Regular paper

The signal to noise ratio in the differential cryptanalysis of 9 rounds of data encryption standard

Michał Misztal

Abstract— There is presented the differential cryptanalysis method of attack on data encryption standard (DES) reduced to 9 rounds. More precise estimation than that of Biham and Shamir of the signal to noise (S/N) ratio is obtained. Also, method of using the ratio in calculation of required number of plaintexts is suggested. There are given results (time of performance) and implementation's issues of practical realisation of this attack.

Keywords— block cipher DES, differential cryptanalysis, substitution boxes, S/N ratio.

1. Introduction

Differential cryptanalysis and its modifications (like for instance impossible differentials) are the most powerful method of attacks on the popular symmetric cryptosystems – block ciphers. The idea of differential cryptanalysis was introduced in 1991 and applied to the former data encryption standard (DES). At present every newly designed block cipher must be at once evaluate due to resistance to the differential attack. Hence differential attacks on contemporary block ciphers are possible only in theory or for small number of rounds [1]. However differential cryptanalysis could be still improved by applying in practice to some well-known ciphers like mentioned DES.

This article is the continuation of paper [2], in which practical attack with differential cryptanalysis on DES cipher reduced to 8 rounds was performed. In the introduction of that paper it was stated that attack on more than 6 or 8 rounds of DES requires too much amount of data (encryption of too many plaintexts) to be preformed in practice. Due to increasing of computational power of computers (processor speed, capacity of operational and disc memory) attacks which were considered as only theoretical become now possible in practice. At the beginning of 90's, when idea of differential cryptanalysis was born, its inventors did not have possibility to verify their theories in practice. Simple attacks on 3 or 4 rounds of DES were possible but for more rounds only theoretical estimation of required amount of data and complexity time was done. Practical attack on 8 rounds of DES [2] showed that many of these theoretical estimations differ from reality (for example 25 000 pairs is far to small to succeed). At presence thanks to available processors and especially to capacity of memory (cf. Section 5) we can perform attack even on 9 rounds. Thanks to that we can do some experiments and obtain practical, precise results and than compare them to theory. It is the main aim of this paper.

We start from the brief recollection of idea of differential cryptanalysis and the scheme of DES cipher. More details can be found in given references [2–4]. In Section 4 we show how to calculate the S/N ratio of counting scheme of attack on 9 round of DES in more precise way than up to now. We also suggest a method of using the ratio in calculation of required number of plaintexts. We theoretically calculate the efficacy of filtration of wrong pairs and compare it to practice. In Section 5 the implementation's issues of practical realisation of this attack are given. At the end we present results (running time and efficacy) of performed cryptanalysis.

2. The DES algorithm

The data encryption standard algorithm was the encryption standard since year 1977. In year 2001 it was replaced by chosen in contest block cipher Rijndael, which became advanced encryption standard (AES).

The DES algorithm is a block cipher, which in standard version encrypts 64-bit block of plaintext to 64-bit block of ciphertext with 64-bit key. Due to standard actual length of key is only 56-bits and 8 bits are extra bits used only for parity check. Algorithm consists of 16 rounds and is based on structure called Feistel's network. In every round left half of block is xored with result of f function applied to right half of block. Then in every round but last both halves are swapped. Hence f function is main element of every round. It transforms half of encrypted block (32-bits) with 48-bit subkey of round. Every subkey is obtained from main key by algorithm called key schedule.

The f function uses two permutations E and P and also 8 nonlinear mappings called substitution tables or substitution boxes (briefly s-boxes). Extending permutation E transforms 32-bit block to 48-bit block. Permutation P transforms 32-bit block to 32-bit block. In every s-box 6-bit input is transformed to 4-bit output. General scheme of algorithm and scheme of f function are presented in Figs. 1 and 2. Key schedule, permutations E and P and all eight s-boxes can be found in given references.

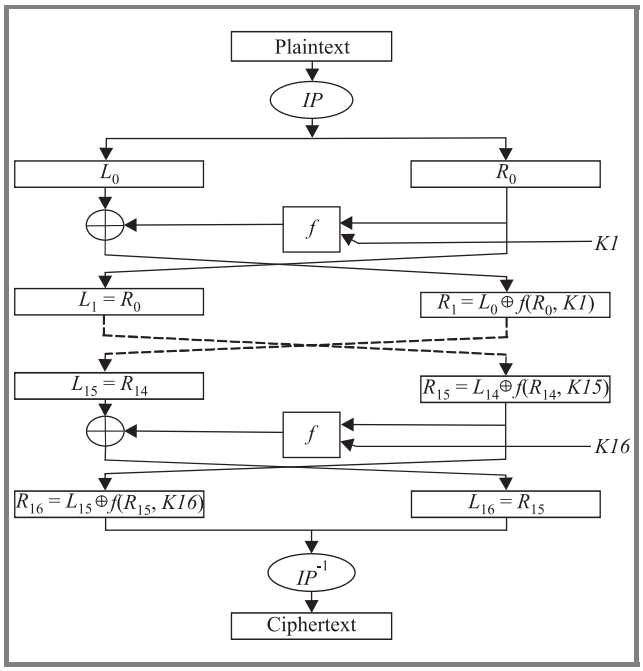


Fig. 1. Scheme of DES algorithm.

