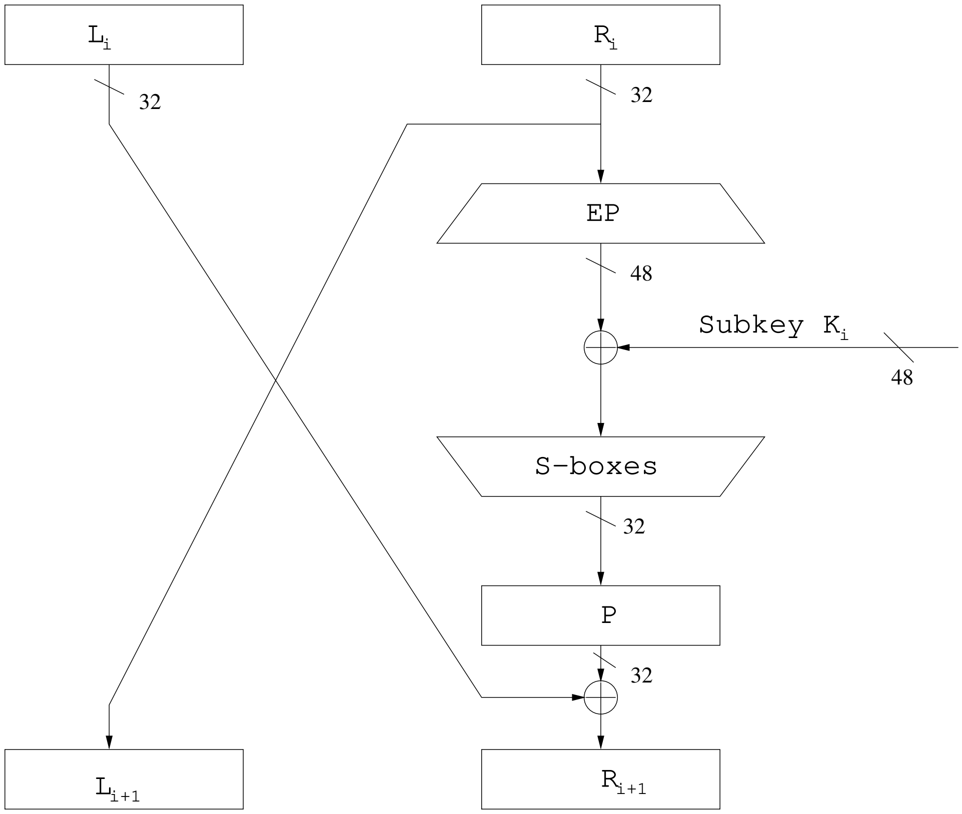

Figure 3: The f-function

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* $\mathcal{PP}$. 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 $\mathcal{PP}$ is `XOR`ed 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.

1.3 A 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 optimisation work and finally we present the speed measures of our routine.

1.3.1 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 $\mathcal{IP}$ 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 !).

1.3.2 The Bitslicing Concept

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

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

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

implementation:

- ⊕ 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.

1.3.3 The Pentium's MMX Architecture

Intel's MMX architecture has been designed with the improvements of multimedia and intensive floating-point arithmetic applications in mind. It achieves this goal by offering a new set of 8 dedicated 64-bits registers, a new instruction set which allows for data parallelisation and a superscalar architecture.

Together with the 8 64-bits wide registers, we have used only 6 logical instructions. They are listed in Figure 4. Unfortunately, there are no other logical instructions available, as it is sometimes the case on RISC architectures. Furthermore, one can note that the operations take two operands, and not three as it is the case sometimes. This means that one of the operands is destroyed during the operation; this reduces considerably the speed improvement possibilities. Another drawback is the absence of true `NOT` binary instruction. The only possibility is to combine this boolean operation with a `AND`, which is not always equivalent in terms of speed.

Instruction	Description
`EMMS`	maps the MMX registers on the CPU floating point unit
`MOVQ x, y`	moves a 64-bits word between reg/mem and mem/reg
`PXOR x, y`	binary XOR
`PAND x, y`	binary AND
`PANDN x, y`	binary NOT followed by an AND
`POR x, y`	binary OR

Figure 4: MMX instructions used in the bitsliced implementation

1.3.4 Cache Latency Optimization

In order to write a very fast DES routine, one have first to notice that one of the key features in the linear cryptanalysis is that the data that will be encrypted are produced by a pseudo-random generator just before the encryption begins, so we can take account of the so-called *temporal property* of our data. In other words, the data which have to be encrypted are concentrated in time in such a manner that probably all of them will be in the L1 cache (the one nearest to the CPU). This is typically not usual conditions for a DES routine to proceed.

Taking account of this temporal property allows us to use the new instructions of the Pentium III family for cache management and prefetching possibilities. The `prefetch` instruction is able to retrieve a minimum of 32 bytes of data prior to the data actually needed. This hides the latency for data access in the time required to process data already resident in the cache. This instruction does not change the user-visible semantics of a program, although it affects most of the time program's performances.

The result is that the cache is minimally polluted by data which are no more needed by the DES routine.

1.3.5 Optimization of DES

In order to have quickly a debugged and fast DES routine at disposal, we have proceeded as follows: first, we have implemented by hand the eight S-boxes in assembly with the goal of minimising the memory usage and optimising the use of the eight registers. This raw code has then been tested and optimised. In a next step, we have written a `Perl` script, a kind of DES compiler, which has produced the code of the whole routine using the raw code of the S-boxes. The hard-coding of all the permutations operations and the key scheduling algorithm have been handled by this script. A second optimisation has been done at this level, the goal being to prefetch in a

S-box data used in the next one.

In order to save the biggest number possible of instructions, the routine is fully unrolled, i.e. there are no `jmp` or `call` instructions *at all* in its core. This allows to avoid a lot of lost CPU cycles which are the consequences of bad branch predictions. One of the drawbacks of this manner of proceeding is that the resulting code size of the routine is very large.

Another optimisation method was to pair instructions such that the instruction decoding unit of the CPU is used at its maximal possible rate. The benefits of this technique are however quite small compared to the necessary work time; this is a consequence of the kind of instructions used in a bitsliced S-box implementation.

The optimisation that we have done are not the only possible ones. We had to find a compromise between the development time and the speed of the routine. We are convinced that it is possible to optimise more and more the S-boxes. In [11], it is for example shown that Kwan's S-boxes are not optimal for the Intel architecture. Unfortunately we got these results too late in our project schedule to make use of them. Furthermore, it seems that their implementation is better in case of a normal DES use, but we get better performance in our situation with our specific optimisations.

1.3.6 Performance Results

We give here the results of the performance results of our DES routine. The speed measurement procedure is exhaustively described in Annex B.

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 (not prefetched input data)

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. In or-

	not prefetched data	prefetched data
raw cycles number	19070	14893
cycles / encrypted block	298.0	232.7
encryption rate on a PIII 666MHz	143 Mbps	183 Mbps

Figure 6: Speed measures of the whole DES routine

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 optimised 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.

2 Theoretical Description of the Attack

In this chapter, we present in a first part the theoretical foundations of the linear cryptanalysis, and in the second part the algorithm which breaks DES in a more concrete way.

2.1 Introduction

Together with the differential cryptanalysis (see [4]), the linear cryptanalysis is one of the most famous generic attack against block ciphers. It was proposed by Matsui in [13], where he shows that DES can be broken with the help of 2^{47} known plaintext-ciphertext pairs faster than an exhaustive search.

Later, in a following paper [12], he refines his technique and shows that it is sufficient to have 2^{43} known plaintext-ciphertext pairs at disposal; furthermore, he implements it and breaks DES in 50 days with the help of 12 computers. Although this attack has only a theoretical importance, the linear cryptanalysis is the most powerful one on DES to date.

2.2 Linear Cryptanalysis Principles

Block ciphers commonly use non-linear operations in their schedule. In DES, the only non-linear stage is the S-boxes one. All the other operations are linear and can be easily analysed. In fact, the S-boxes have more features in common with a linear transformation than one would expect if they were chosen completely at random. Thus one can be convinced that the DES S-boxes are not optimised against linear cryptanalysis, as we will see it later.

For the theoretical aspect of the attack, we will use the Matsui notation for bit locations. Annex A gives exhaustive conversion tables between standard's and Matsui's way to denote bits in a word.

The principle of the linear cryptanalysis is very simple: one approximates the (non-linear) block cipher using a linear expression:

$$\left(\bigoplus_{i\in\{1...64\}} \mathcal{P}^{(i)}\right) \oplus \left(\bigoplus_{j\in\{1...64\}} \mathcal{C}^{(j)}\right) = \bigoplus_{k\in\{1...56\}} \mathcal{K}^{(k)} \tag{3}$$

where $\mathcal{P}$, $\mathcal{C}$ and $\mathcal{K}$ denote plaintext- ciphertext- and key-bits respectively and $\oplus$ the boolean operator `XOR`. The indices i, j and k denote *fixed* bit locations.

As the DES function is non-linear and is not a perfect cipher (i.e. the ciphertext is Dependants of the plaintext), equation (3) will hold with a given probability $p \neq \frac{1}{2}$ for randomly given plaintext P and the corresponding ciphertext C. The magnitude

$$\epsilon = \left| p - \frac{1}{2} \right| \tag{4}$$

represents the effectiveness of the linear expression (3).

One of the goals of linear cryptanalysis is to find the best linear expression, i.e. the linear expression which holds with the bigger bias ϵ. We will not treat this problem in this thesis, and we will take the ones given by Matsui in his papers (see [14] for more details about searching good linear approximations).

2.2.1 Getting One Bit of Information about the Key

Given an effective linear approximation, it is possible to determine *one* bit of information about the key

$$\bigoplus_{k \in \{1 \dots 56\}} \mathcal{K}^{(k)}$$

with the help of Algorithm 2.1, the indices k being fixed by the linear expression; its core is a maximum-likelihood method. It is quite easy to see

Algorithm 2.1 Determination of one key bit

$T :=$ # of plaintexts (out of N) such that the left side of (3) is equal to 0.
`IF` $T > \frac{N}{2}$
`THEN` guess $\bigoplus \mathcal{K}^{(k)} = 0$ (when $p > \frac{1}{2}$) or 1 (otherwise)
`ELSE` guess $\bigoplus \mathcal{K}^{(k)} = 1$ (when $p > \frac{1}{2}$) or 0 (otherwise)
`END`

that the success rate increases when N (the number of plaintext-ciphertext pairs) or $\epsilon = \left| p - \frac{1}{2} \right|$ does.

Lemma 2.1
Let p_s be the success probability of Algorithm 2.1. Then, p_s increases with N and ϵ.

Proof :
Let's assume that the probability that the linear expression holds is $p = \frac{1}{2} + \epsilon$, with $\epsilon > 0$ and that $\bigoplus \mathcal{K}^{(k)} = 0$. Let the random variable T be the sum of N identically distributed and mutually independent random variables X_i, which have the following distribution:

x	$P_X[X=x]$
0	$\frac{1}{2}+\epsilon$
1	$\frac{1}{2}-\epsilon$

One can compute easily the expected value of T $E[T] = N(\frac{1}{2}-\epsilon)$, and its variance $Var[T] = N(\frac{1}{4}-\epsilon^2)$. Applying Chebyshev inequality to the failure probability $p_f = 1 - p_s$ (i.e. the probability that the sum deviates from its expected value from more than $N\epsilon$), we get the following bound:

$$\begin{aligned} 1 - p_s \leq P_T\left[|T - E\left[T\right]| \geq N\epsilon\right] &\leq \frac{Var\left[T\right]}{N^2\epsilon^2} \\ &= \frac{1}{N}\left(\frac{1}{4\epsilon^2} - 1\right) \end{aligned}$$

It is then clear that the success probability p_s increases when N or ϵ (or both) do.

$\diamondsuit$

2.2.2 Getting Multiple Bits of Information about the Key

For a practical known-plaintext attack on DES, i.e. an attack which gives more than one bit of information about the key, Matsui makes use of the most effective $n-1$-rounds linear approximation. In other words, one deciphers the final round using the subkey candidates $K_{16}^{(i)}$, building a linear approximation accepting one term of f-function in its core.

Consequently, we obtain a slightly different expression than (3), which holds with the best bias of $n-1$-rounds DES.

$$\begin{aligned} &\left(\bigoplus_{i\in\{1...64\}} \mathcal{P}^{(i)}\right) \oplus \left(\bigoplus_{j\in\{1...64\}} \mathcal{C}^{(j)}\right) \oplus \left(\bigoplus_{m\in\{1...32\}} f(\mathcal{C},\mathcal{K}_{16})^{(m)}\right) \\ = &\bigoplus_{k\in\{1...56\}} \mathcal{K}^{(k)} \end{aligned} \qquad (5)$$

If one substitutes an incorrect subkey candidate $\mathcal{K}_{16}$, the effectiveness of the previous linear expression will decrease. This fact is stated by the *wrong key randomisation hypothesis* (see Figure 7). The probability p_G that the linear approximation holds if we decrypt the final round with the good subkey is bigger than the probability p_W where the round is decrypted by wrong subkey candidates.

$$\frac{\left|p_G - \frac{1}{2}\right|}{\left|p_W - \frac{1}{2}\right|} \gg 1 \tag{6}$$

Figure 7: Wrong key randomisation hypothesis

This can be intuitively understood: one round of decryption with a wrong subkey candidate can be seen as one round more of encryption; thus, the plaintext and its corresponding ciphertext will be less dependent and the linear expression will be less biased.

Algorithm 2.2 implements a maximum-likelihood method for the modified linear approximation. This algorithm retrieves with a certain probability of success i bits which are coming from the good subkey candidate and one bit of information about the sum of some key bits which builds the right part of the linear expression. With the help of this algorithm, Matsui shows in [13]

Algorithm 2.2 Determination of multiple key bits

`FOREACH` subkey candidate $\mathcal{K}^i$ of $\mathcal{K}$ `DO`

$T_i := \#$ plaintexts (out of N) such that the left side of the linear approximation is equal to 0.

`END`

$T_{max} := \max\{T_i\}$

$T_{min} := \min\{T_i\}$

`IF` $|T_{max} - \frac{N}{2}| > |T_{min} - \frac{N}{2}|$ `THEN`

adopt the subkey candidate corresponding to T_{max} and guess $\bigoplus \mathcal{K}^{(k)} = 0$ (when $p > \frac{1}{2}$) or 1 (otherwise) `END`

`IF` $|T_{max} - \frac{N}{2}| < |T_{min} - \frac{N}{2}|$ `THEN`

adopt the subkey candidate corresponding to T_{min} and guess $\bigoplus \mathcal{K}^{(k)} = 1$ (when $p > \frac{1}{2}$) or 0 (otherwise) `END`

how it is possible to break DES using 2^{47} known-plaintexts and two different linear approximations applied each on one S-box. This method retrieves 14 bits of the key, the 42 remaining having to be found using an exhaustive search.

With 2^{47} known plaintext-ciphertext pairs at disposal, Matsui estimates the success probability of his attack to be 97.7 % with a complexity of 2^{42} DES evaluations for the exhaustive key search part.

2.3 The 16-Rounds DES Attack

In order to break 16-rounds DES, Matsui shows in [12] how it is possible to improve the attack described previously. First of all, he works with linear expression on 14 rounds of DES and no more on 15 rounds; each equation has two active S-boxes (S-box 1 and S-box 5) and can recover each 13 bits of the key, or 26 in total.

2.3.1 The Best 14-rounds Linear Approximations

Matsui has found the two following best 14-rounds linear expression which are the central point in the attack. The first one sounds

$$\begin{aligned}&\mathcal{P}_L^{(7)} \oplus \mathcal{P}_L^{(18)} \oplus \mathcal{P}_L^{(24)} \oplus \mathcal{C}_H^{(7)} \oplus \mathcal{C}_H^{(18)} \oplus \mathcal{C}_H^{(24)} \oplus \mathcal{C}_H^{(29)} \oplus \mathcal{C}_L^{(15)} = \\ &\mathcal{K}_2^{(22)} \oplus \mathcal{K}_3^{(44)} \oplus \mathcal{K}_4^{(22)} \oplus \mathcal{K}_6^{(22)} \oplus \mathcal{K}_7^{(44)} \oplus \mathcal{K}_8^{(22)} \oplus \mathcal{K}_{10}^{(22)} \oplus \\ &\mathcal{K}_{11}^{(44)} \oplus \mathcal{K}_{12}^{(22)} \oplus \mathcal{K}_{14}^{(22)}\end{aligned} \tag{7}$$

and the second one

$$\begin{aligned}&\mathcal{C}_L^{(7)} \oplus \mathcal{C}_L^{(18)} \oplus \mathcal{C}_L^{(24)} \oplus \mathcal{P}_H^{(7)} \oplus \mathcal{P}_H^{(18)} \oplus \mathcal{P}_H^{(24)} \oplus \mathcal{P}_H^{(29)} \oplus \mathcal{P}_L^{(15)} = \\ &\mathcal{K}_{13}^{(22)} \oplus \mathcal{K}_{12}^{(44)} \oplus \mathcal{K}_{11}^{(22)} \oplus \mathcal{K}_9^{(22)} \oplus \mathcal{K}_8^{(44)} \oplus \mathcal{K}_7^{(22)} \oplus \mathcal{K}_5^{(22)} \oplus \\ &\mathcal{K}_4^{(44)} \oplus \mathcal{K}_3^{(22)} \oplus \mathcal{K}_1^{(22)}\end{aligned} \tag{8}$$

They both hold with an approximate probability of $\frac{1}{2} - 1.19 \cdot 2^{-21}$.

