Differential Fault Analysis of Secret Key Cryptosystems

Eli Biham
Computer Science Department
Technion – Israel Institute of Technology
Haifa 32000, Israel
biham@cs.technion.ac.il
http://www.cs.technion.ac.il/~biham/

Adi Shamir
Applied Math. and Comp. Sci. Department
The Weizmann Institute of Science
Rehovot 76100, Israel
shamir@wisdom.weizmann.ac.il

Abstract

In September 1996 Boneh, Demillo, and Lipton from Bellcore announced a new type of cryptanalytic attack which exploits computational errors to find cryptographic keys. Their attack is based on algebraic properties of modular arithmetic, and thus it is applicable only to public key cryptosystems such as RSA, and not to secret key algorithms such as the Data Encryption Standard (DES).

In this paper, we describe a related attack, which we call *Differential Fault Analysis*, or *DFA*, and show that it is applicable to almost any secret key cryptosystem proposed so far in the open literature. Our DFA attack can use various fault models and various cryptanalytic techniques to recover the cryptographic secrets hidden in the tamper-resistant device. In particular, we have demonstrated that under the same hardware fault model used by the Bellcore researchers, we can extract the full DES key from a sealed tamper-resistant DES encryptor by analyzing between 50 and 200 ciphertexts generated from unknown but related plaintexts.

In the second part of the paper we develop techniques to identify the keys of completely unknown ciphers (such as SkipJack) sealed in tamper-resistant devices, and to reconstruct the complete specification of DES-like unknown ciphers.

In the last part of the paper, we consider a different fault model, based on permanent hardware faults, and show that it can be used to break DES by analyzing a small number of ciphertexts generated from completely unknown and unrelated plaintexts.

1 Introduction

In September 1996 Boneh, Demillo, and Lipton from Bellcore announced an ingenious new type of cryptanalytic attack which received widespread attention[11,5]. This attack is applicable only to public key cryptosystems such as RSA, and not to secret key algorithms such as the Data Encryption Standard (DES)[17]. According to Boneh, "The algorithm that we apply to the device's faulty computations works against the algebraic structure used in public key cryptography, and another algorithm will have to be devised to work against the non-algebraic operations that are used in secret key techniques". In particular, the original Bellcore attack is based on specific algebraic properties of modular arithmetic, and cannot handle the complex bit manipulations which underly most secret key algorithms.

This type of attack on a tamper-resistant device shows that even cryptographic schemes sealed inside such devices might leak information about the secret key. Earlier papers on this subject, including the papers of Anderson and Kuhn[1], and of Kocher[8], had shown that tamper-resistant devices are vulnerable to several types of attacks including attacks against the protocols, attacks using carelessness of the device's programmers, and timing attacks.

In this paper, we describe a new attack, related to Boneh, Demillo, and Liptons' attack, which we call *Differential Fault Analysis*, or *DFA*, and show that it is applicable to almost any secret key cryptosystem proposed so far in the open literature. In particular, we have actually implemented DFA in the case of DES, and demonstrated that under the same hardware fault model used by the Bellcore researchers, we can extract the full DES key from a sealed tamper-resistant DES encryptor by analyzing between 50 and 200 ciphertexts generated from unknown but related plaintexts. In more specialized cases, as few as five ciphertexts are sufficient to completely reveal the key. The power of Differential Fault Analysis is demonstrated by the fact that even if DES is replaced by triple DES (whose 168 bits of key were assumed to make it practically invulnerable), essentially the same attack can break it with essentially the same number of given ciphertexts.

Differential Fault Analysis can break many additional secret key cryptosystems, including IDEA[9], RC5[19] and Feal[21,16,14,15]. Some ciphers, such as Khufu[13], Khafre[13] and Blowfish[20] compute their S boxes from the key material. In such ciphers, it may be even possible to extract the S boxes themselves, and the keys, using the techniques of Differential Fault Analysis. Differential Fault Analysis can also be applied against stream ciphers, but the implementation might differ in some technical details from the implementation described above. At this point we should note that small differences in the fault models might crucially affect the capabilities and the complexities of the attacks.