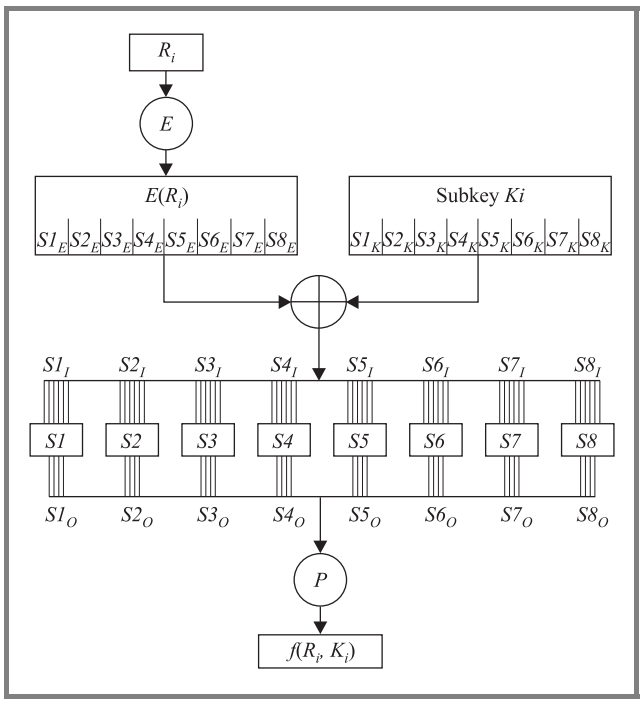


Fig. 2. Scheme of f function.

The DES algorithm reduced to n rounds is an algorithm in which two changes were made:

- number of rounds was reduced from 16 to n, but in the nth (last) round there is no swapping of halves, like after 16th round in standard version;
- permutations IP and IP^{-1} were removed due to their insignificancy to cryptanalysis.

3. Differential cryptanalysis

The differential cryptanalysis was introduced in year 1991 by Biham and Shamir [3] as modern method of cryptanalysis of DES. At least in theory this method is better than exhaustive search, it means testing all possible 2^{56} keys. It is based on dependency between pairs of plaintexts with certain difference (in term of XOR) and differences of their ciphertexts. From above the name "differential" was derived. It is a chosen plaintext (CPA) type of cryptanalysis.

Basic idea of differential cryptanalysis is observation of behaviour of pair of blocks with certain difference transformed through rounds of cipher rather than single block. For linear mappings like permutations, XOR operations the difference of pair of blocks behaves in deterministic way, like single block. The most important thing is that XOR with unknown key does not change this difference. If two arbitrary blocks X and X^* with known difference $X' = X \oplus X^*$ are XORed with unknown key K then the new values of this blocks $X \oplus K$ and $X^* \oplus K$ are unknown but their difference X' remains the same. It happens because of property of XOR operation in which $K \oplus K = 0$. The only problem during analysis of propagation of differences is the application of nonlinear mappings, i.e., s-boxes. For s-boxes we can find input and output difference, which occurs more frequently than others, i.e., with higher probability. However it makes that differential cryptanalysis is only probabilistic method. Its results depend on certain value of key, chosen plaintext and it requires sufficiently many tries to find correct key. To find the best input – output difference propagations of s-boxes the so-called XOR profiles are constructed. The XOR profile of s-box is a table, which shows how many certain input difference goes to certain output difference. XOR profiles are discussed also in the next section.

If we know or we can predict behaviour of differences in individual operations and rounds, then we can find input difference, i.e., difference of two plaintexts which after first round goes to certain difference with some probability, which subsequently after second round goes to another difference with some probability and so on. This sequence of successive differences between successive rounds from plaintext difference to ciphertext difference is called differential characteristic. Every characteristic has its probability, which is calculated as product of probabilities of difference propagations for all rounds. Main problem in differential cryptanalysis is to find "good" differential characteristic, which means characteristic with high probability.

Only differential characteristic with sufficiently high probability makes possible to perform a differential attack. If we have such characteristic we choose pairs of plaintexts with difference given by this characteristic and we obtain their ciphertexts. Then we try to discard pairs, which do not follow our characteristic. This process we call filtration. We know only plaintext and ciphertext difference and we do not

know difference between individual rounds, because cipher is a black box for us. Hence we cannot be sure which pairs are good, i.e., follow our characteristic and which pairs are wrong and only look like they follow this characteristic. In the filtration process we can analyse ciphertext difference and discard pairs, which are wrong for sure. All good pairs will survive this filtration but some number of wrong pairs will survive as well.

After filtration for every non-discarded pair we may find possible subkeys for example last or first round or parts of these subkeys (for example 30 out of 48 bits) in procedure called key recovery. Detailed scheme of this procedure can be found in [3] or [2]. Every pair suggests several subkeys. Good pair suggests exactly one good subkey and few wrong subkeys. Wrong pair suggests only wrong subkeys. Hence we have to count for every non-discarded pair how many times every subkey occurred. For sufficient number of analysed pairs the correct subkey should be the most frequently appeared subkey. The indicator of how many times the correct subkey is more frequent than other subkeys is signal to noise (S/N) ratio. Precise calculation of this parameter can inform us about chances of success of certain attack. Due to this parameter we can also determine the number of pairs required to assure success of the attack. In the next section we discuss how to calculate the S/N ratio in attack on 9 rounds DES and how to determine the required number of pairs from the S/N ratio. Given method is general and could be used in other attacks and for other ciphers.

4. The signal to noise ratio in attack on 9 rounds of DES

4.1. Differential characteristic and its probability

To attack algorithm DES reduced to 9 rounds we use the following 6-round differential characteristic – see Fig. 3.

The characteristic is taken from [3] and it is the best differential characteristic of DES found up to now. Its probability is a product of probabilities of 6 successive rounds and it is equal to:

$$p = \frac{(12\cdot 14\cdot 16)\cdot 1\cdot (10\cdot 16)\cdot 1\cdot (10\cdot 16)\cdot 1}{64^3\cdot 4\cdot 64^2\cdot 64^2\cdot 4} = \frac{2^{17}\cdot 525}{2^{46}}$$

$$= \frac{525}{2^{29}} \approx 9.7788870334625244140625\cdot 10^{-7}$$

$$\approx \frac{1}{1000000}\,.$$

With this characteristic we can attack 9 rounds of data encryption standard by adding three more rounds (so-called 3R attack). Due to the characteristic for good pairs we have:

$$R'_6 = 40\ 5C\ 00\ 00_x\,,$$

hence for five s-boxes: $S2$, $S5$, $S6$, $S7$, $S8$:

$$S'_{E7} = S'_{I7} = 0 \quad \text{and} \quad S'_{O7} = 0,$$

where:

S'_{E7} means 6-bit difference after permutation E in 7th round,

S'_{I7} means 6-bit difference before s-boxes layer in 7th round,

S'_{O7} means 4-bit difference after s-boxes layer in 7th round.