If we apply these equations to 14 consecutive f-functions, from the 2^{nd} to the 15^{th} round of DES, we have the two following final linear approximations:

$$\begin{aligned}&\mathcal{P}_H^{(7)} \oplus \mathcal{P}_H^{(18)} \oplus \mathcal{P}_H^{(24)} \oplus \mathcal{C}_H^{(15)} \oplus \mathcal{C}_L^{(7)} \oplus \mathcal{C}_L^{(18)} \oplus \mathcal{C}_L^{(24)} \oplus \mathcal{C}_L^{(29)} \oplus \\ &f(\mathcal{C}_L, \mathcal{K}_{16})^{(15)} \oplus f(\mathcal{P}_L, \mathcal{K}_1)^{(7)} \oplus f(\mathcal{P}_L, \mathcal{K}_1)^{(18)} \oplus f(\mathcal{P}_L, \mathcal{K}_1)^{(24)} = \\ &\mathcal{K}_3^{(22)} \oplus \mathcal{K}_4^{(44)} \oplus \mathcal{K}_5^{(22)} \oplus \mathcal{K}_7^{(22)} \oplus \mathcal{K}_8^{(44)} \oplus \mathcal{K}_9^{(22)} \oplus \mathcal{K}_{11}^{(22)} \oplus \\ &\mathcal{K}_{12}^{(44)} \oplus \mathcal{K}_{13}^{(22)} \oplus \mathcal{K}_{15}^{(22)}\end{aligned} \tag{9}$$

and

$$\begin{aligned}&\mathcal{C}_H^{(7)} \oplus \mathcal{C}_H^{(18)} \oplus \mathcal{C}_H^{(24)} \oplus \mathcal{P}_H^{(15)} \oplus \mathcal{P}_L^{(7)} \oplus \mathcal{P}_L^{(18)} \oplus \mathcal{P}_L^{(24)} \oplus \mathcal{P}_L^{(29)} \oplus \\ &f(\mathcal{P}_L, \mathcal{K}_1)^{(15)} \oplus f(\mathcal{C}_L, \mathcal{K}_{16})^{(7)} \oplus f(\mathcal{C}_L, \mathcal{K}_{16})^{(18)} \oplus f(\mathcal{C}_L, \mathcal{K}_{16})^{(24)} = \\ &\mathcal{K}_{14}^{(22)} \oplus \mathcal{K}_{13}^{(44)} \oplus \mathcal{K}_{12}^{(22)} \oplus \mathcal{K}_{10}^{(22)} \oplus \mathcal{K}_9^{(44)} \oplus \mathcal{K}_8^{(22)} \oplus \mathcal{K}_6^{(22)} \oplus \\ &\mathcal{K}_5^{(44)} \oplus \mathcal{K}_4^{(22)} \oplus \mathcal{K}_2^{(22)}\end{aligned} \tag{10}$$

We note that one decrypts *two* rounds instead of a single one. One can expect that this fact will amplify the randomisation effect in case of wrong subkey candidates.

Let's define now the concept of *effective text bits* and of *effective key bits*. They are simply the bits which affect the left part of the linear approximations. We count the XORed value of several text- or key-bits affecting the left side of our expressions as one effective bit.

The effective bits of the first linear expression are the following:

$\mathcal{P}_L^{(11)}$, $\mathcal{P}_L^{(12)}$, $\mathcal{P}_L^{(13)}$, $\mathcal{P}_L^{(14)}$, $\mathcal{P}_L^{(15)}$, $\mathcal{P}_L^{(16)}$, $\mathcal{C}_L^{(0)}$, $\mathcal{C}_L^{(27)}$, $\mathcal{C}_L^{(28)}$, $\mathcal{C}_L^{(29)}$, $\mathcal{C}_L^{(30)}$, $\mathcal{C}_L^{(31)}$, $\mathcal{P}_H^{(7)} \oplus \mathcal{P}_H^{(18)} \oplus \mathcal{P}_H^{(24)} \oplus \mathcal{C}_L^{(7)} \oplus \mathcal{C}_L^{(18)} \oplus \mathcal{C}_L^{(24)} \oplus \mathcal{C}_L^{(29)} \oplus \mathcal{C}_H^{(15)}$, $\mathcal{K}_1^{(18)}$, $\mathcal{K}_1^{(19)}$, $\mathcal{K}_1^{(20)}$, $\mathcal{K}_1^{(21)}$, $\mathcal{K}_1^{(22)}$, $\mathcal{K}_1^{(23)}$, $\mathcal{K}_{16}^{(42)}$, $\mathcal{K}_{16}^{(43)}$, $\mathcal{K}_{16}^{(44)}$, $\mathcal{K}_{16}^{(45)}$, $\mathcal{K}_{16}^{(46)}$ and $\mathcal{K}_{16}^{(47)}$.

and the ones in the second approximation:

$\mathcal{C}_L^{(11)}$, $\mathcal{C}_L^{(12)}$, $\mathcal{C}_L^{(13)}$, $\mathcal{C}_L^{(14)}$, $\mathcal{C}_L^{(15)}$, $\mathcal{C}_L^{(16)}$, $\mathcal{P}_L^{(0)}$, $\mathcal{P}_L^{(27)}$, $\mathcal{P}_L^{(28)}$, $\mathcal{P}_L^{(29)}$, $\mathcal{P}_L^{(30)}$, $\mathcal{P}_L^{(31)}$, $\mathcal{C}_H^{(7)} \oplus \mathcal{C}_H^{(18)} \oplus \mathcal{C}_H^{(24)} \oplus \mathcal{P}_L^{(7)} \oplus \mathcal{P}_L^{(18)} \oplus \mathcal{P}_L^{(24)} \oplus \mathcal{P}_L^{(29)} \oplus \mathcal{P}_H^{(15)}$, $\mathcal{K}_{16}^{(18)}$, $\mathcal{K}_{16}^{(19)}$, $\mathcal{K}_{16}^{(20)}$, $\mathcal{K}_{16}^{(21)}$, $\mathcal{K}_{16}^{(22)}$, $\mathcal{K}_{16}^{(23)}$, $\mathcal{K}_1^{(42)}$, $\mathcal{K}_1^{(43)}$, $\mathcal{K}_1^{(44)}$, $\mathcal{K}_1^{(45)}$, $\mathcal{K}_1^{(46)}$ and $\mathcal{K}_1^{(47)}$.

We can note that 13 text-bits can be used to derive 12 key-bits and the bit of the right side in each equation. We obtain hence a total of 26 (fortunately not duplicated) secret key-bits from the both equations, using 26 bits of text.

2.3.2 An Improved Algorithm

We are now close to be able to give the whole algorithm for breaking DES. For each subkey candidate and for both linear approximations, we count the number of times that the left side of the linear expressions is equal to 0. Then, the resulting value of the counters must reflect the reliability of the corresponding subkey candidate.

In other words, we will get a list of subkey candidates for each linear expression, and it is possible to sort these lists, the most likely 13-bits subkey candidate (i.e. the one which produces the biggest bias in the linear approximation) being in first position, the least likely one being at the end of a list.

We have then to combine these two lists, in order to provide a sorted list of 26-bits subkey candidates. The exhaustive search will then occur from the most likely candidate to the least likely one. This allows to reduce the number of known plaintext-ciphertext pairs from 2^{47} to 2^{43}. A formal description of this process is described in Algorithm 2.3.

Algorithm 2.3 Breaking DES

Prepare 2^{13} counters for each linear expression denoted thereafter $C_1^{(i)}$ and $C_2^{(i)}$ with $0 \leq i \leq 2^{13}$. Initialise them to 0.
`/*` Each index i corresponds to the state of 13 effective text bits. `*/`
`FOR` 2^{43} plaintext-ciphertext pairs (p, c) `DO`
Compute the values i_1 and i_2 using p and c for linear expressions l_1 and l_2.
Increment by one $C_1^{(i_1)}$ and $C_2^{(i_2)}$.
`END`
Prepare 2^{12} counters for each linear expression denoted thereafter $K_1^{(k)}$ and $K_2^{(k)}$ with $0 \leq k \leq 2^{12}$.
`/*` Each counter corresponds to the state of 12 effective key bits. `*/`
`FOREACH` k_1, k_2 `DO`
$K_1^{(k_1)} := \sum \left\{ C_1^{(x)} \quad \text{such that} \quad l_1(p_x, c_x, k_1) = 0 \right\}$.
$K_2^{(k_2)} := \sum \left\{ C_2^{(x)} \quad \text{such that} \quad l_2(p_x, c_x, k_2) = 0 \right\}$.
`END`
Sort K_1's and K_2's by decreasing magnitude $|K_x^{(y)} - 2^{42}|$ with $x \in \{1, 2\}$ and $0 \leq y \leq 2^{12}$. We get two lists of 2^{12} counters each that we denote $S_1^{(r)}$ and $S_2^{(r)}$ with $0 \leq r \leq 2^{12}$.
`/*` Guess of the last bit of information for each subkey candidate `*/`
`FOREACH` $S_x^{(r)}$ `DO`
`IF` $|S_x^{(r)} - 2^{42}| \leq 0$ `THEN` guess that right side of linear expression x is 0. `END`
`IF` $|S_x^{(r)} - 2^{42}| > 0$ `THEN` guess that right side of linear expression x is 1. `END`
`END`
`/*` Combination of the two lists `*/`
Let $f(r_1, r_2) := (r_1 + 1) \cdot (r_2 + 1)$, where r_x is the index corresponding to the sorted list x.
Build the final ranking of the subkey candidates by combining the two lists S_1 and S_2 by increasing $f(.)$-value. We denote this final list by F.
`FOREACH` subkey candidate in F `DO`
Search the remaining 30 bits.
`IF` good key found `THEN EXIT END`
`END`

2.3.3 Comparison Between the Two Attacks

While there is only one exhaustive search of 42 bits in the first version of the attack, which should be successful with a very high probability, the improved one makes several searches of lesser complexity (search for 30 bits), but with a lower, decreasing success probability for each successive candidate. Matsui estimates that, if 2^{43} known plaintext-ciphertexts pairs are at disposal, the success probability is 85 % with a complexity of 2^{43} DES computations (i.e. 2^{13} exhaustive searches on 30 bits). The main advantage of the second attack is its modest need in known plaintext-ciphertext pairs.

3 A Practical Implementation of the Attack

In a first part, we describe the process implementing the attack in a general manner, then more specific points are treated, like the problem of the pseudo-random bytes generation, the collecting phase of the statistical material, and the management of the remote processes in a pseudo-cluster.

3.1 Introduction and Generalities

As described in the previous chapter, the goal of the linear cryptanalysis is to evaluate in a probabilistic way a boolean linear expression which have, given a uniform distributed input, an output distribution slightly disbalanced compared to the one of an hypothetical perfect block cipher, which has to be uniformally distributed.

In order to achieve this goal, one have to produce a big number of random plaintext blocks (in our case, $2^{43} \approx 8.8 \cdot 10^{12}$ blocks of 64 bits), to encrypt them with the secret key, and to compute the number of specific situations which involve the so-called effective bits, i.e., the bits which play an active role in the linear expression.

So we need a fast pseudo-random byte generator which has acceptable statistical properties, to collect the statistical information for the plaintext, to feed the fast DES encryption routine described in a previous chapter with this plaintext, and finally to collect the statistical information concerning the ciphertext. This process is summarised as Algorithm 3.1.

In order to have an acceptable process management granularity, we have decided to split the quantity of 2^{43} blocks in 2048 equal parts, which leads to a computing time of about 45 minutes per process on a Pentium III 666MHz. This will bring however some constraints in the choice and the implementation of the pseudo-random generator.

The input parameters of the process are the following:

- A true random seed for the pseudo-random generator: 1024 bytes generated by the very good entropy source `/dev/random` of the Linux operating system. They are furnished to the process in a binary file.
- The key, as a 7-bytes binary file.

The aim of the whole process is to select the good counters which will be incremented; this is implemented by the computation of the corresponding offset in a counters array. First of all, the process loads the input parameters, inits all the counters, seeds the pseudo-random generator and fills the

Algorithm 3.1 The whole process

```
m_64 plaintext[64];
m_64 key[56];
m_64 ciphertext[64];
m_64 seed[128];

m_32 counters_eq1[8192];
m_32 counters_eq2[8192];
m_32 offsets_eq1[64];
m_32 offsets_eq2[64];

load_key (key);
load_true_random_seed (seed);

/* 64 encryptions / iteration                    */
/* 67108864 iterations / process                 */
/* 2048 processes  => 2^43 blocks                */
for (i = 0; i < 67108864; ++i) {
  generate_4K_pseudo_random_bytes (plaintext);
  collect_plaintext_stats (plaintext, offsets_eq1);
  collect_plaintext_stats (plaintext, offsets_eq2);
  DES_encrypt (plaintext, key, ciphertext);
  collect_plaintext_stats (ciphertext, offsets_eq1);
  collect_plaintext_stats (ciphertext, offsets_eq2);
  update_counters (counters_eq1, offsets_eq1);
  update_counters (counters_eq2, offsets_eq2);
}

store_counters (counters_eq1);
store_counters (counters_eq2);
```

64 plaintext blocks of 64 bits each (`plaintext`) with its output. Then, it collect the first part of the statistical material, by computing the first half of the offsets, which are stored in `offsets_eqx[]`. Each offset represents the state of 13 bits, therefore a 32 bits word is sufficient for the storage. The next operation is the encryption of the plaintext, which is done by the fast DES routine described in a previous chapter. Then, the missing statistical material is collected, the second half of the offsets is computed and in the last part, the counters pointed by the 64 offsets are incremented by one. At the end of the process, i.e. after repeating this process a given number of time, the counters are stored in a binary file.

We can note that the expected value of each counter is approximately equal to $\frac{2^{32}}{8192} = 2^{19}$. A 32-bit counter is therefore sufficient. The total needed memory is hence about 540 KB per process.

3.2 Generating Huge Amounts of Pseudo-Random Blocks

As suggested sooner, one have to solve the problem of finding a very fast pseudo-random generator which can provide a stream of bits with very good statistical properties. We recall that one have to produce 2^{43} 64-bits blocks, which corresponds to 64 TB (2^{46} bytes) of data.

3.2.1 Linear Feedback Shift Registers

As suggested in Matsui's paper [12], we chose sequences $\{g^0, g^1, g^2, g^3, \ldots\}$ where g is a generator of the cyclic group $\mathrm{GF}(2^n)^*$ and n is a power of two. These sequences can be very well implemented using a Linear Feedback Shift Register (LFSR) and are convenient for parallel computing, as we will see it later.

A LFSR is made up of two parts (see Figure 8): a shift register and a feedback function. Each time a bit is needed, all of the bits in the register are shifted one position to the right, and the new leftmost bit is computed as a linear function (using only the `XOR` boolean function) of the other bits remaining in the register. A formal definition of its functioning is given in Definition 3.1.