Differential fault analysis is not limited to finding the keys of known ciphers: We introduce an asymmetric fault model which makes it possible to find the secret key stored in a tamper-resistant cryptographic device even when nothing is known about the structure and operation of the cryptosystem. A prime example of such a scenario is the SkipJack cryptosystem, which was developed by the NSA, has unknown design, and is embedded as a tamper-resistant chip inside the commercially available Fortezza PC cards. We have not tested this attack on SkipJack, but we believe that it is a realistic threat against some smartcard applications which were not specifically designed to counter it.

Moreover, we show that in most interesting cases we can extract the exact structure of an unknown DES-like cipher sealed in the tamper-resistant device, including the identification of its round functions, S boxes, and subkeys. If the attacker can only encrypt with the tamper-resistant device with a fixed key (e.g., in PIN verifying devices), still the attacker can identify the operation of the cipher with this particular key, which lets him encrypt and decrypt under this key.

The transient fault model used in all these attacks has not been demonstrated in physical experiments, and was questioned by representatives of the smartcard industry. We thus suggest a practical attack based on a different fault model which will hopefully be less controversial. In this model we cut one wire or permanently destroy a single memory cell in the smartcard using a narrow laser-beam. This model allows us to mount a *pure* ciphertext only attack which finds the key with only a few ciphertexts generated from random unrelated unknown plaintexts. The attack is thus applicable even when the smartcard chooses the message it wants to encrypt, and even it if uses counters or random bits to foil differential attacks on related plaintexts.

2 The Attack on DES

Todays' computers are extremely reliable. Still, it is possible for interested parties to intentionally induce faults into some kinds of computations. Although this is almost impossible to do to a remote computer or to a mainframe, it can be done to smartcards.

In many applications (like electronic money, identification systems, or access control) smartcards are used as secure extensions of the host, and enable their owners to apply cryptographic computations without knowing the hosts' secret keys. It is assumed that the smartcards are tamper-resistant, and thus even the owner of a smartcard cannot open it or reverse-engineer it in order to reveal the secret keys kept inside the smartcard.

However, due to the simplicity of the smartcards, and the ability of its owner to control the environment, the owner of the smartcard can force the smartcard to malfunction in many ways, such as changing the power supply voltage, changing the frequency of the (external) clock, and applying radiation of many kinds.

Boneh, Demillo, and Lipton suggested using faults induced by the card owner in order to deduce the private keys in number theoretic public key cryptosystems.

We use the following fault model, similar to the model used by Boneh, Demillo, and Lipton: The smartcard is assumed to have random transient faults in its registers, with some small probability of occurrence in each bit, so that during each encryption/decryption there appears a small number of faults (typically one) during the computation, and each such fault inverts the value of one of the bits, either from zero to one or from one to zero.

More accurately, we assume that during each faulty computation there occurs one (or a few) faults, at random times during the computations, and with random choice of the registers and positions in the registers. Both the bit location and the exact timing of the fault are unknown to the attacker.

This model lets the attacker induce errors at some random position during the DES encryption, i.e., at some random bit of one of the registers in a random round. For the sake of simplicity we assume that the only affected registers are those keeping the right half of the data (the registers keeping the left half do not affect the results till they are exchanged with the right half), i.e., we assume that faults may occur in one of the $16 \cdot 32 = 512$ bits of the right halves of the 16 rounds, and the fault position is unknown to the attacker.

In the attack, the attacker uses the smartcard to encrypt some (possibly unknown) plaintext twice. The attacker compares the two results, and if they differ, he assumes that a fault occurred during exactly one of the two encryptions. As a result, he obtains two ciphertexts derived from the same (unknown) plaintext and key, where one of the ciphertexts is correct and the other is the result of a computation corrupted by a single bit error during the computation.