Due to scheme of DES for these s-boxes we have the following relation:

$$f(R_8, K9)' = C'_L \oplus R'_5 \oplus f(R_6, K7)\,.$$

The characteristic allows us to obtain for these 5 s-boxes input and output in 9th round required for key recovery procedure. By applying the characteristic and the key recovery procedure for one round (9th in that case) we obtain 5 (s-boxes) · 6 bits = 30 bits of subkey $K9$, and 30 bits of main key as well. Remaining 26 bits of main key can be found by exhaustive search.

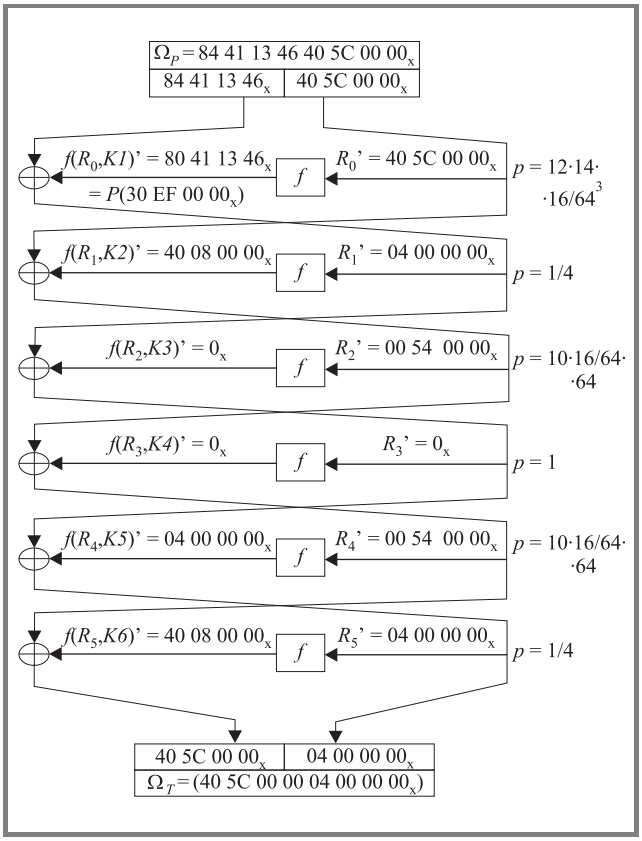


Fig. 3. Six-round characteristic.

Due to the probability of the characteristic only some fraction of pairs will follow it. We know only the difference of

plaintexts and corresponding difference of ciphertexts after 9 rounds so we cannot confirm whether it is a good or bad pair. But we can do some analysis for s-boxes $S2$, $S5$, $S6$, $S7$ and $S8$. If for any s-box obtained input difference does not go to obtained output difference, or equivalently set of suggested subkeys is empty, than we know for sure that analysed pair does not follow the characteristic and should be discarded. Some number of wrong pairs cannot be revealed and discarded in that way. Unfiltered pairs will suggest only wrong subkeys and they will provide only disinformation noise. Also good pairs will suggest a few wrong subkeys and exactly one correct subkey. To distinguish the correct subkey from the noise we count occurrences of all suggested subkeys. The correct subkey should occurs more times than others wrong subkeys. The question is how many pairs we need to analyse to distinguish the correct subkey from the noise. To determine the number of required pairs the S/N ratio is introduced. The parameter estimates the ratio of the number of good pairs equals the number of occurrences of the correct subkey (signal) to the number of occurrence of all subkeys (noise).

4.2. The signal to noise ratio parameter

The S/N ratio of counting scheme is defined as ratio of the number of good pairs and average number of counts of wrong subkeys in counting scheme. In other words the parameter shows how many more times will the correct subkey occur than any other subkey. The formula for the S/N ratio is given below:

$$S/N = \frac{mp}{m\alpha\dfrac{\beta}{2^k}} = \frac{2^k p}{\alpha\beta},$$

where:

p – the probability of the characteristic,

k – the number of bits of counted subkeys,

α – the average number of subkeys suggested by one analysed pair,

β – the ratio of analysed pairs to all pairs, an efficiency of filtration,

m – the number of decrypted pairs.

From the formula it is easy to see that:

- the S/N ratio is independent of the number of pairs used in the attack,
- different schemes of counting based on the same characteristic but counting different number of bits of subkey have different value of the S/N ratio.

The number of good pairs required to find the correct subkey is a function of the S/N ratio parameter. For one s-box of DES we assume $k = 6$, $\alpha = 4$ (average number of 6-bits subkeys suggested by one pair), $\beta = 0.8$ (average percentage of possible difference transitions in s-box).

With these values we can calculate the S/N ratio for the attack on 9 rounds in the following way. We use key recovery procedure for 5 s-boxes in the last round simultaneously; hence we assume following values:

- $k = 5 \cdot 6 = 30$ bits,
- $\alpha\beta = 4 \cdot 4 \cdot 4 \cdot 4 \cdot 4 = 4^5 = 2^{10} = 1024$, the average number of 30-bit subkeys suggested by one analysed pair; for every s-box we have in average four 6-bit subkeys, hence to obtain 30-bit subkey for five s-boxes we have to determine all combinations of these 6-bit subkeys.