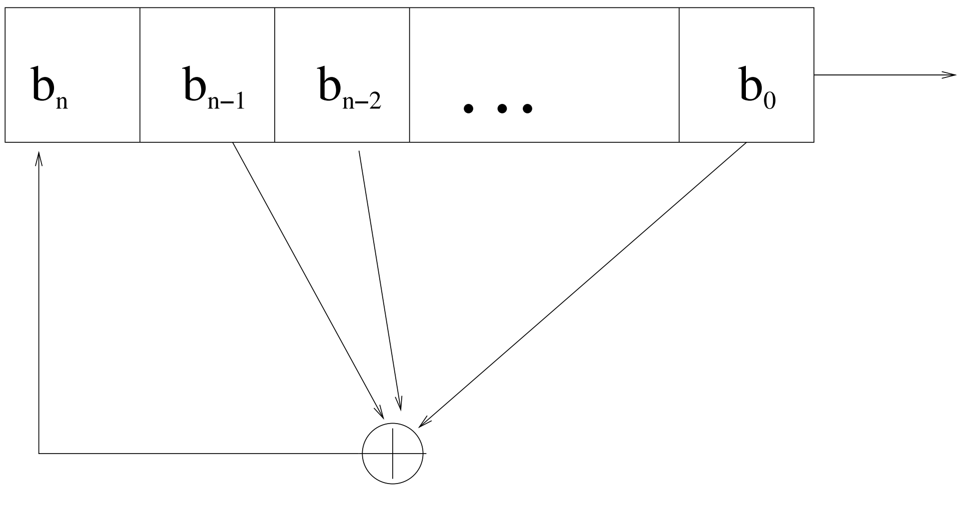

Figure 8: Linear Feedback Shift Register

Definition 3.1 (LFSR, [1])
A Linear Feedback Shift Register (LFSR) *of length n consists of n stages numbered from $n-1$ to 0, each capable of storing one bit and having one input and one output; and a clock which controls the movement of data. During each unit of time the following operations are performed:*

- *the content of stage 0 is output and forms part of the output sequence*
- *the content of stage i is moved to stage $i-1$ for each i, $1 \leq i \leq n-1$*
- *the new content of stage $n-1$ is the feedback bit b_{n-1} which is calculated by adding together modulo 2 the previous contents of a fixed subset of stages $n-1, \ldots, 1, 0$.*

It is easy to see that a n-bit LFSR can be in maximal $2^n - 1$ different internal states. This means that it can, in theory, generate a $2^n - 1$ 1-bit long pseudo-random sequence before repeating itself.

In fact, only LFSRs with certain linear feedback functions will produce such a pseudo-random sequence with a period of $2^n - 1$. One calls these LFSRs *maximal-period* LFSRs. In order to study their properties, one defines first the concept of *connection polynomial*:

Definition 3.2 (Connection Polynomial of a LFSR)
The connection polynomial *of a LFSR is the polynomial $C(D) = 1 + c_1 D + c_2 D^2 + ... + c_n D^n \in \mathbb{Z}_2[D]$, where the coefficient c_i have the value 1 if the corresponding state is taken in the feedback function and 0 otherwise.*

The following Theorem states that a LFSR has a maximal period for a certain kind of polynomial.

Theorem 3.1
The output sequence of a linear feedback shift register on a nonzero initial state has maximum period if and only if its connection polynomial is primitive.

Lists of primitive polynomial are available in several good book about cryptography. By choosing a big enough n and a corresponding primitive polynomial, one can be sure that no similar sequences will be produced by such a pseudo-random generator, because of the period's length.

Fortunately, LFSR have good statistical properties, which states the Theorem 3.2.

Theorem 3.2
If a LFSR with n registers has maximum period $2^n - 1$, then any output sequence of length $2^n - 1$ has following properties:

- *it contains exactly $2^{n-1} - 1$ zeroes and 2^{n-1} ones*
- *for any t, with $1 \leq t \leq n-2$, it contains 2^{n-t-2} blocks of length t and the same number of gaps of length t.*

Furthermore, it is possible to show that the bit sequence of a LFSR exhibits a periodic autocorrelation $\phi(n) = m$ for $m = 0, \pm n, \pm 2n, ...$ and $\phi(n) = -1$ for all other shifts. This impulse-like autocorrelation implies that the power spectrum is nearly white and, hence, the sequence resembles white noise.

3.2.2 Choice of the Primitive Polynomial's Degree

It is desirable that the management of the processes is as comfortable as possible. In order to achieve this goal, we have split the 2^{43}-blocks work in a fine granularity, more precisely in 2048 partial jobs, each producing 64 sequences at a time in parallel. In [12], Matsui uses the cyclic group $\mathrm{GF}\left(2^{64}\right)^*$ for generating the random plaintexts. The sequence has a period of $2^{64} - 1 \approx 1.8 \cdot 10^{19}$.

Knowing the effect of the birthday paradox, it is now interesting to compute the probability of generating two same sequences by seeding the generator with the same value. In the following of the section, we assume that the seed values are uniformally distributed. In practice, we take the output of `/dev/random` under Linux, which is considered to be a cryptographic secure pseudo-random bytes generator. Lemma 3.1 gives an approximation formula for this probability.

Lemma 3.1
Let $x \in_U GF(2^n)^*$ be the seed value of a sequence generated by a maximal-period LFSR. Let $m \leq 2^n - 1$ be the number of generated sequences. Then, the probability p_c of generating two or more identical sequences is upper bounded by

$$p_c \leq 1 - \left(1 - \frac{m-1}{2^n - 1}\right)^{m-1} \tag{11}$$

Proof :
From Theorem 3.2, we know that the period of the sequence generated by a maximal-period LFSR is equal to $2^n - 1$. One can see the seeding operation as choosing a random sample of size m in a set of $2^n - 1$ seeds *with* replacement. We assume that all arrangements have equal probability,

thus we can conclude that the probability of *no* repetition in our sample is equal to

$$
\begin{aligned}
& \frac{2^n-2}{2^n-1} \cdot \frac{2^n-3}{2^n-1} \cdot \ldots \cdot \frac{2^n-1-m+1}{2^n-1} \\
= & \prod_{i=1}^{m-1} \frac{2^n-1-i}{2^n-1} \\
= & \prod_{i=1}^{m-1} \left(1 - \frac{i}{2^n-1}\right) \\
\geq & \left(1 - \frac{m-1}{2^n-1}\right)^{m-1}
\end{aligned}
$$

and thus the probability of having one or more identical sequences is upper bounded by

$$p_c \leq 1 - \left(1 - \frac{m-1}{2^n-1}\right)^{m-1}$$

$\diamondsuit$

We give in Figure 9 values which are in our interest domain: we assume that an experiment needs $2048 \cdot 64 = 131072$ sequences. It is easier to evaluate numerically this collision probability if we express the previous bound as follows:

$$p_c \leq 1 - e^{(m-1)\ln\left(1-\frac{m-1}{2^n-1}\right)} \tag{12}$$

n	p_c
32	0.98
64	$9.3 \cdot 10^{-10}$
128	$5.1 \cdot 10^{-29}$
256	$1.5 \cdot 10^{-67}$

Figure 9: Some numerical approximations for p_c and $m = 131072$ sequences

One cannot take the risk of generating two or more times the same amount of random plaintexts if we want accurate statistical results. If we study Figure 9, we notice that a generator with a period of $2^{32} - 1$ is totally not useful for our purposes, which is an intuitive fact. The probabilities in the other cases are quite small, so we can forgive the risk of generating two or more identical sequences.

But what we cannot avoid with small n's are *overlapping* sequences, or in other words, sequences which have a common part with other sequences. Obviously, such an event can bring some bias in the results. We give in Lemma 3.2 an approximation mean for this event.

Lemma 3.2
Let $x \in_U GF(2^n)^*$ be the seed value of a sequence of length 2^l generated by a maximal-period LFSR. Let be $\alpha = \left\lfloor \frac{2^n-1}{3\cdot 2^l-2} \right\rfloor$ and let $m < \alpha$ be the number of generated sequences. Then the probability p_o of generating two sequences which are overlapping (i.e. have one or more elements in common) is upper bounded by

$$p_o \leq 1 - \left(1 - \frac{m-1}{\alpha}\right)^{m-1} \tag{13}$$

Proof :
We consider a sequence somewhere in the cycle of length $2^n - 1$ as illustrated in Figure 10. If we reserve space with a length of $2^l - 1$ at the left and at the right of the sequence, any start point which is not in such an enhanced interval will produce a non-overlapping sequence. So we divide the whole period in α buckets of $3 \cdot 2^l - 2$ points. Then we apply the same reasoning as in the proof of Lemma 3.1.

$\diamondsuit$

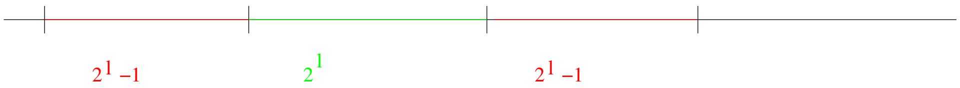

Figure 10: Reserving space around a sequence

If we have to produce 2^{43} random plaintexts blocks of 64 bits each in 131072 sequences, each sequence will have a length $l = 2^{32}$. Figure 11 furnishes approximation values for bound (13).

We conclude that a LFSR with a period of $2^{64} - 1$ is not adapted to our purposes because of the huge probability of collision between the different sequences in our case of granularity. The difference of the overlapping probabilities between LFSR with a size of 128 and 256 bits is not significant, so we propose, for speed purposes, to take the following primitive polynomial as a feedback function:

$$\boxed{D^{128} + D^7 + D^2 + D + 1 \quad \in \mathbb{Z}_2[D]} \tag{14}$$

n	$\mathcal{P}_2$
64	0.99
128	$6.5 \cdot 10^{-18}$
256	$1.9 \cdot 10^{-57}$

Figure 11: Some numerical approximations for p_o with $m = 131072$ sequences

3.2.3 Efficient Implementation of a LFSR

It is not difficult to see that LFSRs are quite inefficient in software. They produce only one bit at a time. But it is possible to parallelise the operations and to run several LFSRs at the same time using the SIMD (Single Instruction Multiple Data) concept. The Intel Pentium MMX Technology provides 64-bits wide registers, which can be used to implement 64 LFSR running in parallel. Algorithm 3.2 describes our method in a `C`-like notation.

Algorithm 3.2 64 LFSRs running in parallel

```
m_64 regs[N];          /* m_64 type is 64-bit wide type      */
unsigned int offset; /* The floating pointer                */
m_64 tmp;              /* A working register                  */
m_64 output;           /* The 64 output bits                  */

tmp      = 0;
tmp     ^= (regs + offset);  /* tmp ^= bit128               */
offset  += 0x3C8;
offset  &= 0x3FF;            /* offset = &regs[121]         */
tmp     ^= (regs + offset);  /* tmp ^= bit7                 */
offset  += 0x28;
offset  &= 0x3FF;            /* offset = &regs[126]         */
tmp     ^= (regs + offset);  /* tmp ^= bit2                 */
offset  += 0x8;
offset  &= 0x3FF;            /* offset = &regs[127]         */
tmp     ^= (regs + offset);  /* tmp ^= bit1                 */
offset  += 0x3F8;            /* Compute new offset          */
offset  &= 0x3FF;            /* offset = &regs[0]           */
output  = (regs + offset);   /* shift the regs !            */
*(regs+offset) = tmp;        /* set the new bit             */
```

Basically, it needs a circular buffer of 128 64-bits words of memory, which represents the registers and a floating pointer which stores the actual start of the circular buffer. By choosing n as a power of 2, one can furthermore implement the modulo operation, needed to update the floating offset, very efficiently with a single `AND` operation.

Our routine, exclusively implemented in assembly language, needs 234 clock cycles for producing 64 pseudo-random bits (i.e 3.7 cycles/bit) on an Intel Pentium III processor, which leads to rates of 182 Mbps and of 137 Mbps for clock rates of 666 MHz and 500 MHz, respectively [1].

3.3 Implementation Specific Aspects

We discuss here some more specific aspects of our implementation.

3.3.1 Feeding the Encryption Routine with Pseudo-Random Blocks

We use the output of the pseudo-random byte generator described in the previous section to feed the encryption part of the process. As described in a previous chapter, we need 64 blocks of 64 bits each, i.e. 4 KB of random data. The LFSRs produce 64 pseudo-random bits at a time, which are used to feed the first bit of the 64 parallel plaintexts, then the next 64 bits are used to feed the second bit and so on. This is illustrated in Figure 12. The notation $b_i^{(j)}$ means "bit i of LFSR j", where we use the standard notation for bit numbering (see Annex A).

$$\begin{pmatrix} b_1^{(1)} & b_1^{(2)} & \dots & b_1^{(63)} & b_1^{(64)} \\ b_2^{(1)} & b_2^{(2)} & \dots & b_2^{(63)} & b_2^{(64)} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ b_{63}^{(1)} & b_{63}^{(2)} & \dots & b_{63}^{(63)} & b_{63}^{(64)} \\ b_{64}^{(1)} & b_{64}^{(2)} & \dots & b_{64}^{(63)} & b_{64}^{(64)} \end{pmatrix}$$

Figure 12: Feeding of the plaintext blocks by the LFSR

The first row corresponds to the first bit of plaintext, so we can note that each plaintext is in reality a 64 bits long sequence of a *single* LFSR.

[1] It is worth to note that one needs about the same time to produce the data and to encrypt them!

3.3.2 Collecting Statistical Properties

The other part of the process is the collecting of statistical properties of the plaintext. As described in a previous chapter, we need $2^{13} = 8192$ counters corresponding to the state of the 13 effective bits for each equation. This is done in a simple manner by Algorithm 3.3. For example, an offset of 1023,

Algorithm 3.3 Collecting the statistical properties

```
/* Set the offsets to 0                                   */
for (eq = 0; eq < 2; eq++) {
  for (offset = 0; offset < 64; offset++) {
     off_array[eq][offset] = 0;
  }
}

/* Compute the offsets                                    */
for (eq = 0; eq < 2; eq++) {
  foreach (effective  bit eff_bit[eq][i]) {
    tmp_eff_bit = eff_bit[eq][i];
    for (offset = 0; offset < 64; offset++) {
      off_array[eq][offset] <<= 1;
      off_array[eq][offset] |= (tmp_eff_bit & 0x1);
    }
    tmp_eff_bit >>= 1;
  }
}

/* Update of the counters                                 */
for (eq = 0; eq < 2; eq++) {
  for (offset = 0; offset < 64; offset++) {
    counters[eq][off_array[eq][offset]]++;
  }
}
```

which has as binary representation `0001111111111`, represents the situation where the first three effective plaintext bits have a value of `0` and the ten lasts a value of `1`. We recall that an offset depends on certain plaintext bits and on certain ciphertexts bits, which are listed in the previous chapter. If such an event occurs, then the corresponding counter is incremented by one.

3.4 Management of the Processes

The management of the processes is entirely done by two little `Perl` scripts, `logging_server.pl` and `process_launcher.pl`. The first one is running on the master machine, and it is responsible to log the messages sent by the remote machines. The second one, which is replicated on each remote computer, is responsible for getting the parameters, launching the process, supervising the functioning, collecting the results and informing the master about the functioning of the slave machine.

Figure 13 illustrates the management of the processes.

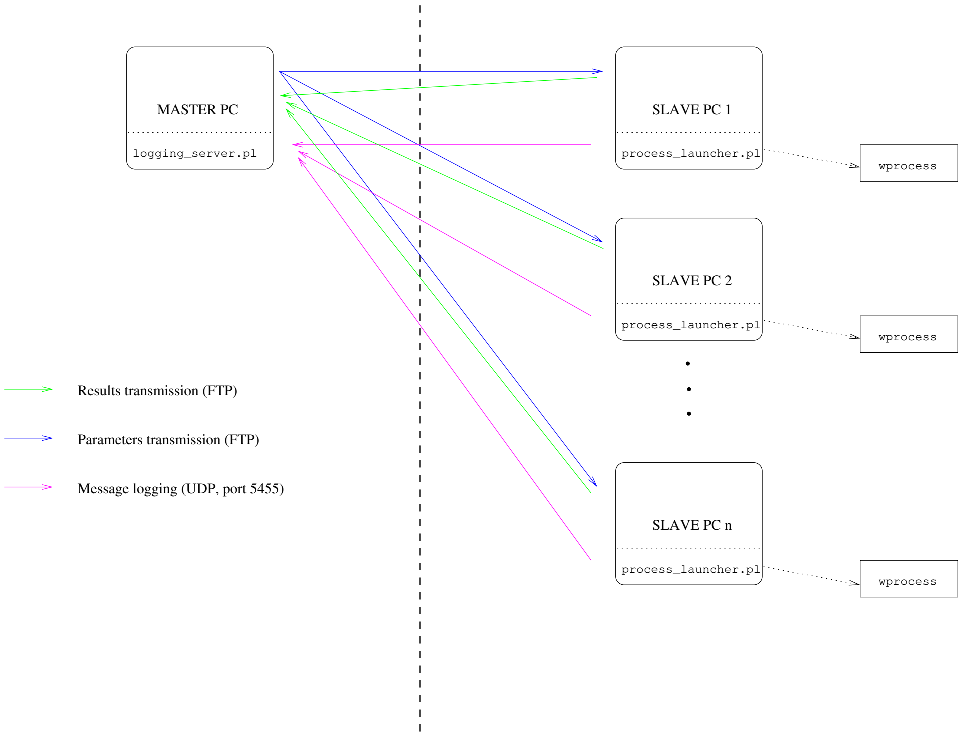

Figure 13: Management of the processes.