The parameters α and β could be determined separately as we will do later in the precise estimation of the S/N ratio, and now we only determine their product like above. It is easier now and as we will show it is also precise.
We know the probability of the characteristic and above values of parameters so we can calculate the S/N ratio now:

$$S/N = \frac{2^{30} \cdot \dfrac{1}{1000000}}{4^5} = \frac{2^{20}}{1000000}$$

$$= \frac{1048576}{1000000} = 1.048576 \approx 1\,.$$

It was assumed in [3], that if the S/N is between 1 and 2, then about 20–40 good pairs are sufficient. If the S/N is high then only a few occurrences of right pairs are needed to uniquely identify a right value of the subkey bits. If the S/N is small the number of required pairs is big and when S/N is less than 1 we never find the correct subkey. In that case the correct subkey occurs more rare than other subkeys in average. Hence the maximum value in counter is not the value of the correct subkey even for huge number of pairs. In that case attack would be impossible.
In our attack the S/N is small but higher than 1, what makes the attack possible at least in theory. We have to determine the number of pairs needed to success of the attack. According to [3] 30 good pairs are sufficient. Good pair appears statistically one time per every 1 000 000 generated pairs, hence about 30 million pairs will be needed to perform the attack and to uniquely identify a right value of 30-bit subkey. Experiments have shown (cf. Section 6) that above number of pairs is too small in general. Sometimes that number is sufficient but it happens too rarely. We would like to determine the number of needed pairs, which is sufficient to uniquely identify a right subkey in all cases. 30 million, it is too small for sure. How many pairs we need and how to determine this number in the clear and faultless way? We will try to answer the question and we will start from precise calculation of the S/N ratio.

4.3. Precise calculation of the signal to noise ratio

To determine the number of pairs needed to success of the attack first we have to calculate the S/N ratio in the most

precise way. If this parameter is much higher than 1 a few (3–4) pairs would be sufficient like in the attack on 4 or 6 rounds DES. But in the case of attack on 9 rounds DES the ratio is only slightly higher than 1, so we have to calculate it very precise. We will of course use the formula given previously:

$$S/N = \frac{2^k p}{\alpha\beta}.$$

Value of k remains unchanged: $k = 30$ but the probability p is now exact value, i.e., $p = \frac{525}{2^{29}}$. We have also assumed different values of α and β., We start from β parameter, which expresses the proportion of the number of analysed pairs to the number of all generated pairs, so it is an efficiency of filtration. Whole filtration we can do in the 3R attack is testing for all five s-boxes and check whether input difference obtained from ciphertexts and the characteristic may cause obtained output difference. If for these input and output differences in appropriate XOR profile it is zero entry then it means this input-output transition is impossible. If it happens even for one s-box out of five we know for sure that this pair is wrong and we discard it. Hence parameter β is a probability that for all five s-boxes simultaneously the input-output differences are possible. In average percentage of non-zero values in XOR profiles of all s-boxes of DES is 0.8. For five s-boxes we may assume $\beta = 0.8^5 \approx 0.32768$, but it is not sufficiently precise for us. We have to look closer at XOR profiles of s-boxes. In [3] percentages of non-zero values are given for all 8 s-boxes. We have determined these percentages with bigger precision. All results are given in Table 1.

Table 1
Percentage of non-zero values in XOR profiles of s-boxes

s-box	Percentage [3]	Precise values
$S1$	79.4	0.794921875
$S2^*$	78.6	0.786132813
$S3$	79.6	0.796875000
$S4$	68.5	0.685546875
$S5^*$	76.5	0.765625000
$S6^*$	80.4	0.804687500
$S7^*$	77.2	0.772460938
$S8^*$	77.1	0.771484375

In our attack we deal only with 5 s-boxes (denoted *) so parameter β is a product of corresponding values from the table and it equals:

$$\beta = 0.288631 < 0.8^5.$$

Than we calculate value of α. Parameter α means the number of subkeys suggested by one analysed pair. It is the average number of different subkeys found by key recovery algorithm for one non-discarded pair. To obtain α we will again use XOR profiles. For individual s-box the average number of found subkeys is equal to average entry in the XOR profile of the s-box. It is equal 64 (the sum of entries in every row) divided by 16 (the number of columns) hence it is 4. But after filtration we do not take impossible transitions in to account. So we should calculate the average only from non-zero entries. The values of α calculated in that way are given in Table 2. As we can see they are different for different s-boxes.

Table 2
The average of non-zero entries in XOR profile of s-boxes

s-box	Average of non-zero values
$S1$	5.031941032
$S2^*$	5.088198758
$S3$	5.019607843
$S4$	5.834757835
$S5^*$	5.224489796
$S6^*$	4.970873786
$S7^*$	5.178255373
$S8^*$	5.184810127

Similarly like for β, α is a product of values for five s-boxes denoted by * and it equals:

$$\alpha = 3547.782689.$$

Product of α and β is equal to $\alpha\beta = 1024 = 4^5$, so previous estimation was also very precise. But the second method is more universal; because it calculates values of α and β separately, however it is also needs more work.
Now we know the values of all parameters so we can calculate S/N again:

$$S/N = \frac{2^{30} \cdot \frac{525}{2^{29}}}{0.288631 \cdot 3547.782689} = \frac{1050}{1024} = 1.025390625.$$

This value is different from previous on the third significant position although we actually have changed only the probability p.
Very interesting thing is that experiments (cf. Section 6) provide different value of β parameter. We have analysed different number of pairs and obtain the number of pairs non-discarded in filtration process. Proportion of these numbers determines β.
From Table 3 we can see that value of the parameter is rather constant. Small differences are consequences of probabilistic behaviour of differential cryptanalysis. It is also slightly different from theoretically obtained value. It may be explained as follows. We generate pairs of plaintexts not actually in random. First plaintext from the pair is generated randomly but second one depends on the characteristic. Than pairs of ciphertexts are not random and differential cryptanalysis can work at all. Theoretical estimation of β was obtained for fully random values. And here this difference appears. About $1/p$ pairs are good and survive filtration for sure, some portion of pairs may follow

the characteristic for a few rounds and have more chances to survive filtration then fully random ciphertexts. Hence in practice more pairs remain after filtration and parameter β is bigger then theoretical estimation made for fully random pairs.

Table 3
Parameter β obtained in experiments

No.	β
1	0.2916292
2	0.2915709
3	0.2915562
4	0.2914597
5	0.2920615
6	0.2915714

Parameter α obtained in those experiments was accurate with its theoretical value.
We put value of β provided by experiments to the formula of the S/N and we have:

$$S/N = \frac{2^{30} \cdot \frac{525}{2^{29}}}{0.2915 \cdot 3547.782689} = 1.015298465\,.$$

Value of the S/N obtained in this way we will use in further considerations.

4.4. The method of determination the number of pairs needed to the attack based on the signal to noise ratio

We have determined as precise as possible the S/N ratio. Now we will use the ratio to estimate the number of required pairs. If the S/N is much bigger then 1 we can assume that even 3–4 good pairs are sufficient to successful attack. But when the S/N is close to 1 like in that case the number of required good pairs is much bigger. In-advance assumption that 20, 30 or 40 good pairs are sufficient must be verified in certain attack. The attack on 9 rounds of DES need more then suggested in [3] 30 or 40 good pairs to uniquely identify the correct key. Sometimes these numbers are sufficient but it happens too rarely. Rough estimation of efficiency of attack with these numbers of pairs is smaller then 50%. So how many pairs are needed to significantly increase this efficiency?
We want to set the number of required pairs to be as small as possible but to be sufficient to uniquely identify the correct key. To recall, found key is a value, which has occurred the most frequently. It means that we search the counter of occurrences of all keys for the maximum value. If a noise is high the maximum value may not correspond to the correct key but some other key. The number of occurrences of the correct key may be second or third or next value in the counter. In that case attack ends with failure. In our attack the S/N ratio is greater than 1 (and it is possible at all) so increasing number of generated and analysed pairs tends to increased number of occurrences of correct key. That number increases faster than average number of occurrences of incorrect key (noise). So the number of pairs to analyse should be sufficiently big to assure that number of occurrences of the correct key will be maximum value in the counter. It means that number will be greater than number of occurrences of all other keys with big probability. Due to the probabilistic nature of our problem values in the counter are different in different experiments. And we can use only average numbers, which are easy to determine. We cannot predict the maximum value of noise or signal in the counter but we know in average how many good pairs were analysed. Also we know that every good pair gives one good key. We know the average value in the counter (level of a noise) as well. Now we want to be sure that number of occurrences of correct key would "stand out of noise". It means it would be greater even by 1 than other values in the counter. Hence we have to set the number of required pairs in order to the expected number of good pairs and occurrences of good key as well be greater than the expected average number in the counter. The expected number of good pairs can be calculated as a product of the number of all analysed pairs and the probability of the characteristic: $m \cdot p$. It is the numerator of the S/N ratio. The expected average number in the counter can be calculated as: $(m \cdot \alpha \cdot \beta)/2^k$ (the denominator of the S/N). Now we find such m that the numerator $(m \cdot p)$ is greater at least by 1 than the denominator. We start with $m = 30\,000\,000$ because we know that value is too small. We assume step 5 million and check successive values of m. Results of our searching are given in Table 4.
The first row gives the number of pairs used in the attack. The second counts the average number in the counter, third – number of good pairs. The fourth row is a proportion of above rows (row 2/row 3), so it is the S/N ratio actually. The last row expresses the difference: row 3 – row 2. As we can see the difference is greater than 1 for $m = 70$ million and we end our search with that value. We believe that in some sense the last row expresses the success of attack with corresponding number of pairs.
Precise value of m can be also calculated by using the S/N ratio. The number of occurrence of good key grows faster than the average number in the counter by factor equals to the S/N ratio. We want the signal be greater by one than noise. We put the nominator to be by 1 greater than the denominator, hence:

$$mp - \frac{mp}{S/N} > 1 \Rightarrow \frac{(S/N-1)mp}{S/N} > 1 \Rightarrow m > \frac{S/N}{S/N-1}\,\frac{1}{p}\,.$$

In that way we obtain the estimation on the number of required pairs to successful attack:

$$m > \frac{S/N}{(S/N-1)p}\,.$$

As we can see, with the growth of the S/N ratio the number m tends to $1/p$.

Table 4
Results of searching for required number of pairs

1.	Number of pairs m	30000000	35000000	40000000	45000000	50000000
2.	Average in counter	28.89462	33.71038743	38.52616	43.34192669	48.15769633
3.	Number good pairs	29.33666	34.22610462	39.11555	44.00499165	48.89443517
4.	The S/N ratio	1.015298	1.015298465	1.015298	1.015298465	1.015298465
5.	Difference	0.44204	0.51571719	0.58939	0.66306496	0.73673884
1.	Number of pairs m	55000000	60000000	65000000	67866654	70000000
2.	Average in counter	52.97346596	57.7892356	62.60500522	65.3660333	67.42077
3.	Number good pairs	53.78387868	58.6733222	63.56276572	66.3660333	68.45221
4.	The S/N ratio	1.015298465	1.015298465	1.015298465	1.015298465	1.015298
5.	Difference	0.81041273	0.884087	0.95776049	1	1.03143

The estimation can be useful in every attack, especially when the S/N ratio is close but greater than 1. Efficacy of attack with that number of pairs is very high (cf. Section 6) but still less than 100%. Due to the probabilistic nature of the problem an experiment in which that number is too small always may happen. But now it will happen rarely. In that case we of course can use more pairs.
In our attack we have:

$$m = \frac{1.015298465 \cdot 2^{29}}{(1.015298465 - 1) \cdot 525} = 67866653.95\,.$$

Hence we need above 67 million pairs. It confirms the results given in the Table 4.
Using the above estimation we can make quite interesting observation. Namely, if we use the estimation with very first calculated value of S/N ratio [3] we will obtain $m > 22074391.17$. It means that for this value of the S/N ratio 30 million pairs would be sufficient.