A first FTP connection between the master PC and a slave is responsible to transfer the two parameters files, i.e the key and the seeds; MD5 hash values are computed before and after the transfer, in order to be sure that the key and/or the seeds are not corrupted. At the end, another FTP connection is responsible to transfer the results to the master PC; the validity of the transfer is assured with the help of a MD5 hash values check.

During the computation, the `Perl` script based on the slave machine checks each minute if everything is going right and logs the results via an UDP connection (port 5455) to the master PC. Furthermore, when a process is terminated, it recovers the results, prepare the next seed, and launch a new process. All these activities are logged to the master, too.

The Perl script running on the remote machines is implemented as a finite state machine. Its functioning is summarised in Figure 14.

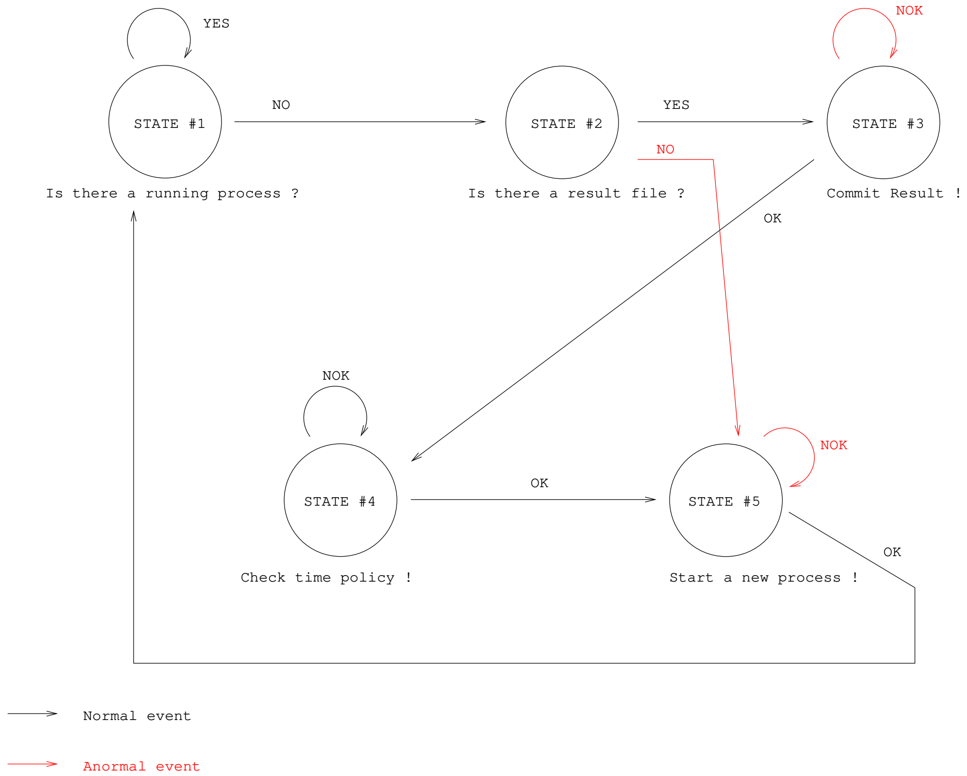

Figure 14: Finite state machine of process_launcher.pl

We can note that a state of the automata is responsible to manage the so-called time policy, which allows to use idle computing time during the night and the week-ends on machines which are unfortunately active during the days.

Using Perl allows to write quickly very powerful management scripts, hence we did not have to employ an existing cluster management package, which has considerably simplified our task.

4 Theoretical Considerations

We present is this chapter our own theoretical considerations about the success rate and the complexity of the attack.

In a first part, we recall some basic definitions from the probability theory; we propose then a way to compute analytically the maximal ranking probability of subkey candidates when they are sorted in a list by decreasing likelihood.

4.1 Some Mathematical Preliminaries

First of all, we recall the continuous density function and distribution function concepts:

Definition 4.1 (Density function / Distribution function)
A probability density on the line $\mathbb{R}^1$ *is a function* f *such that*

$$f(x) \geq 0 \quad \textit{and} \quad \int_{-\infty}^{+\infty} f(x)dx = 1 \tag{15}$$

To each density function f*, we let correspond its* distribution function F *defined by*

$$F(x) = \int_{-\infty}^{x} f(t)dt \tag{16}$$

We recall that as a continuous distribution function is meant a right continuous *non-decreasing* function with

$$\lim_{x \to -\infty} = 0$$

and

$$\lim_{x \to +\infty} = 1$$

The density function f is considered as an assignment of probabilities to the intervals of the line, the interval $[a, b]$ having probability

$$F(b) - F(a) = \int_{a}^{b} f(x)dx \tag{17}$$

Under this assignment an individual point carries probability zero. Useful values derived from continuous random variables are its expectation and variance.

Definition 4.2 (Expectation / Variance of a continuous RV)
The expectation $E[X]$ *of the random variable* X *having* f_X *as density function is given by*

$$E[X] = \int_{-\infty}^{+\infty} x f_X(x) dx \tag{18}$$

provided that the integral converges. The variance *of* X *is defined by*

$$Var[X] = E\left[(X - E[X])^2\right] = \int_{-\infty}^{+\infty} x^2 f_X(x) dx - E[X]^2 \tag{19}$$

A very important probability law is the normal one:

Definition 4.3 (The Normal Law)
Let X *be a* normal law *with mean* μ, $-\infty < \mu < +\infty$ *and standard deviation* σ, $\sigma > 0$. *Its density function* f_X *is denoted* $\phi_{(\mu,\sigma)}(x)$ *and is given by*

$$\phi_{(\mu,\sigma)}(x) := \frac{1}{\sigma\sqrt{2\pi}} \cdot e^{\frac{-(x-\mu)^2}{2\sigma^2}} \tag{20}$$

for $-\infty < x < +\infty$.

The distribution function F_X *of* $\phi_{(\mu,\sigma)}(x)$ *is denoted* $\Phi_{(\mu,\sigma)}(x)$ *and given by*

$$\Phi_{(\mu,\sigma)}(x) := \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{(z-\mu)^2}{2\sigma^2}} dz \tag{21}$$

for $-\infty < x < +\infty$.

The standard normal law *is a normal law with* $\mu = 0$ *and* $\sigma = 1$. *We denote its density function and its distribution function by* $\phi(x)$ *and* $\Phi(x)$, *respectively.*

The normal law plays a central role in the probability theory. One of the most famous results is Theorem 4.1, which states that the sum of identically distributed and mutually independent random variables is approximately normal. A proof of it can be found in [6], chapter *X*.

Theorem 4.1 (Central Limit Theorem)
Let X_i, *with* $1 \leq i \leq n$, *be a sequence of mutually independent random variables with a common distribution* X. *Suppose that* $\mu = E[X]$ *and* $\sigma^2 = Var[X]$ *exist and let* $S_n = X_1 + \cdots + X_n$. *Then, for every fixed* β

$$\lim_{n\to\infty} P\left[\frac{S_n - n\mu}{\sigma\sqrt{n}} < \beta\right] = \Phi(\beta) \tag{22}$$

4.2 Success Probability of the Attack

Let's have a closer look at the linear cryptanalysis process. Basically, we produce a huge number N, typically 2^{43}, plaintext-ciphertext pairs, and we feed a linear expression with these pairs. If the linear expression's resulting value is equal to 0, then the counter corresponding to the plaintext-ciphertext pair is incremented. This is done for each subkey candidate.

Following the wrong key randomisation hypothesis, the good subkey candidate will bias its counter in a more sensible way than most of the wrong ones. The goal of the linear cryptanalysis is hence to combine these two lists in order to get a final ranking of subkey candidates sorted by decreasing likelihood and finally to find out the right subkey candidate from the other ones by searching exhaustively the remaining bits for the most likely candidate first, and then for the other ones, following this "likelihood-order".

4.2.1 Modelling the Statistical Experiment

We denote in the following the counter corresponding to the right subkey candidate by the discrete random variable $C^{(r)}$ and the ones corresponding to wrong subkey candidates by the discrete random variables $C_i^{(w)}$ with $1 \leq i \leq 4095$. All these random variables are defined over the positive integers. Furthermore, we define two discrete random variables. The first one X corresponds to the value which will increase the corresponding counter in case of a wrong subkey candidate; it has the following distribution:

x	$P_X[X = x]$
1	ρ_w
0	$1 - \rho_w$

The second one Y relates to the same event, but in the case of the right key candidate:

y	$P_Y[Y = y]$
1	ρ_r
0	$1 - \rho_r$

We take a first assumption relating to the independence of the counters:

Assumption 4.1 (Mutual independence of the counters)
The final values of $C^{(r)}$ and of $C_i^{(w)}$, with $1 \leq i \leq 4095$ are the the sum of N mutually independent outcomes of Y, respectively X.

Assuming this fact is motivated by the relative small number out of 2^{64} possible plaintext-ciphertext pairs which are playing a role in getting the final values of the counters.

The second assumption relates to the imbalance measure of the two central equations. We assume here that the approximation given by Matsui in [12] is equal to the real probability.

Assumption 4.2 (Imbalance Measure of the two best Expressions)
The two best 14-rounds expression (9) and (10) hold with probability

$$\rho_r = \frac{1}{2} - 1.19 \cdot 2^{-21} \approx 0.499999432 \tag{23}$$

in case of the right subkey candidate.

The last assumption relates to the wrong key randomisation conjecture. It states that there is no more imbalance in the equations when a wrong key is used to decrypt the first and the last rounds. We assume here that DES is a perfect cipher, i.e. that the ciphertext is independent of the plaintext, which is obviously not the case.

Assumption 4.3 (Wrong Key Randomisation)
The two best 14-rounds expressions (9) and (10) hold with probability

$$\rho_w = \frac{1}{2} \tag{24}$$

in case of a wrong subkey candidate.

The random variables $C^{(r)}$ and of $C_i^{(w)}$ are defined formally as follows:

$$C^{(r)} = \sum_{j=1}^{N} Y_j \tag{25}$$

and

$$C_i^{(w)} = \sum_{j=1}^{N} X_j \quad \text{with} \quad 0 \leq i \leq 4095 \tag{26}$$

where X_j and Y_j are mutually independent random variables having X and Y as common distribution, respectively. The counters follow the binomial distribution. More exactly, the probability that the counter of a wrong key candidate takes a value n with $0 \leq n \leq N$ and $1 \leq i \leq 4095$ is given by

$$P\left[C_i^{(w)} = n\right] = \binom{N}{n} \rho_w^n \left(1 - \rho_w\right)^{N-n} \tag{27}$$

and the counter $C^{(r)}$ has the following distribution for $0 \leq n \leq N$

$$P\left[C^{(r)} = n\right] = \binom{N}{n} \rho_r^n \left(1 - \rho_r\right)^{N-n} \tag{28}$$

Obviously, these distributions are not easy to deal with, but we can fortunately use Theorem 4.1 to approximate them with the corresponding (continuous) normal laws, denoted respectively $\phi^{(w)}_{(\mu_w,\sigma_w)}$ and $\phi^{(r)}_{(\mu_r,\sigma_r)}$. The parameters are summarised in Figure 15.

μ_w	$=$	$N\rho_w$	σ_w	$=$	$\sqrt{N\left(\rho_w - \rho_w^2\right)}$
μ_r	$=$	$N\rho_r$	σ_r	$=$	$\sqrt{N\left(\rho_r - \rho_r^2\right)}$

Figure 15: The parameters of the normal law approximations

Because we ignore the true value of the linear expression, we are in fact more interested in the absolute value of the bias; therefore, we subtract from each counter corresponding to the keys the expected value of a perfect cipher's counter, i.e. $\frac{N}{2} = 2^{42}$, and we take the absolute value of this difference. The resulting random variables are

$$B^{(r)} = \left|C^{(r)} - \rho_w\right| \tag{29}$$

and

$$B_i^{(w)} = \left|C_i^{(w)} - \rho_w\right| \tag{30}$$

for $1 \leq i \leq 4095$. Following lemma is useful to compute the resulting continuous distribution.

Lemma 4.1
Let X be a continuous random variable following the normal law and having $\phi_{(\mu,\sigma)}$ as density function. Then the density function f_Y of

$$Y = |X - a|, \quad a \leq \mu$$

is given by

$$f_Y = \phi_{(\mu,\sigma)}(y + a) + \phi_{(\mu,\sigma)}(-y + a) \tag{31}$$

for $0 \leq y \leq +\infty$ and $f_Y = 0$ otherwise.

Proof :
Let's compute first the cumulative distribution function F_Y:

$$\begin{aligned} F_Y(y) &= P[Y \leq y] \\ &= P[|X-a| \leq y] \\ &= P[-y \leq X-a \leq y] \\ &= \Phi_{(\mu,\sigma)}(y+a) - \Phi_{(\mu,\sigma)}(-y+a) \end{aligned}$$

Then, by definition,

$$\begin{aligned} f_Y = \frac{\partial}{\partial y} F_Y &= \frac{\partial}{\partial y}\Phi_{(\mu,\sigma)}(y+a) - \frac{\partial}{\partial y}\Phi_{(\mu,\sigma)}(-y+a) \\ &= \phi_{(\mu,\sigma)}(y+a) + \phi_{(\mu,\sigma)}(-y+a) \end{aligned}$$

$\diamondsuit$

We can note that in the case of $a = \mu$, we have the simple expression

$$f_Y(y) = 2\phi_{(0,\sigma)}(y) \tag{32}$$

The next trivial Lemma is useful to prove the convergence of some specific function of a convergent sequence of values.

Lemma 4.2
Let a_n be a sequence of values with

$$\lim_{n\to+\infty} a_n = \alpha \quad \textit{with} \quad -\infty < \alpha < +\infty \tag{33}$$

Let be $b_n := |a_n - C|$ where $0 \leq C < +\infty$ is a constant. Then the limit

$$\beta = \lim_{n\to+\infty} b_n \tag{34}$$

is finite.

Proof :
A fundamental result of calculus states that

$$\lim_{n\to+\infty}(a_n \pm b_n) = \lim_{n\to+\infty} a_n \pm \lim_{n\to+\infty} b_n$$

where a_n and b_n are two convergent sequences. Let be $\alpha \geq C$. We have then the following straightforward calculation:

$$\lim_{n\to+\infty} a_n - C = \lim_{n\to\infty} a_n - \lim_{n\to\infty} C = \alpha - C$$

which is a finite real number. The reasoning is similar when $\alpha < C$, which concludes the proof.

Feeding the values of Figure 15 in the equations (31) and (32), respectively, we obtain two functions corresponding to the probability density of the counter's values in both cases. Figure 16 is a plot in a normalised scale (i.e the random variables $C_i^{(w)}$ having a mean of 0 and a variance equal to 1 and $C^{(r)}$ having a mean equal to $\frac{\mu_r - \mu_w}{\sigma_w}$ and a variance equal to $\frac{\sigma_r^2}{\sigma_w^2}$).

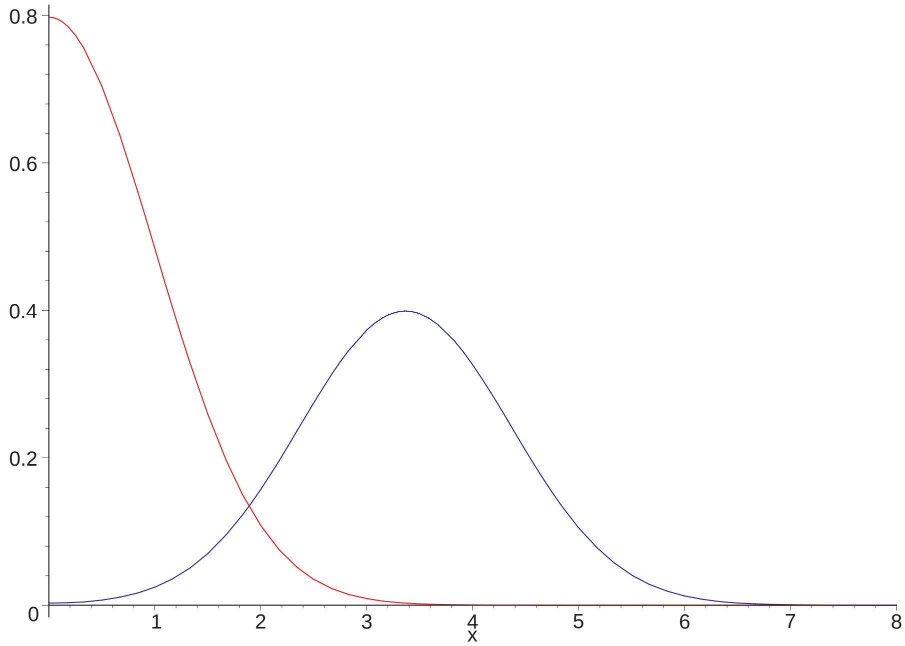

Figure 16: The two fundamental distributions

4.2.2 A Simplified Statistical Experiment

We can as a first step simplify the statistical experiment and define the following one: we take one realization x of the distribution X corresponding to a wrong key and one realization y of the distribution Y corresponding to the right key. What is the probability that $x < y$, i.e. what is the probability that the counter corresponding to the right key is further from the value $N/2$ than the one corresponding to a wrong key ?

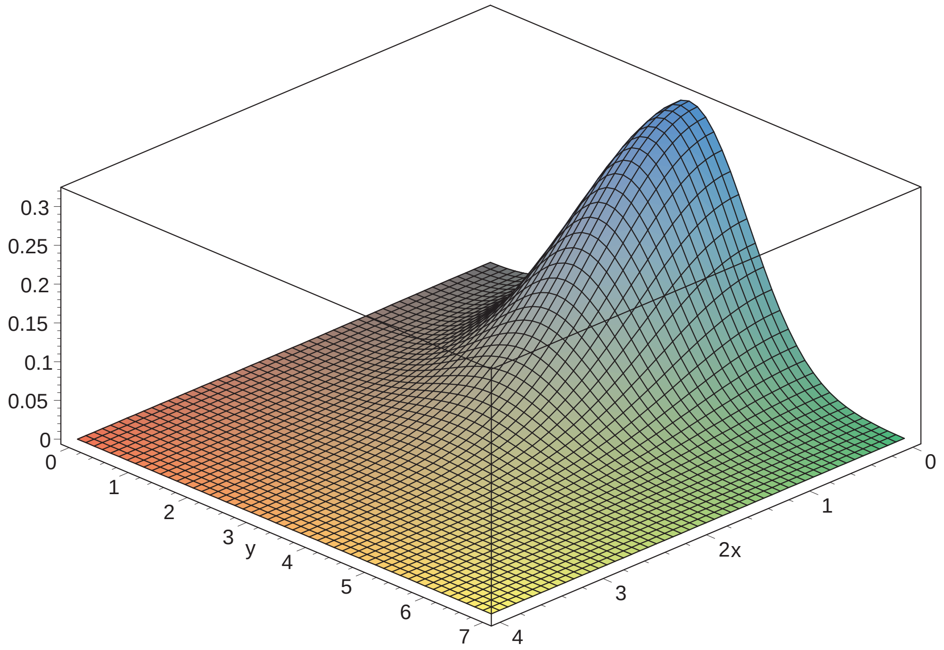

Figure 17: The simple scenario: the joint density f_{XY}

Lemma 4.3
Let $B^{(w)}$ and $B^{(r)}$ be the functions as defined in (29) and (30) of the sum of N mutually independent outcomes of the Bernoulli random variables X and Y, respectively. Under the three previous assumptions, the probability that $B^{(w)} < B^{(r)}$ is given by

$$\lim_{N\to+\infty} P[B^{(w)} < B^{(r)}] = \int_0^{\infty}\int_0^{y} f_{XY}(x,y)\,dx\,dy \tag{35}$$

where

$$\begin{aligned} f_{XY}(x,y) &= f_X(x)\cdot f_Y(y) \\ &= 2\phi_{(0,\sigma_w)}(x)\cdot\left(\phi_{(\mu_r,\sigma_r)}(y+\mu_w)+\phi_{(\mu_r,\sigma_r)}(-y+\mu_w)\right) \end{aligned}$$

Proof :
Following Theorem 4.1, the sums $C^{(w)}$ and $C^{(r)}$ will converge towards two different normal laws, the first one being $\sigma_{(\mu_w,\sigma_w)}$ and the second one $\sigma_{(\mu_r,\sigma_r)}$. It is then easy to see that the convergence of the sum of Bernoulli trials to a normal law X implies the convergence of the random variable $Y = |X - a|$ (see Lemma 4.2). Assumption 4.2 allows us to compute easily the joint

density function, which is in fact the product of the two densities, because of their independence. Hence, we have

$$\begin{aligned} f_{XY}(x,y) &= f_X(x) \cdot f_Y(y) \\ &= 2\phi_{(0,\sigma_w)}(x) \cdot \big(\phi_{(\mu_r,\sigma_r)}(y+\mu_w) + \phi_{(\mu_r,\sigma_r)}(-y+\mu_w)\big) \end{aligned}$$

which concludes the proof.

$\diamondsuit$

Following previous Theorem, the probability computation leads then to the numerical evaluation of a double integral:

$$\lim_{N\to+\infty} P[X<Y] = \int_0^{\infty}\int_0^{y} f_{XY}(x,y)\,dx\,dy \approx 0.98283 \tag{36}$$

One can conclude that the good key will have a bias value larger than the wrong one's approximately 98.3 % of the time.

4.2.3 Towards the Good Distribution

However, the real situation is slightly more complicated. One have to find the right key's counter out of 4096, i.e. the probability that our interesting counter is the bigger one will decrease in a sensitive manner. Our goal is now to compute the probability that the right key's counter has a given rank in the sorted counter list.

In order to clarify the notations, let's denote the distribution function of a bad key candidate in the following manner:

$$W(x) := \int_0^{x} 2\phi_{(0,\sigma_1)}(a)\,da \tag{37}$$

for $0 \le x \le +\infty$. Then, we have

$$P[B_i^{(w)} > x] = 1 - W(x) \tag{38}$$

for $1 \le i \le 4095$.

Following Assumption 4.2, we assume that the values of the 4095 wrong keys counters are mutually independent. In this case, we can compute the probability that less than n out of 4095 counters are bigger than *a fixed value* x using the binomial distribution:

$$P\left[\left(\sum_{i=1}^{4095} 1_{B_i^{(w)}>x}\right) < n\right] = \sum_{i=0}^{n} \binom{4095}{i} (1-W(x))^i\, W(x)^{4095-i} \tag{39}$$

Using Theorem 4.1 a once again, one can show that this binomial law converges towards a normal law as $N \to +\infty$. Thus, one can acceptably approximate this expression by a such a distribution. Let

$$\mu_n(x) := n \cdot (1 - W(x)) \tag{40}$$

and

$$\sigma_n(x) := \sqrt{n\,(1 - W(x))\,W(x)} \tag{41}$$

be the (variable) mean and standard deviation, respectively, of this normal law. Then, one can approximate (39) in the following manner:

$$W_n(x) := P\left[\left(\sum_{i=1}^{4095} 1_{B_i^{(w)} > x}\right) = n\right] \approx \int_{\alpha}^{\beta} \phi(y)\,dy \tag{42}$$

where

$$\alpha := -\infty \tag{43}$$

and

$$\beta := \frac{n - \mu_n(x)}{\sigma_n(x)} \tag{44}$$

We can now fix the parameter n, derive (42) with respect to x and we get a density function $w_n(x)$ over the counter values x, with $0 \leq x \leq +\infty$:

$$w_n(x) := \frac{\partial}{\partial x} W_n(x) \tag{45}$$

We don't give the explicit expression of $w_n(x)$ because of its complexity (the derivative of β being quite complicated), but a symbolic mathematical software gives this expression without any difficulty.

Figure 18 illustrates two density functions for $n = 50$ (red curve) and $n = 100$ (blue curve), the horizontal scale being units of bias scaled by σ_w. One can interpret this figure in the following way for $n = 50$: the probability that less than 50 counters are bigger than $2.3\sigma_w$ is negligible (i.e. there are high probability more than 50 counters which are bigger than $2.3\sigma_w$), while one can be almost certain that less than 50 counters have a value greater than $2.7\sigma_w$.

If we have a look at the blue curve, we can notice that it is located on the left of the red curve, which is very intuitive: the probability that 100 or less counters have a value of, say $2.35\sigma_w$, or more is greater than the probability that less than 50 counters are beyond this value. The displacement of the density towards μ_w for increasing n is illustrated in a more convenient way by Figure 19.

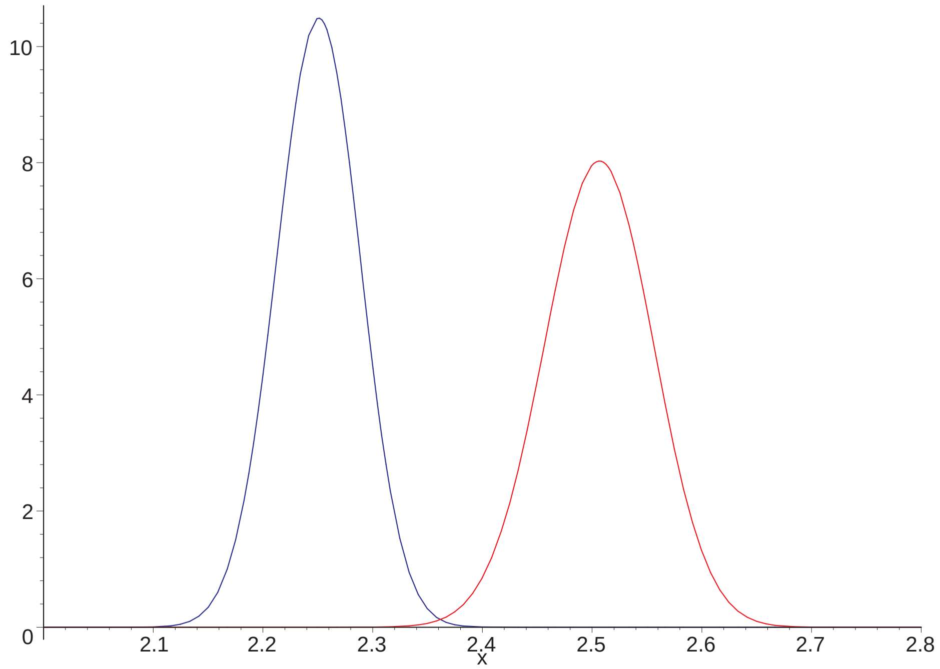

Figure 18: Density functions of the rank probability for $n = 50$ (red curve) and $n = 100$ (blue curve).

4.2.4 Maximal Rank Probability

We have at this time all the needed tools to compute the success probability of the attack. In order to achieve this goal, one have just to replace the density function of a single bad counter by the one of a given number n of counters which are further than a value x.

In order to give a simple expression, let's define the random variable R, which is defined over the possible rankings of the good subkey candidate's counter in the list, i.e. from 1 to 4096. As done before, we denote by f_Y the density function of the random variable Y corresponding to the good key and by $f_{X^{(n)}} := w_n(x)$ the density function $X^{(n)}$ corresponding to n wrong key being further than x. We assume a new time that these random variables are mutually independent, so we can write:

$$f_{X^{(n)}Y}(x, y) = f_{X^{(n)}}(x) \cdot f_Y(y) \qquad (46)$$

We give in Figure 20 the plot of two densities, one with $n = 10$ (the little one) and one with $n = 50$.

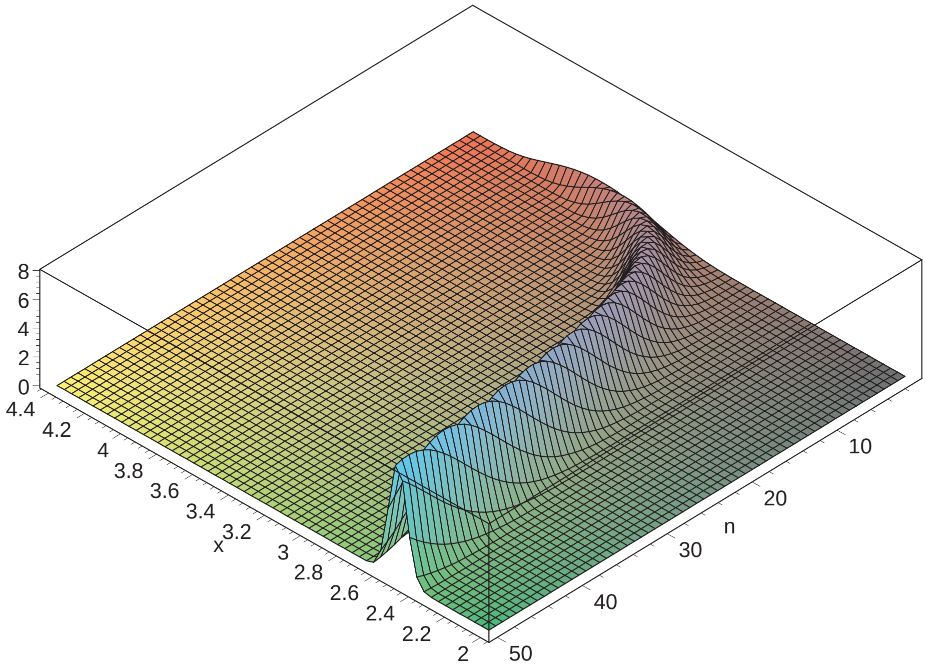

Figure 19: Displacement of rank density towards μ_w for increasing n

The success probability (i.e. the good counter being further than $4095 - n$ wrong counters) is then formally defined by the following expression:

$$P[X^{(n)} < Y] = \int_0^\infty \int_0^y f_{X^{(n)}Y}(x, y)\, dx\, dy \tag{47}$$

or equivalently

$$P\left[R < n + 1\right] = \int_0^\infty \int_0^y f_{X^{(n)}Y}(x, y)\, dx\, dy \tag{48}$$

We have hence proven the following Theorem 4.2.

Theorem 4.2 (Maximal Rank Probability)
Under the three assumptions defined before, the probability that a right subkey candidate has a maximal rank of $n+1$ in the subkey candidates list, with $0 \leq n \leq 4095$, is equal to

$$\lim_{N \to +\infty} P\left[R < n + 1\right] = \int_0^\infty \int_0^y f_{X^{(n)}Y}(x, y)\, dx\, dy \tag{49}$$

$f_{X^{(n)}Y}$ *being defined as before.*

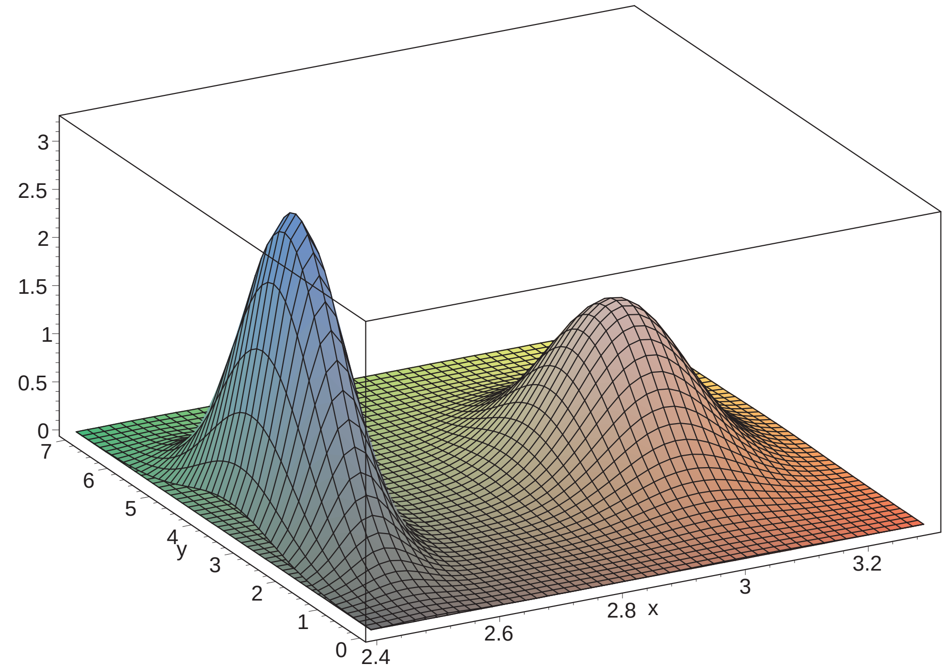

Figure 20: Joint density of the probability success with $n = 10$ and $n = 50$

Annex C gives some interesting values of (49) while Figure 21 gives a plot for these success probabilities. The computation of the corresponding density function succeeded, but it was not possible to evaluate it numerically. The horizontal scale represents the rank n for the good subkey candidate, the vertical scale being the probability that the rank is less or equal to n. The curves are, from the highest to the lowest one, for 2^{43}, $2^{42.5}$, 2^{42}, 2^{41}, 2^{40}, 2^{39} and 2^{38} known plaintext-ciphertexts pairs.