5. Implementation issues

Implementation and performance of considered attack is possible now thanks to progress in computational power of computers (processor speed, capacity of operational and disc memory). But still there are some problems and restrictions we have to solve. Main problem is a size of memory to store the counter.
We count the number of occurrences of all 30-bit subkeys, so we need the counter, which consists 2^{30} at least 8-bit values. It requires operational memory of size 1 GB. In the attack on 8 rounds of DES the problem was solved by dividing it on two less complicated subproblems [2]. In the case of 9 rounds this solution is impossible. Computer with 1 GB of operational memory is still unavailable due to high costs. The problem was solved on computer with operational memory of size 256 MB by time-memory trade off 256 MB of memory allows on using a counter with 2^{28} elements. The space of 2^{30} was divided on four separate subspaces of size 2^{28} depending on two most significant bits of 30-bit key. Whole process of counting keys is divided on four substages.
In first substage we generate required number of pairs. Than we perform key recovery algorithm and save in temporary file only non-discarded pairs (ciphertexts actually). It requires $2 \cdot \beta \cdot m \cdot 8$ bytes of disc memory and for $m = 70$ million it makes about 324 MB and now it is easily available. In that smaller counter we put only keys with the same two significant bits equal for example 00. In the counter we search for the maximum. If the distinct maximum exists we assume it corresponds with the correct key and we end this stage. The distinct maximum means that it is only one maximum value and the value is significantly greater than any others. In the other case we proceed to next substage. We put into the new counter only keys with the same two significant bits equal now for example 01.
In the second substage we use saved in the first substage non-discarded pairs. We do not perform the filtration again what significantly reducing the time of the substage. Than we again search for the distinct maximum in the new counter taking into account maximum value from the first counter. If the distinct maximum exists we finish with that value of key. In the other case we proceed to third and forth if it is needed. The time of that stage may differ significantly with respect to the number of substages performed until the key was found. In average we perform two substages. That approach makes the issues of required number of pairs very complicated. If the number appeared too small increasing it is very troublesome and we must repeat whole attack actually. As we can see it is very important to set the appropriate number of required pairs at the beginning. From the other hand, the number should be as small as possible to decrease the time of attack.
The second and very unexpected problem, which showed out, was generating the pairs of plaintexts. As we stated we need 70 million pairs, so we have to generate above $2^{26} \cdot 2^3 = 2^{29}$ bytes in random. The period of standard pseudorandom number generator in used programming language (C++) should be 2^{31}, but for the least significant byte it is smaller and it is only 2^{27} (it is a discovered error

Table 5

First results

Number of generated pairs [million]	Number of non-discarded pairs	β	Max in counter	Number of occurrences of correct key	Result
30	8743769	0.2914580	64 (4 times)	63	Failure
35	10202090	0.2914883	77	77	Success
35	10203548	0.2915299	73	59	Failure
35	10205025	0.2915721	81	81	Success
35	10204767	0.2915648	71	61	Failure
35	10203362	0.2915246	70 (3 times)	70	Failure
35	10200564	0.2914447	76	76	Success
35	10204390	0.2915537	73 (2 times)	64	Failure
35	10207181	0.2916337	72 (2 times)	59	Failure
35	10205876	0.2915965	75	48	Failure
35	10201090	0.2914597	73	60	Failure
40	11655712	0.2913928	88	88	Success
40	11658656	0.2914664	92	92	Success
40	11662246	0.2915562	87	87	Success
40	11661319	0.2915330	82	76	Failure
40	11665598	0.2916400	79 (2 times)	64	Failure
40	11662246	0.2915562	84	84	Success
50	14576501	0.2914666	99	99	Success
50	14573329	0.2916178	105	105	Success
50	14580889	0.2916178	98	80	Failure
50	14575285	0.2915057	95	88	Failure
50	14586415	0.2917283	109	109	Success
50	14576447	0.2915289	101	101	Success
50	14579432	0.2915886	94	85	Failure
50	14579003	0.2915801	100	100	Success
50	14575915	0.2915183	95	94	Failure

of compiler!). So we have to use our own pseudorandom number generator with the suitable period, which generates numbers with uniform distribution. We did not need any cryptographically strong generator but only fast one so we used ordinary linear feedback shift register (LFSR) with length 64 bits, what made its period equals to $2^{64} - 1$ bits. That problem is irrelevant from cryptanalytic point of view but we would like to point out that in the case of such huge amounts of data similar problems may completely warp the results of the cryptanalysis.

6. Results

Now we present the results of performance of considered attack. The attack was implemented in C++ language in Borland C++ Builder 5 on a computer with processor Celeron II 1.3 GHz and 320 MB of operational memory. We used implementation of 9 rounds of DES running with speed 3.2 million blocks per second what makes throughoutput 200 Mbit/s.

We start with attacks with too small number of pairs. Table 5 shows the results of attacks with 30–50 million pairs. The first column presents the number of generated pairs. The second expresses the number of pairs, which survive the filtration. The third is a proportion of two previous columns and it is an efficacy of filtration (parameter β). In the next column the maximum value in the counter is given. The number in the parenthesis means how many times the maximum appeared if more than one. The fifth column gives the number of occurrences of the correct key. In the last column the result of the attack (success or failure) is given. The result can be derived from two previous columns.

From the table we can roughly estimate the probability of success of the attack. The probability of success of attack with 30 or 35 million pairs is smaller than 1/3, with 40 or 50 million pairs slightly exceeds 1/2.

Table 6 presents in similar way the results of attacks with 70 million pairs. Additionally the time of performance of substages is given.