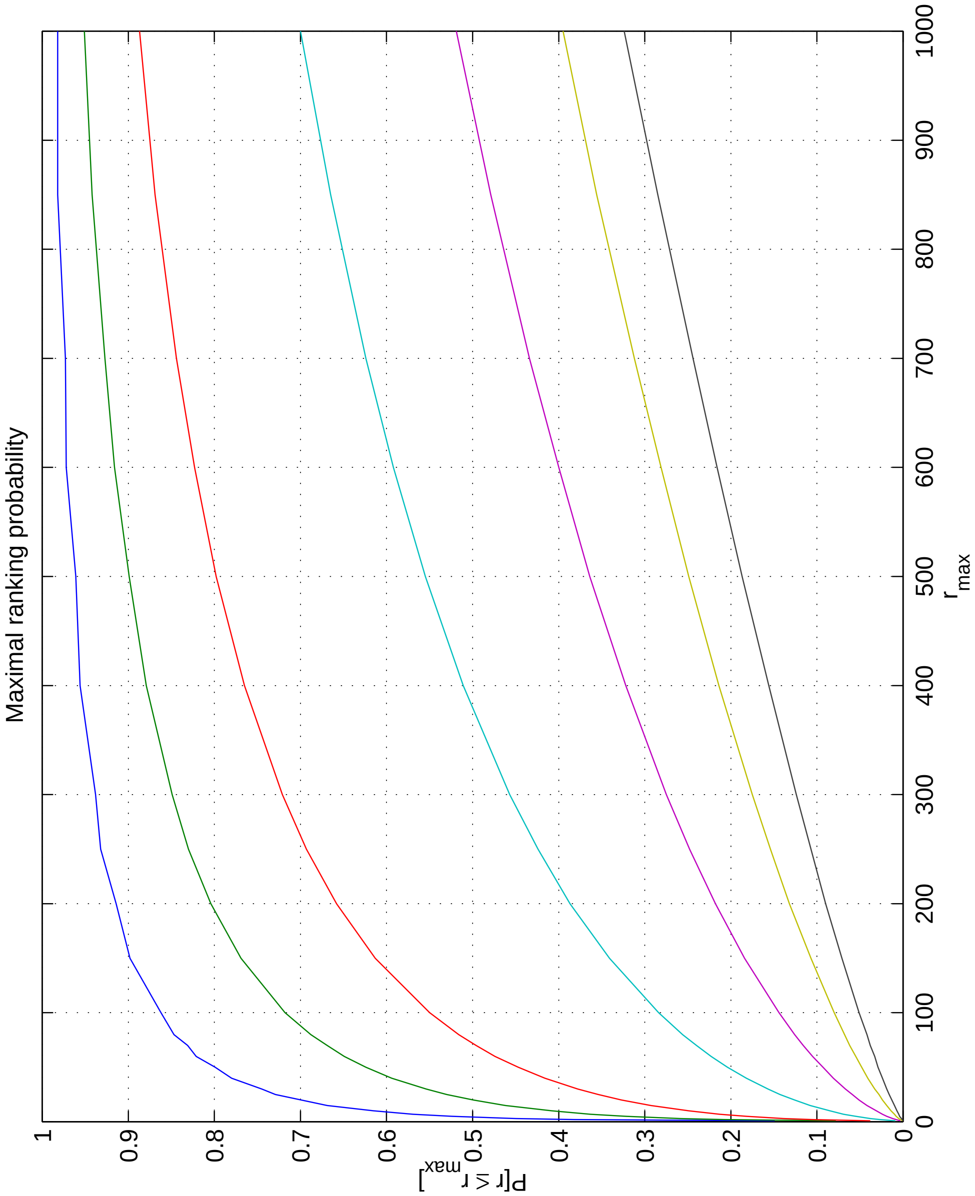

Figure 21: Success probabilities for 2^{43}, $2^{42.5}$, 2^{42}, 2^{41}, 2^{40}, 2^{39} and 2^{38} known plaintext-ciphertexts pairs

4.2.5 Complexity of the Attack

The complexity of the linear cryptanalysis of DES is the number of DES computations done by the exhaustive search part of the attack. Thus, it is clearly related to the ranking of both right subkey candidates. The interesting part is the combination of the two lists in order to get a single list of 26-bits subkey candidates, the most likely one being in the first position, the least likely one in last position.

The function used to combine the two lists was derived experimentally. We call it *Matsui's rule*. It is interesting to note that Matsui's rule only cares of the rank of each candidate, and not of the corresponding bias.

Definition 4.4 (Matsui's rule)
Let be two lists of subkey candidates sorted in order of reliability. Let be r_A the rank of an element in the first list and r_B the rank of an element of the second list. The rank R_{AB} in the final candidates list of the pair (r_A, r_B) is given by the following formula:

$$R_{AB} = f(r_A, r_B) = \sum_{1 \leq a,b \leq 4096} 1_{a \cdot b < r_A \cdot r_B} \tag{50}$$

for $r_A, r_B, a, b \in \mathbb{N}^*$ and $1 \leq r_A, r_B \leq 4096$.

It is not easy to find an exact, simple analytical expression for $f(r_A, r_B)$, although it is possible to approximate it. We have therefore computed it by "brute-force", each function evaluation being a linear search in a 200MB text file.

Having the rank R_{AB} in the final list of the right subkey candidate allows us to compute the average complexity C_{AB} (in DES evaluations) of the exhaustive key search phase:

$$C_{AB} = 2^{30} \cdot (R_{AB} - 1) + 2^{29} \approx \mathcal{O}\left(R_{AB} \cdot 2^{30}\right) \tag{51}$$

We can note that it is possible to optimise slightly the attack with the following modified Matsui's rule:

Definition 4.5 (Modified Rule)
Let be two lists of subkey candidates sorted in order of reliability. Let be p_A the maximal rank probability of an element in the first list and p_B the maximal rank probability of an element in the second list. The rank R_{AB} in the final candidates list of the pair (p_A, p_B) is given by the following formula:

$$R_{AB} = f(p_A, p_B) = \sum_{1 \leq a,b \leq 4096} 1_{p_A(a) \cdot p_B(b) < p_A \cdot p_B} \tag{52}$$

for $0 \leq p_A, p_B \leq 1$, $a, b \in \mathbb{N}^*$ and $1 \leq r_A, r_B \leq 4096$.

We have noticed that this new rule changes the order of pairs which have the same likelihood measure, i.e. when different rank combinations give the same product. Ongoing research will compute the exact gain in complexity.

Another difficulty we have encountered is to formally describe the attack success in function of the maximal rank probability. Or, in other words, how it is possible to compute efficiently $P_{XY}[a < XY]$ when we have $P_X[x < X]$ and $P_Y[y < Y]$ at disposal ? If we succeed to estimate this expression, then it will be possible to give a slightly better lower bound for the success probability of the attack than the experimental values of Matsui.

5 Experimental Results

In this chapter, we give the experimental results of our attacks. The fastest attack was carried in 4.32 days on 18 CPUs. Most of the time, we had between 12 and 15 CPUs at disposal which leads to an average duration of 5-6 days. Furthermore, the computers were used by other people. The consequence is that from time to time a computer crash or a reboot have occurred, or we had simply not all the whole CPU time at disposal.

However, we managed to perform eight attacks in total. This number is not large enough in order to get an *accurate* mean of the success probability and the complexity, but the results suggest strongly that the attack has a far lower complexity than expected.

5.1 Experimental Complexities

Following table summaries the complexity logarithms (in base 2 and given with 2 decimals) of these cases at different level of quantity of plaintext-ciphertext pairs. A value of 33.32 means that the complexity of the exhaustive search part was $2^{33.32}$ DES computations, i.e. $2^{3.32} \doteq 10$ searches on the remaining 30 bits. Annex D gives the detail of the ranks for both linear expressions.

Exp. \ N	2^{43}	$2^{42.5}$	2^{42}	2^{41}	2^{40}	2^{39}	2^{38}
#1	33.32	42.16	39.90	46.91	51.15	51.26	53.21
#2	39.18	42.01	47.19	49.00	51.86	52.62	53.60
#3	38.61	41.80	43.96	48.55	51.60	52.46	53.27
#4	38.35	34.95	41.38	48.93	51.51	49.59	44.92 [1]
#5	37.49	30.00	36.52	46.20	52.37	53.50	53.80
#6	34.09	34.28	46.68	48.52	47.36	52.99	53.87
#7	35.81	45.83	44.43	47.69	51.25	52.21	52.10
#8	36.13	37.86	47.53	48.35	52.64	53.98	52.57

[1] It's not an error, but simply an occurring low-probability event!

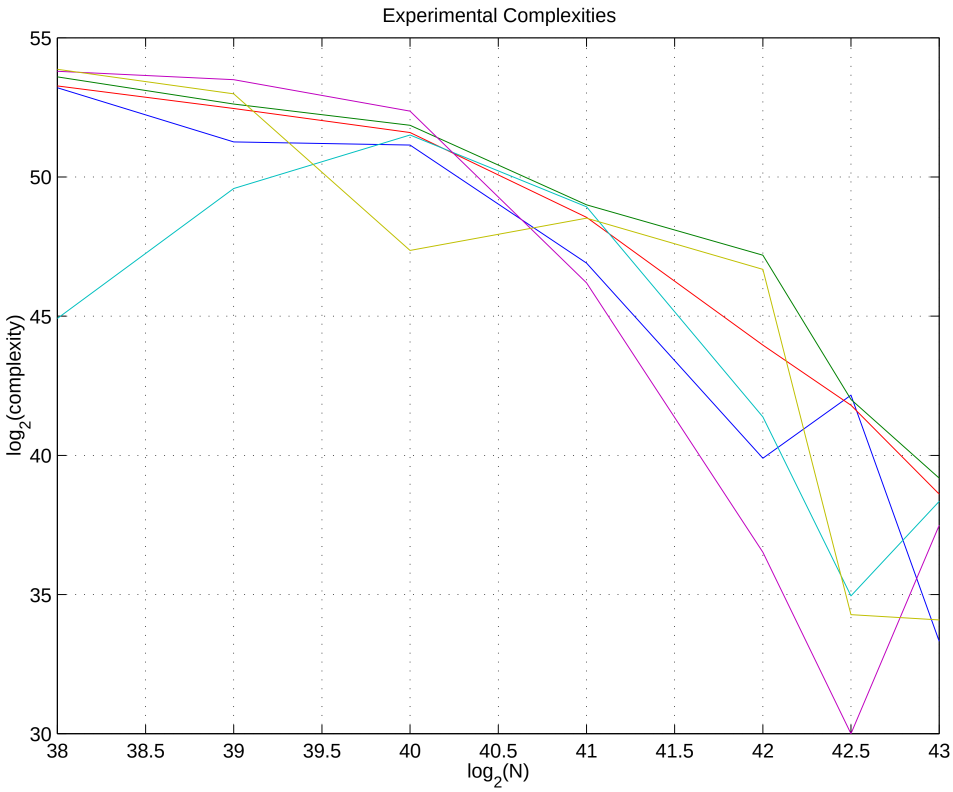

Figure 22: Experimental complexities

Figure 22 gives these results in a graphical way.

Even if we have to take these values carefully because of the small number of trials, the averages over eight experiments (see following table) suggest a far lower complexity than the expected ones. We recall here that Matsui expected a $\mathcal{O}\left(2^{43}\right)$ complexity with a success probability of 85 % in the case of 2^{43} known plaintext-ciphertexts pairs.

N	2^{43}	$2^{42.5}$	2^{42}	2^{41}	2^{40}	2^{39}	2^{38}
μ	37.67	43.11	45.89	48.27	51.68	52.77	53.32
max	39.18	45.83	47.53	49.00	52.64	53.98	53.87

The theoretical probability of guessing the two right bits corresponding to the two right parts of the linear expression being in case of 2^{43} known plaintext-ciphertext pairs approximately equal to 0.992, no false guess occurring in our experiments, one can admit that the success probability only relates to the maximal rank probability.

Knowing this fact, we can note that we never got an attack with a complexity less than 2^{40} DES computations for 2^{43} pairs, which is eight times less than expected with a better success probability. This strongly suggests that the complexity is only *loosely* upper-bounded by Matsui's expectations.

Concerning the other cases of number of known plaintext-ciphertext pairs, we note following points:

- With $2^{42.5}$ known plaintext-ciphertext pairs (i.e. with only 70% of the proposed number of pairs), all but one experiments have broken a DES key with a complexity upper-bounded by $2^{42.5}$ DES computations; this represents a work load of less than 2 days for 18 Intel Pentium III 666MHz CPUs.
- Even with 2^{41} or 2^{42} pairs, linear cryptanalysis remains a very interesting attack, the complexity being always inferior to 2^{50} DES computations; this remains quite acceptable compared to an exhaustive search. We recall that the wrong guessing probabilities are equal to 0.91 and 0.98, respectively.
- For less than 2^{41} known pairs, the complexity remains interesting, but the guessing probability becomes to decrease fast. For example, for experiment #4, we get by chance a *very* low complexity with 2^{38} known pairs, but unfortunately, one of the two guessed bit was false...

5.2 Experimental Success Probabilities

As discussed before, the success probability of linear cryptanalysis can be categorised in two parts: what concerns the guessing of the right part of the two linear expressions, and one which concerns with the maximal rank probability of a subkey candidate in a list.

5.2.1 Experimental Maximal Rank Probabilities

Each experiment gives us statistical material on *two* trials. We have thus 16 samples at disposal. Following table summarises our experimental results while Figure 23 gives the corresponding curves.

If we compare Figure 23 with Figure 21, we can note that the experimental data seem to be lower-bounded, in quite a loose way in the tail of the distribution (i.e. for maximal ranking lesser than 100) by the analytical expression. Comparing Matsui's experimental data, those coming from our analytical expression and the experimental ones, we can see that our analytical expression gives slightly better values for high ranking than Matsui's experimental data. But this expression is obviously far from being very tight.

R_{max} \ N	2^{43}	$2^{42.5}$	2^{42}	2^{41}	2^{40}	2^{39}	2^{38}
≤ 5	8	4	1	0	0	0	1
≤ 10	13	4	1	0	0	0	1
≤ 25	15	10	6	2	0	0	1
≤ 50	15	12	10	4	0	0	1
≤ 100	16	13	11	5	1	1	1
≤ 250	16	16	14	8	3	2	1
≤ 500	16	16	15	9	7	2	1
≤ 1000	16	16	16	15	9	5	3
≤ 2000	16	16	16	15	11	7	10

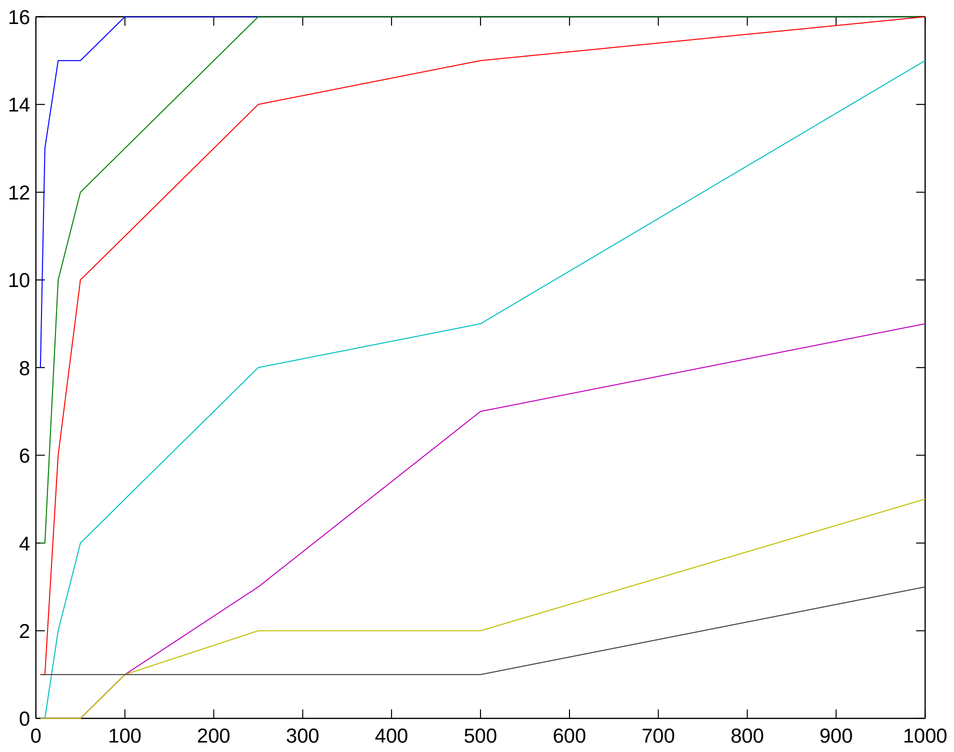

Figure 23: Experimental maximal rank probabilities

5.2.2 Guessing Success Probability

The two bits which are guessed from the right part of the linear expression induce an error which can render the attack unsuccessful even if *all* (i.e.

$4096^2 = 16777216$) the subkey candidates pairs are searched exhaustively. We give in the following table the success of the guessing strategy for our attacks. Each entry in the table corresponds to the number (out of 2) of correctly guessed bits.

Exp. \ N	2^{43}	$2^{42.5}$	2^{42}	2^{41}	2^{40}	2^{39}	2^{38}
#1	2	2	2	2	2	2	2
#2	2	2	2	1	1	1	2
#3	2	2	2	2	2	1	1
#4	2	2	2	2	2	1	1
#5	2	2	2	2	2	2	1
#6	2	2	2	2	2	2	1
#7	2	2	2	2	2	2	2
#8	2	2	2	2	2	1	2
μ_{exp}	2.000	2.000	2.000	1.875	1.875	1.500	1.500
μ_{ana}	1.998	1.996	1.982	1.908	1.766	1.600	1.448

A new time, one have to be careful because of the small sample size, but we can remark that the experimental data seem to correspond in a good way to the analytical expected ones.

5.3 Discussion

We are now able to discuss our analytical and experimental results and to formulate some (empirical) propositions.

5.3.1 Complexity

One of the goals of this research was to perform a statistical complexity analysis and was motivated by the surprisingly low complexity of the first (and at this time the only documented) experimental linear cryptanalysis implemented by Matsui. If have a look at our experimental results, one can be convinced that the average complexity is upper-bounded by Matsui's expectations.

The $\mathcal{O}(2^{43})$ reference complexity for 2^{43} known plaintext-ciphertext pairs has a success probability which is significantly greater than Matsui's experimental value of 85 %. Our experimental results suggest even that the average complexity is upper bounded up to a little failure probability by a value equal to $\mathcal{O}(2^{40})$.

Furthermore, by using only $2^{42.5}$ known plaintext-ciphertext pairs, we get an average complexity which is loosely upper-bounded by a value of 2^{43} DES computations in all but one cases.

Regarding our experimental results, which are quite poor in a statistical way because of the small size of the sample space, we *propose* the following:

Proposition 5.1 (First Proposition)
If 2^{43} *known plaintext-ciphertext pairs are available, the linear cryptanalysis of DES has an average complexity upper-bounded by* 2^{40} *DES computations up to a negligible failure probability.*

Proposition 5.2 (Second Proposition)
If $2^{42.5}$ *known plaintext-ciphertext pairs are available, the linear cryptanalysis of DES has an average complexity upper-bounded by* 2^{43} *DES computations up to a success probability of* 80%.

We repeat a new time that these propositions have to be taken carefully because of the small sample size of our experiments. Although they are a lot better than Matsui's expectations, they reflect in a quite generous way our experimental results.

5.3.2 Maximal Rank Probability of Subkey Candidates

In the previous chapter, we have shown how it was possible to approximate the maximal ranking probability of a subkey candidate using an analytical expression. This expression gives almost the same values than Matsui's experimental ones for high numbers of known pairs and slightly better one in case of lower numbers. But we have to note that these values are only loose lower-bounds of the experimental data; thus the experimental probabilities are a lot better in term of success than the analytical ones.

It is worth to recall how Matsui got his experimental results on the maximum rank probability. He "solved" the linear approximation 100'000 times for 8-rounds DES. Then he showed that, under some assumptions, it possible to estimate the values for the 16-rounds DES case with the help of the experimental values of 8-rounds DES. In term of accuracy, his base data should be almost perfect, up to a flaw in the implementation of the linear approximation solving process, which is not probable at all. The assumptions he takes regarding the step from 8- to 16-rounds DES obviously biases the results in a very sensitive manner.

Our approach is different in the sense that we have taken similar assumptions very early in the process of estimating these probabilities. Our results are however very close to those of Matsui; thus, with two different approaches, one get more or less the same results, which are quite far from the reality. Obviously, one of the hypotheses disturbs a lot the estimations.

Linear cryptanalysis has been extensively studied in the literature. A generalisation of it has been done in [7]. In this paper, the idea of linear expression is studied and generalised by the concept of "Input/Output Sums". One of the results in this paper states that the probabilities given by Matsui for his linear expressions are very likely to be pessimistic. Our experimental and theoretical results go in this direction very clearly and give thus a confirmation of this proposition.

In [19], Vaudenay has shown that it is possible to have a complexity of $\mathcal{O}\left(2^{42.86}\right)$ DES computations by using $2^{42.93}$ known pairs, which is a more optimistic result than Matsui's one. Our experimental results confirm this optimistic trend.

Assumption 4.2 is hence very likely the one which reduce the accuracy of our analytical expression the most. Getting a better bias estimation will give a tighter approximation.

A way which should also lead to possibly interesting results is the better estimation of the tail of the distribution, i.e. for maximal ranking lesser than 100. The most inaccuracy comes from this point. Then normal law has the biggest error for events which have small outcome probabilities.

As a future research, it could be interesting to investigate more precisely this assumptions, in order to understand its relation to the accuracy of the results and to derive tight bounds on the average bias of the linear expression. This would allow to approximate analytically the expected complexity in a far better way!

6 Conclusion

The aim of this diploma thesis 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.

List of Figures

References

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

[2] Kazumaro Aoki and Helger Lipmaa. Fast Implementations of AES Candidates. Second AES Candidate Conference in Rome, Italy, March 1999.

[3] E. Biham. A fast new DES implementation in software. *Lecture Notes in Computer Science*, 1267, 1997.

[4] Eli Biham and Adi Shamir. *Differential Cryptanalysis of the Data Encryption Standard*. Springer-Verlag, Berlin, Germany, 1993.

[5] Electronic Frontier Foundation. *Cracking DES: Secrets of Encryption Research, Wiretap Politics & Chip Design*. O'Reilly & Associates, Inc., July 1998.

[6] William Feller. *An Introduction to Probability Theory and Its Applications, Vol. 1.* Wiley, 1970.

[7] C. Harpes, G. G. Kramer, and J. L. Massey. A generalization of linear cryptanalysis and the applicability of Matsui's piling-up lemma. *Lecture Notes in Computer Science*, 921, 1995.

[8] Intel Corporation. *Intel Architecture Optimization Reference Manual*, 1999.

[9] Intel Corporation. *Intel Architecture Software Developer's Manual, Volume 2: Instruction Set Manual*, 1999.

[10] M. Kwan. Bitslice DES, S-Boxes Implementation. `http://www.darkside.com.au/bitslice/sboxes.c`, March 1998.

[11] A. Clark L. May, L. Penna. An Implementation of Bitsliced DES on the Pentium MMX™ Processor. Accepted at ACISP 2000, 2000.

[12] M. Matsui. The first experimental cryptanalysis of the Data Encryption Standard. *Lecture Notes in Computer Science*, 839:1–11, 1994.

[13] M. Matsui. Linear cryptanalysis method for DES cipher. *Lecture Notes in Computer Science*, 765:386–397, 1994.

[14] M. Matsui. On correlation between the order of S-boxes and the strength of DES. *Lecture Notes in Computer Science*, 950:366–375, 1995.

[15] National Bureau of Standards. *Data Encryption Standard*. U. S. Department of Commerce, Washington, DC, USA, January 1977.

[16] NIST. Advanced Encryption Standard (AES) Development Effort. `http://csrc.nist.gov/encryption/aes/`.

[17] Thomas Pornin. Automatic Software Optimization of Block Ciphers using Bitslicing Techniques. École Normale Supérieure, Paris, 1999.

[18] Bruce Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. John Wiley and Sons, Inc., New York, NY, USA, 1994.

[19] Serve Vaudenay. An experiment on DES statistical cryptanalysis. In Clifford Neuman, editor, *3rd ACM Conference on Computer and Communications Security*, pages 139–147, New Delhi, India, March 1996. ACM Press.

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

A Conversion Between Standard and Matsui's Notations

We begin this part with a quotation of Bruce Schneier ([18]):

> *I am numbering the bits from left to right and from* 1 *to* 64*. Matsui ignores this convention with DES and numbers his bits from right to left and from* 0 *to* 63*. It's enough to drive you mad.*

We can note that the Schneier's notation is the same used in the DES standard. In order to clarify the three different notations used in the literature and in the implementation of Kwan ([10]), we give here explicit conversion tables between all these notations of the different data involved in the DES at different level of the algorithm.

A.1 Standard, Kwan's and Matsui's Notation

The common notation regarding the DES is to number the bits of a 64-bit vector *from left to right and from* 1 *to* 64 like in the figure 24. So, the plaintext bits are denoted P_1 to P_{64}, the corresponding ciphertext bits C_1 to C_{64} and the key bits K_1 to K_{64}.

1	2	3	4	. . .	62	63	64

Figure 24: The common notation

Matsui uses in his papers the following notation: the bits are numbered *from right to left and from* 0 *to* 63, as illustrated in the figure 25. Then the 64-bits vectors are split in two halves, following the Feistel structure of the cipher. We denote these two parts $\mathcal{P}_H^{(i)}$ and $\mathcal{P}_L^{(i)}$ with $0 \leq i \leq 31$ for the plaintext bits and $\mathcal{C}_H^{(i)}$ and $\mathcal{C}_L^{(i)}$ with $0 \leq i \leq 31$ for the ciphertext bits.

63	62	61	60	. . .	2	1	0

Figure 25: Matsui's notation

Furthermore, for simplicity reasons, Matsui ignores the initial permutation and the final one, because they don't modify the behaviour of his attack, the plaintext being assumed to be random. Following Matsui's convention,

$\mathcal{P}_H^{(31)}$ is the leftmost bit of the 64-bits plaintext and $\mathcal{P}_L^{(0)}$ is its rightmost one *after the initial permutation* $\mathcal{IP}$. In a similar manner, $\mathcal{C}_H^{(31)}$ is the leftmost bit of the 64-bits ciphertext and $\mathcal{C}_L^{(0)}$ is its rightmost one *before the final permutation* $\mathcal{IP}^{-1}$.
A further difference between the common and Matsui's notation is that he ignores the initial permutation-compression $\mathcal{PC}_1$ of the key scheduling. Kwan, on his side, numbers the bits like Matsui, but he uses the `C`-array notation. The plaintext bits $P_{64}...P_1$ are stored from `p[0]` to `p[63]` and the ciphertext bits $C_{64}...C_1$ from `c[0]` to `c[63]`.
In the two next parts of this section, we give exhaustive conversion tables between these three notations for the plaintext and the ciphertext, and for the subkeys, respectively.

A.2 Conversion Tables for Plaintext

Matsui	DES	Kwan	Matsui	DES	Kwan	Matsui	DES	Kwan
$\mathcal{P}_H^{(31)}$	P_{58}	`p[6]`	$\mathcal{P}_H^{(30)}$	P_{50}	`p[14]`	$\mathcal{P}_H^{(29)}$	P_{42}	`p[22]`
$\mathcal{P}_H^{(28)}$	P_{34}	`p[30]`	$\mathcal{P}_H^{(27)}$	P_{26}	`p[38]`	$\mathcal{P}_H^{(26)}$	P_{18}	`p[46]`
$\mathcal{P}_H^{(25)}$	P_{10}	`p[54]`	$\mathcal{P}_H^{(24)}$	P_{2}	`p[62]`	$\mathcal{P}_H^{(23)}$	P_{60}	`p[4]`
$\mathcal{P}_H^{(22)}$	P_{52}	`p[12]`	$\mathcal{P}_H^{(21)}$	P_{44}	`p[20]`	$\mathcal{P}_H^{(20)}$	P_{36}	`p[28]`
$\mathcal{P}_H^{(19)}$	P_{28}	`p[36]`	$\mathcal{P}_H^{(18)}$	P_{20}	`p[44]`	$\mathcal{P}_H^{(17)}$	P_{12}	`p[52]`
$\mathcal{P}_H^{(16)}$	P_{4}	`p[60]`	$\mathcal{P}_H^{(15)}$	P_{62}	`p[2]`	$\mathcal{P}_H^{(14)}$	P_{54}	`p[10]`
$\mathcal{P}_H^{(13)}$	P_{46}	`p[18]`	$\mathcal{P}_H^{(12)}$	P_{38}	`p[26]`	$\mathcal{P}_H^{(11)}$	P_{30}	`p[34]`
$\mathcal{P}_H^{(10)}$	P_{22}	`p[42]`	$\mathcal{P}_H^{(9)}$	P_{14}	`p[50]`	$\mathcal{P}_H^{(8)}$	P_{6}	`p[58]`
$\mathcal{P}_H^{(7)}$	P_{64}	`p[0]`	$\mathcal{P}_H^{(6)}$	P_{56}	`p[8]`	$\mathcal{P}_H^{(5)}$	P_{48}	`p[16]`
$\mathcal{P}_H^{(4)}$	P_{40}	`p[24]`	$\mathcal{P}_H^{(3)}$	P_{32}	`p[32]`	$\mathcal{P}_H^{(2)}$	P_{24}	`p[40]`
$\mathcal{P}_H^{(1)}$	P_{16}	`p[48]`	$\mathcal{P}_H^{(0)}$	P_{8}	`p[56]`	$\mathcal{P}_L^{(31)}$	P_{57}	`p[7]`
$\mathcal{P}_L^{(30)}$	P_{49}	`p[15]`	$\mathcal{P}_L^{(29)}$	P_{41}	`p[23]`	$\mathcal{P}_L^{(28)}$	P_{33}	`p[31]`
$\mathcal{P}_L^{(27)}$	P_{25}	`p[39]`	$\mathcal{P}_L^{(26)}$	P_{17}	`p[47]`	$\mathcal{P}_L^{(25)}$	P_{9}	`p[55]`
$\mathcal{P}_L^{(24)}$	P_{1}	`p[63]`	$\mathcal{P}_L^{(23)}$	P_{59}	`p[5]`	$\mathcal{P}_L^{(22)}$	P_{51}	`p[13]`
$\mathcal{P}_L^{(21)}$	P_{43}	`p[21]`	$\mathcal{P}_L^{(20)}$	P_{35}	`p[29]`	$\mathcal{P}_L^{(19)}$	P_{27}	`p[37]`
$\mathcal{P}_L^{(18)}$	P_{19}	`p[45]`	$\mathcal{P}_L^{(17)}$	P_{11}	`p[53]`	$\mathcal{P}_L^{(16)}$	P_{3}	`p[61]`
$\mathcal{P}_L^{(15)}$	P_{61}	`p[3]`	$\mathcal{P}_L^{(14)}$	P_{53}	`p[11]`	$\mathcal{P}_L^{(13)}$	P_{45}	`p[19]`
$\mathcal{P}_L^{(12)}$	P_{37}	`p[27]`	$\mathcal{P}_L^{(11)}$	P_{29}	`p[35]`	$\mathcal{P}_L^{(10)}$	P_{21}	`p[43]`
$\mathcal{P}_L^{(9)}$	P_{13}	`p[51]`	$\mathcal{P}_L^{(8)}$	P_{5}	`p[59]`	$\mathcal{P}_L^{(7)}$	P_{63}	`p[1]`
$\mathcal{P}_L^{(6)}$	P_{55}	`p[9]`	$\mathcal{P}_L^{(5)}$	P_{47}	`p[17]`	$\mathcal{P}_L^{(4)}$	P_{39}	`p[25]`
$\mathcal{P}_L^{(3)}$	P_{31}	`p[33]`	$\mathcal{P}_L^{(2)}$	P_{23}	`p[41]`	$\mathcal{P}_L^{(1)}$	P_{15}	`p[49]`
$\mathcal{P}_L^{(0)}$	P_{7}	`p[57]`						