Table 6

Main results

Number of all pairs [million]	Number of non-discarded pairs	β	Time of substages [s]				Time of II stage [s]	Time of attack [s]	Max	Number of occurrences of correct key	Result
			1	2	3	4					
70	20411466	0.2915924	7391	4987	4977	–	138	17493	139	139	Success
70	20407887	0.2915412	7395	4979	4983	–	6	17363	144	144	Success
70	20400900	0.2914414	7374	–	–	–	245	7619	127	127	Success
70	20410516	0.2915788	7384	–	–	–	84	7468	134	134	Success
70	20408476	0.2915497	7363	–	–	–	29	7392	128	128	Success
70	20408331	0.2915476	7382	–	–	–	85	7467	145	145	Success
70	20402070	0.2914581	7358	4982	4948	6085	313	23686	121	116	Failure
70	20414410	0.2916344	7376	–	–	–	273	7649	149	149	Success
70	20411673	0.2915953	7384	4945	4955	4951	136	22371	143	143	Success
70	20404332	0.2914905	7360	4935	–	–	72	12367	133	133	Success
70	20405427	0.2915061	7353	4938	–	–	211	12502	132	132	Success
70	20414960	0.2916423	7389	4944	–	–	155	12488	123	123	Success
70	20404748	0.2914964	7391	–	–	–	85	7476	133	133	Success
70	20409296	0.2915614	7353	4932	4927	–	89	17301	135	135	Success
70	20415691	0.2916527	7365	4916	–	–	196	12477	148	148	Success
70	20408939	0.2915563	7455	–	–	–	17	7472	126	126	Success
70	20403624	0.2914803	6792	–	–	–	210	7002	151	151	Success
70	20409608	0.2915658	6804	4729	4707	4710	152	21102	119	118	Failure
70	20404448	0.2914921	6766	–	–	–	232	6998	121	–	Failure
70	20412821	0.2916117	6768	–	–	–	182	6950	144	144	Success
70	20413305	0.2916186	6764	–	–	–	77	6841	152	152	Success
70	20404312	0.2914902	6772	–	–	–	13	6785	149	149	Success
70	20413398	0.2916200	6769	4709	4696	–	28	16202	146	146	Success
Average		**0.2915574**	**7167**	**4892**	**4847**	**5249**	**132**	**11500**	**Succ./fail.**		**20/3**

Lack of given time of any substage means that substage was not necessary because the maximum was found in previous substage. The first substage is always about 1.5 times longer than others. It results from applied procedure of counting keys. In the first stage we generate all pairs, than we perform the filtration and save them. In next substages we only analyse non-discarded pairs what is significantly faster.

As we can see only 3 out of 23 attacks have ended with failure. In the two first cases all four substages were performed, but in any of them distinct maximum was not found, and the correct key occurred more rare than others. In the third case distinct maximum was found but it did not correspond to the correct key. The efficiency of the attack with 70 million pairs we can consider as very high and close to 90% even for a few dozen experiments.

After assuming that 70 million is the proper number of required pairs we can perform a full attack. It means after finding the 30 bits of main key in the first stage we can find remaining 26 bits by exhaustive search of space 2^{26}. It was now a simple and fast task. The column "Time of II stage" in Table 6 gives time (in seconds) of exhaustive search that was needed to find remaining 26 bits of main key. As we can see that search takes not longer than 5 minutes. The column "Time of attack" gives a total time of the attack calculated as the sum of fives values in the previous columns.

In the cases of first two failures the correct key occurred too less times to be found. So all four substages were performed and entire search in the II stage (313 and 152 seconds), but without success. Of course if first 30 bits are wrong we can never adjust the last 26 bits to get the correct key.

The last failure was of different type. In the initial substages (in the first in the certain case) a distinct maximum was found, but it did not correspond to the correct key. Algorithm ended without performing the next substages in which the correct key should be found. That way we did not found the number of occurrences of the correct key. That failure is not based on too less number of analysed pairs, but it comes from extorted dividing the first stage on four

Table 7
Final results

Number of all pairs [million]	Number of non-discarded pairs	β	Time of substages [s]				Time of II stage [s]	Time of attack [s]	Max	Number of occurrences of correct key	Result
			1	2	3	4					
70	20407072	0.2915296	6799	4772	4782	4793	151	21297	124	124	Failure
70	20411849	0.2915978	6847	–	–	–	31	6878	130	130	Success
70	20412699	0.2916100	6837	–	–	–	138	6975	133	133	Success
70	20414234	0.2916319	6842	4934	4926	4940	73	21715	145	145	Success
70	20407731	0.2915390	6838	4907	4932	4928	179	21784	131	141	Success
70	20402279	0.2914611	6834	4919	4933	4917	152	21755	124	117	Failure
70	20409520	0.2915646	6841	–	–	–	233	7074	125	–	Failure
70	20407149	0.2915317	6829	–	–	–	69	6898	156	156	Success
70	20409663	0.2915666	6835	–	–	–	152	6987	156	156	Success
70	20413565	0.2916224	7030	4776	4896	4812	151	21665	121	115	Failure
70	20412273	0.2916049	6827	4797	4805	–	142	16571	128	128	Success
70	20412370	0.2916053	6867	5002	5014	5015	212	22110	122	122	Failure
70	20405958	0.2915137	8277	5406	5118	4886	41	23728	130	130	Success
70	20412912	0.2916130	4704	–	–	–	225	4929	139	139	Success
70	20416821	0.2916689	6740	4699	4695	–	163	16297	144	144	Success
70	20406430	0.2915204	6818	4832	4855	4942	152	21599	123	120	Failure
70	20401766	0.2914548	6827	4809	–	–	226	11862	149	149	Success
70	20409267	0.2915610	6855	–	–	–	71	6926	136	136	Success
70	20409910	0.2915701	6832	4808	4837	4819	69	21365	148	148	Success
70	20411009	0.2915858	6830	–	–	–	205	7035	149	149	Success
70	20406152	0.2915165	6880	–	–	–	109	6989	138	138	Success
70	20407093	0.2915299	7576	5120	5120	4882	68	22766	144	144	Success
70	20399601	0.2914229	6712	4806	4804	4817	146	21285	130	130	Success
70	20410754	0.2915822	6838	–	–	–	255	7093	136	136	Success
70	20406290	0.2915184	7796	4959	5184	–	90	18029	133	133	Success
70	20407788	0.29153983	7571	5119	5119	4895	197	22901	152	152	Success
70	20416450	0.29166357	6722	–	–	–	222	6944	142	142	Success
70	20413332	0.29161903	6840	4937	4929	4943	150	21799	146	146	Success
70	20408688	0.29155269	8576	5337	5115	–	208	19236	141	141	Success
70	20408957	0.29155653	6835	4925	4931	–	244	16935	136	136	Success
70	20408126	0.29154466	6831	4925	4933	4922	220	21831	135	135	Success
70	20407964	0.29154234	6836	4930	4946	4948	46	21706	154	154	Success
70	20408866	0.29155523	6835	4941	–	–	31	11807	128	128	Success
Average		**0.29156041**	**6938**	**4939**	**4944**	**4897**	**146**	**15599**	**Succ./fail.**		**27/6**