Figure 26: Conversion table for the plaintext bits *after* $\mathcal{IP}$

A.3 Conversion Tables for the Subkeys

Matsui	DES	Kwan	Matsui	DES	Kwan
$\mathcal{K}_{55}$	K_{50}	`k[6]`	$\mathcal{K}_{54}$	K_{43}	`k[13]`
$\mathcal{K}_{53}$	K_{36}	`k[20]`	$\mathcal{K}_{52}$	K_{29}	`k[27]`
$\mathcal{K}_{51}$	K_{22}	`k[34]`	$\mathcal{K}_{50}$	K_{15}	`k[41]`
$\mathcal{K}_{49}$	K_{8}	`k[48]`	$\mathcal{K}_{48}$	K_{1}	`k[55]`
$\mathcal{K}_{47}$	K_{51}	`k[5]`	$\mathcal{K}_{46}$	K_{44}	`k[12]`
$\mathcal{K}_{45}$	K_{37}	`k[19]`	$\mathcal{K}_{44}$	K_{30}	`k[26]`
$\mathcal{K}_{43}$	K_{23}	`k[33]`	$\mathcal{K}_{42}$	K_{16}	`k[40]`
$\mathcal{K}_{41}$	K_{9}	`k[47]`	$\mathcal{K}_{40}$	K_{2}	`k[54]`
$\mathcal{K}_{39}$	K_{52}	`k[4]`	$\mathcal{K}_{38}$	K_{45}	`k[11]`
$\mathcal{K}_{37}$	K_{38}	`k[18]`	$\mathcal{K}_{36}$	K_{31}	`k[25]`
$\mathcal{K}_{35}$	K_{24}	`k[32]`	$\mathcal{K}_{34}$	K_{17}	`k[39]`
$\mathcal{K}_{33}$	K_{10}	`k[46]`	$\mathcal{K}_{32}$	K_{3}	`k[53]`
$\mathcal{K}_{31}$	K_{53}	`k[3]`	$\mathcal{K}_{30}$	K_{46}	`k[10]`
$\mathcal{K}_{29}$	K_{39}	`k[17]`	$\mathcal{K}_{28}$	K_{32}	`k[24]`
$\mathcal{K}_{27}$	K_{56}	`k[0]`	$\mathcal{K}_{26}$	K_{49}	`k[7]`
$\mathcal{K}_{25}$	K_{42}	`k[14]`	$\mathcal{K}_{24}$	K_{35}	`k[21]`
$\mathcal{K}_{23}$	K_{28}	`k[28]`	$\mathcal{K}_{22}$	K_{21}	`k[35]`
$\mathcal{K}_{21}$	K_{14}	`k[42]`	$\mathcal{K}_{20}$	K_{7}	`k[49]`
$\mathcal{K}_{19}$	K_{55}	`k[1]`	$\mathcal{K}_{18}$	K_{48}	`k[8]`
$\mathcal{K}_{17}$	K_{41}	`k[15]`	$\mathcal{K}_{16}$	K_{34}	`k[22]`
$\mathcal{K}_{15}$	K_{27}	`k[29]`	$\mathcal{K}_{14}$	K_{20}	`k[36]`
$\mathcal{K}_{13}$	K_{13}	`k[43]`	$\mathcal{K}_{12}$	K_{6}	`k[50]`
$\mathcal{K}_{11}$	K_{54}	`k[2]`	$\mathcal{K}_{10}$	K_{47}	`k[9]`
$\mathcal{K}_{9}$	K_{40}	`k[16]`	$\mathcal{K}_{8}$	K_{33}	`k[23]`
$\mathcal{K}_{7}$	K_{26}	`k[30]`	$\mathcal{K}_{6}$	K_{19}	`k[37]`
$\mathcal{K}_{5}$	K_{12}	`k[44]`	$\mathcal{K}_{4}$	K_{5}	`k[51]`
$\mathcal{K}_{3}$	K_{25}	`k[31]`	$\mathcal{K}_{2}$	K_{18}	`k[38]`
$\mathcal{K}_{1}$	K_{11}	`k[45]`	$\mathcal{K}_{0}$	K_{4}	`k[52]`

Figure 27: Key after the compression permutation $\mathcal{PC}_1$

Matsui	DES	Kwan	Matsui	DES	Kwan
$\mathcal{K}_{47}^{(1)}$	K_9	`k[47]`	$\mathcal{K}_{46}^{(1)}$	K_{45}	`k[11]`
$\mathcal{K}_{45}^{(1)}$	K_{30}	`k[26]`	$\mathcal{K}_{44}^{(1)}$	K_{53}	`k[3]`
$\mathcal{K}_{43}^{(1)}$	K_{43}	`k[13]`	$\mathcal{K}_{42}^{(1)}$	K_{15}	`k[41]`
$\mathcal{K}_{41}^{(1)}$	K_{29}	`k[27]`	$\mathcal{K}_{40}^{(1)}$	K_{50}	`k[6]`
$\mathcal{K}_{39}^{(1)}$	K_2	`k[54]`	$\mathcal{K}_{38}^{(1)}$	K_8	`k[48]`
$\mathcal{K}_{37}^{(1)}$	K_{17}	`k[39]`	$\mathcal{K}_{36}^{(1)}$	K_{37}	`k[19]`
$\mathcal{K}_{35}^{(1)}$	K_3	`k[53]`	$\mathcal{K}_{34}^{(1)}$	K_{31}	`k[25]`
$\mathcal{K}_{33}^{(1)}$	K_{23}	`k[33]`	$\mathcal{K}_{32}^{(1)}$	K_{22}	`k[34]`
$\mathcal{K}_{31}^{(1)}$	K_{39}	`k[17]`	$\mathcal{K}_{30}^{(1)}$	K_{51}	`k[5]`
$\mathcal{K}_{29}^{(1)}$	K_{52}	`k[4]`	$\mathcal{K}_{28}^{(1)}$	K_1	`k[55]`
$\mathcal{K}_{27}^{(1)}$	K_{32}	`k[24]`	$\mathcal{K}_{26}^{(1)}$	K_{24}	`k[32]`
$\mathcal{K}_{25}^{(1)}$	K_{16}	`k[40]`	$\mathcal{K}_{24}^{(1)}$	K_{36}	`k[20]`
$\mathcal{K}_{23}^{(1)}$	K_{20}	`k[36]`	$\mathcal{K}_{22}^{(1)}$	K_{25}	`k[31]`
$\mathcal{K}_{21}^{(1)}$	K_{35}	`k[21]`	$\mathcal{K}_{20}^{(1)}$	K_{48}	`k[8]`
$\mathcal{K}_{19}^{(1)}$	K_{33}	`k[23]`	$\mathcal{K}_{18}^{(1)}$	K_4	`k[52]`
$\mathcal{K}_{17}^{(1)}$	K_{42}	`k[14]`	$\mathcal{K}_{16}^{(1)}$	K_{27}	`k[29]`
$\mathcal{K}_{15}^{(1)}$	K_5	`k[51]`	$\mathcal{K}_{14}^{(1)}$	K_{47}	`k[9]`
$\mathcal{K}_{13}^{(1)}$	K_{21}	`k[35]`	$\mathcal{K}_{12}^{(1)}$	K_{26}	`k[30]`
$\mathcal{K}_{11}^{(1)}$	K_{54}	`k[2]`	$\mathcal{K}_{10}^{(1)}$	K_{19}	`k[37]`
$\mathcal{K}_{9}^{(1)}$	K_{34}	`k[22]`	$\mathcal{K}_{8}^{(1)}$	K_{56}	`k[0]`
$\mathcal{K}_{7}^{(1)}$	K_{14}	`k[42]`	$\mathcal{K}_{6}^{(1)}$	K_{18}	`k[38]`
$\mathcal{K}_{5}^{(1)}$	K_{40}	`k[16]`	$\mathcal{K}_{4}^{(1)}$	K_{13}	`k[43]`
$\mathcal{K}_{3}^{(1)}$	K_{12}	`k[44]`	$\mathcal{K}_{2}^{(1)}$	K_{55}	`k[1]`
$\mathcal{K}_{1}^{(1)}$	K_{49}	`k[7]`	$\mathcal{K}_{0}^{(1)}$	K_{28}	`k[28]`

Figure 28: Subkey of Round #1

Matsui	DES	Kwan	Matsui	DES	Kwan
$\mathcal{K}_{47}^{(16)}$	K_{16}	`k[40]`	$\mathcal{K}_{46}^{(16)}$	K_{52}	`k[4]`
$\mathcal{K}_{45}^{(16)}$	K_{37}	`k[19]`	$\mathcal{K}_{44}^{(16)}$	K_{3}	`k[53]`
$\mathcal{K}_{43}^{(16)}$	K_{50}	`k[6]`	$\mathcal{K}_{42}^{(16)}$	K_{22}	`k[34]`
$\mathcal{K}_{41}^{(16)}$	K_{36}	`k[20]`	$\mathcal{K}_{40}^{(16)}$	K_{32}	`k[24]`
$\mathcal{K}_{39}^{(16)}$	K_{9}	`k[47]`	$\mathcal{K}_{38}^{(16)}$	K_{15}	`k[41]`
$\mathcal{K}_{37}^{(16)}$	K_{24}	`k[32]`	$\mathcal{K}_{36}^{(16)}$	K_{44}	`k[12]`
$\mathcal{K}_{35}^{(16)}$	K_{10}	`k[46]`	$\mathcal{K}_{34}^{(16)}$	K_{38}	`k[18]`
$\mathcal{K}_{33}^{(16)}$	K_{30}	`k[26]`	$\mathcal{K}_{32}^{(16)}$	K_{29}	`k[27]`
$\mathcal{K}_{31}^{(16)}$	K_{46}	`k[10]`	$\mathcal{K}_{30}^{(16)}$	K_{1}	`k[55]`
$\mathcal{K}_{29}^{(16)}$	K_{2}	`k[54]`	$\mathcal{K}_{28}^{(16)}$	K_{8}	`k[48]`
$\mathcal{K}_{27}^{(16)}$	K_{39}	`k[17]`	$\mathcal{K}_{26}^{(16)}$	K_{31}	`k[25]`
$\mathcal{K}_{25}^{(16)}$	K_{23}	`k[33]`	$\mathcal{K}_{24}^{(16)}$	K_{43}	`k[13]`
$\mathcal{K}_{23}^{(16)}$	K_{27}	`k[29]`	$\mathcal{K}_{22}^{(16)}$	K_{5}	`k[51]`
$\mathcal{K}_{21}^{(16)}$	K_{42}	`k[14]`	$\mathcal{K}_{20}^{(16)}$	K_{55}	`k[1]`
$\mathcal{K}_{19}^{(16)}$	K_{40}	`k[16]`	$\mathcal{K}_{18}^{(16)}$	K_{11}	`k[45]`
$\mathcal{K}_{17}^{(16)}$	K_{49}	`k[7]`	$\mathcal{K}_{16}^{(16)}$	K_{34}	`k[22]`
$\mathcal{K}_{15}^{(16)}$	K_{12}	`k[44]`	$\mathcal{K}_{14}^{(16)}$	K_{54}	`k[2]`
$\mathcal{K}_{13}^{(16)}$	K_{28}	`k[28]`	$\mathcal{K}_{12}^{(16)}$	K_{33}	`k[23]`
$\mathcal{K}_{11}^{(16)}$	K_{6}	`k[50]`	$\mathcal{K}_{10}^{(16)}$	K_{26}	`k[30]`
$\mathcal{K}_{9}^{(16)}$	K_{41}	`k[15]`	$\mathcal{K}_{8}^{(16)}$	K_{4}	`k[52]`
$\mathcal{K}_{7}^{(16)}$	K_{21}	`k[35]`	$\mathcal{K}_{6}^{(16)}$	K_{25}	`k[31]`
$\mathcal{K}_{5}^{(16)}$	K_{47}	`k[9]`	$\mathcal{K}_{4}^{(16)}$	K_{20}	`k[36]`
$\mathcal{K}_{3}^{(16)}$	K_{19}	`k[37]`	$\mathcal{K}_{2}^{(16)}$	K_{7}	`k[49]`
$\mathcal{K}_{1}^{(16)}$	K_{56}	`k[0]`	$\mathcal{K}_{0}^{(16)}$	K_{35}	`k[21]`

Figure 29: Subkey of Round #16

B Speed Measurement Procedure

In order to get accurate speed measurements for our routines, we decided to use the well-defined measurement procedure which was used in [2] to measure optimised versions of the five final AES candidates.

B.1 The Speed Measurement Routine

The Intel Pentium architecture offers the possibility (see [9]) of monitoring events like cache activity, instruction decoders activity or bus activity among others. The one which interests us is the processor's *time-stamp counter*, which is incremented by one every clock-cycle. The machine instruction `RDTSC` loads the current value of this counter into the `EDX:EAX` registers. We note here that the `EAX` register overflows every 6.4 seconds for a processor clocked at 666 MHz. Thus, it is sufficient for our purposes to ignore the high-order bits of this counter. Listing B.1 gives the assembly code of the measurement routine.

Listing B.1 Time measurement code

```
mov     ebx, time       ; load the address of the counter in ebx
movd    mm0, [ebx]      ; warm cache and set MMX state
xor     eax, eax        ; eax := 0
cpuid                   ; serialise instructions
rdtsc                   ; read time-stamp counter
mov     time, eax       ; save counter
xor     eax, eax        ; eax := 0
cpuid                   ; serialise instructions

;; Here the code to be measured

xor     eax, eax        ; eax := 0
cpuid                   ; serialise instructions
rdtsc                   ; read time-stamp counter
sub     [ebx], eax      ; compute the difference and store it
emms                    ; empty MMX state
```

B.2 The Measurement Procedure

Furthermore, we can note that this method has some overhead, mainly due to both high latency of the `rdtsc` instructions and to the possible branch instructions in case of iterating a function. Looping instructions could be seen as a part of the routine, however. In order to measure this overhead, we define, as in [2], the null function which does nothing (see Listing B.2).

Listing B.2 Null function

```
  cmp     iter, 0       ; Is number of iterations equal to 0 ?
  jz      L1            ; If yes, end of the iteration
  align 16
L0:
  dec     iter          ; iter--
  jnz L0                ; if (iter > 0) iterate one more time
L1:
```

If we subtract the overhead due to the iteration management, which can be measured with the help of the null function, we obtain intuitively the number of clock cycles corresponding to an unrolled implementation of the routine, without any overhead.

Even if these performance-monitoring events counters are intended as guides for performance tuning, the counter values are not always absolutely accurate ([8]). By repeating the measures 1000 times and getting the smaller number of clock cycles, it is however possible to get a quite acceptable precision, since it corresponds most likely to the case where most of the data are in the L1 cache and the branch prediction works successfully: i.e. the speed of the routine in the best conditions.

C Approximation Values of the Success Probability

We give here some numerical approximations for the success probability given a rank n in a single list (i.e. corresponding to *one* linear expression). These values are those given by the analytical expression of Theorem 4.2.

n \ $\log_2 N$	43	42.5	42	41	40	39	38
1	0.149	0.078	0.038	0.009	0.003	0.000	0.000
2	0.382	0.208	0.106	0.027	0.008	0.003	0.001
3	0.450	0.259	0.139	0.038	0.012	0.005	0.002
5	0.524	0.322	0.182	0.055	0.019	0.008	0.004
7	0.569	0.363	0.213	0.069	0.024	0.010	0.005
10	0.615	0.408	0.249	0.085	0.031	0.014	0.007
20	0.699	0.499	0.327	0.126	0.051	0.024	0.013
30	0.745	0.554	0.378	0.157	0.067	0.033	0.019
50	0.799	0.624	0.447	0.204	0.093	0.048	0.029
70	0.831	0.669	0.496	0.240	0.116	0.062	0.038
100	0.862	0.718	0.550	0.284	0.144	0.080	0.051
200	0.914	0.804	0.658	0.387	0.218	0.132	0.090
300	0.938	0.849	0.721	0.457	0.275	0.175	0.124
500	0.961	0.899	0.798	0.555	0.364	0.249	0.187
700	0.973	0.927	0.844	0.624	0.434	0.312	0.244
1000	0.982	0.951	0.887	0.700	0.519	0.395	0.324

D Detailed Experimental Ranks

We give in this table the detailed experimental ranks for the eight experiments. A value of $(199, 2888)$ represents a rank of 199 in the first list and a rank of 2888 in the second one. The first table gives the value for pairs number from 2^{43} down to 2^{41}.

Exp. \ $\log_2 N$	43	42.5	42	41
1	(5, 1)	(18,38)	(15, 12)	(57, 281)
2	(59, 2)	(57, 11)	(99, 201)	(548, 153)
3	(7, 12)	(24, 23)	(114, 18)	(576, 101)
4	(9, 8)	(12, 1)	(11, 39)	(83, 951)
5	(45, 1)	(1, 1)	(1, 27)	(719, 13)
6	(1, 8)	(22, 113)	(39, 345)	(36, 1581)
7	(3, 6)	(48, 148)	(14, 195)	(48, 610)
8	(7, 3)	(14, 4)	(717, 36)	(2058, 24)

The remaining values are the following ones:

Exp. \ $\log_2 N$	40	39	38
1	(270, 1918)	(199, 2888)	(748, 1948)
2	(819, 1214)	(1018, 2042)	(2810, 3417)
3	(383, 2047)	(775, 2301)	(978, 1343)
4	(185, 3913)	(60, 2275)	(2, 3718)
5	(506, 3212)	(1454, 3905)	(3980, 2715)
6	(57, 398)	(833, 3715)	(3584, 2869)
7	(205, 2779)	(2344, 602)	(1211, 1026)
8	(2773, 771)	(3940, 3737)	(1136, 1752)

The following table gives the final (combined) rankings for the case of 2^{43} known pairs:

Exp.	1	2	3	4	5	6	7	8
Rank	10	582	390	326	180	17	56	70