substages. In that case we should despite to find suspect key in the initial substages continue with next substages. So the open problem appears: should we always perform all four substages what will increasing time of the attack significantly or like it was done stop after finding the first distinct maximum what is faster but generate above failures?

The way of omitting that problem is fixing a threshold on the maximum value in the counter. If the found maximum is lower than the threshold we will continue with the next substages. As the performed experiments show "the level of noise" which is the maximum number of occurrences of incorrect key does not exceed 125. However the correct key usually occurs (if we reached adequate substage) more often, and even more than 130 times. Hence we can assume that if we find the distinct maximum but lower than 125 that value does not correspond to the correct key and we will continue with next substages. The results of attacks performed with that rule are presented in Table 7.

Introducing the threshold almost eliminated the failures, which comes from dividing problem on substages. Failures, which appeared now, derive from too small number of occurrences of the correct key. In those cases we should generate more pairs to analyse. The exception is the first failure and the failure where the number of occurrences of the correct key is not given. In the first case the maximum value in the counter corresponded to the correct key but it did not exceeded the fixed threshold and was not taken in to account. In that case even without applying the "threshold" rule attack would end with failure, because found maximum was not distinct, the second biggest value was only smaller by 1. In the second case the distinct maximum was found and it exceeded the threshold but it was not the correct key. In that case the threshold should be higher. But the higher threshold may cause more failures of the first (previous) type. Fixing the threshold is very hard and important case. Increasing the threshold will reduce the number of failures of second type but will increase the number of failures of the first type and vice versa. However the small number of total failures in our experiments let us consider that we fixed the threshold correctly.

The efficiency of the attack we can roughly approximate on 80%, 6 failures in 33 tries, and the average time of performance of entire attack was 15 600 seconds, which is about 4 hours and 20 minutes. It is very short time for recovering full 56-bit key of 9-round algorithm.

Acknowledgement

This work has been partly supported by Polish Committee of Science Research project number 0 T00A 020 25 and partly supported by the European Commission under contract IST 2002-507932 (ECRYPT).

References

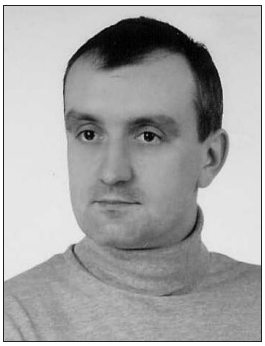
[1] E. Biham, V. Furman, M. Misztal, and V. Rijmen, "Differential cryptanalysis of Q", in *Fast Software Encryption: 8th International Workshop, FSE 2001, Yokohama, Japan, April 2–4, 2001*, M. Matsui, Ed., *Lecture Notes in Computer Science*. Berlin [etc.]: Springer-Verlag, 2002, vol. 2355, pp. 174–186.

[2] M. Misztal, "Praktyczna kryptoanaliza różnicowa algorytmu DES zredukowanego do 8 rund", *Bull. WAT Cryptology Part I*, vol. XLVII, no. 10(566), pp. 125–146, 1999 (in Polish).

[3] E. Biham and A. Shamir, *Differential Cryptanalysis of the Data Encryption Standard*. New York: Springer-Verlag, 1993.

[4] D. Kwiatkowski, "Implementacja i kryptoanaliza wybranych szyfrów blokowych", Warszawa, Wojskowa Akademia Techniczna, Wydział Cybernetyki, 1998, Master thesis (in Polish).

[5] B. Schneier, *Kryptografia dla praktyków. Protokoły, algorytmy i programy źródłowe w języku C*. Warszawa: WNT, 2002 (in Polish).



Michał Misztal was born in 1973 in Kielce, Poland. He got his M.Sc. in 1997 from Faculty of Cybernetics of Military University of Technology (MUT), Warsaw. He has studied on "cryptology" specialty in the individual course. He works as an Assistant in the Institute of Mathematics and Cryptology on Faculty of Cybernetics MUT. He conducts tutorials and lectures on mathematics and linear algebra, but also on profiled by the Institute "cryptology" specialty on such subjects like cryptanalysis of block and stream ciphers, differential and linear cryptanalysis and designing of block ciphers. He is the co-author of handbook entitled "Introduction to Cryptology" and the author of several papers published among others in the bulletin of MUT. He has also given many lectures on scientific conferences devoted to cryptology.

e-mail: mmisztal@wat.edu.pl
Institute of Mathematics and Cryptology
Faculty of Cybernetics
Military University of Technology
S. Kaliskiego st 2
00-908 Warsaw, Poland