In the first step of the attack we identify the round in which the fault occurred. This identification is very simple and effective: If the fault occurred in the right half of round 16, then only one bit in the right half of the ciphertext (before the final permutation) differs between the two ciphertexts. The left half of the ciphertext can differ only in output bits of the S box (or two S boxes) to which this single bit enters, and the difference must be related to non-zero entries in the difference distribution tables[4] of these S boxes. In such a case, we can guess the six key bits of each such S box in the last round, and discard any value which disagrees with the expected differences of these S boxes. On average, about four possible 6-bit values of the key remain for each active S box.

If the faults occur in round 15, we can gain information on the key bits entering more than two S boxes in the last round: the difference of the right half of the ciphertext equals the output difference of the F function of round 15. We guess the single bit fault in round 15, and verify whether it can cause the expected output difference, and also verify whether the difference of the right half of the ciphertext can cause the expected difference in the output of the F function in the last round (i.e., the difference of the left half of the ciphertext XOR the fault). If successful, we can discard possible key values in the last round, according to the expected differences. We can also analyse the faults in the 14'th round in a similar way. We use counting methods in order to find the key. In this case, we count for each S box separately[1], and increase the counter by one for any pair which suggests the six-bit key value by at least one of its possible faults in either the 14'th, 15'th, or 16'th round. The right value of the key is expected to be counted more frequently than any wrong value, and thus can be identified.

We have implemented the algorithmic part of this attack on a personal computer. Our analysis program found the whole last subkey given between 50 and 200 ciphertexts, by simulating random single-faults in all the rounds.

This attack finds the last subkey. Once this subkey is known, we can proceed in two ways: We can use the fact that this subkey contains 48 out of the 56 key bits in order to guess the missing 8 bits in all the possible $2^8 = 256$ ways. Alternatively, we can use our knowledge of the last subkey to peel off the last round (and remove faults that we already identified), and analyse the preceding rounds with the same data using

[1]Counting on more than one consecutive S box should reduce the number of required ciphertexts.

the same attack. This latter approach makes it possible to attack triple DES (with 168 bit keys), or DES with independent subkeys (with 768 bit keys).

This attack still works even under more general assumptions on the fault locations, such as faults inside the function F, more than one fault during encryption, or even faults in the key or the key scheduling algorithm.

To check these claims, we have implemented a variant of this attack in which the faults may occur also in the inputs of the F function, rather than only in the right half of the data (i.e., the faults do not affect the following rounds directly), and found that the number of ciphertexts required to find the key is about the same as in the original attack, although the faults may occur in twice as many positions.

If the attacker can induce the faults in a chosen position or a chosen time during encryption, he can improve his results by a large factor. For example, if the attacker can cause the faults to appear uniformly in the last two, three, or four rounds of the DES encryption (rather than in all the 16 rounds), our attack requires only about 10 ciphertexts! If the attacker can choose the exact position of the fault, this number can be further reduced to about 3 ciphertexts.

2.1 Discussion

We described a new attack on ciphers using transient hardware faults. Smartcard designers can try to counter this attack by computing the encryption function twice, and outputting the ciphertext only if the results are identical. This solution is however insufficient: the probability that the same fault occurs during both encryptions may not be sufficiently small. In the attack there are only 512 possible positions for a fault. When computing twice with a single random fault in each of the two encryptions, there is a probability of 1/512 to generate the same fault in both computations. Since the device outputs the doubly-corrupted results, the attacker receives the same data as in the original attack, if he tries 512 as many encryptions. Thus, instead of generating about 200 ciphertexts in the original attack, the device performs about 100000 encryptions.

In many cryptographic implementations, the key scheduling algorithm precomputes all the subkeys in advance, or computes the subkeys from the key every single encryption. In such cases, it is easy to find ciphertext pairs whose differences result only from one faulty subkey bit, when the faults affect the subkeys. In the attack we should assume that the difference is caused by one faulty subkey bit, or by several subkey bits caused by one fault during the key scheduling algorithm. The number of ciphertexts required for such analysis is expected to remain about the same as in the attack we described.

DFA can also be combined with other types of cryptanalytic attacks: for example, when the cipher is computed by software in the device, the faults might affect the program counter or the loop index. In such cases, we can apply more or fewer rounds than required.

In some implementations, the DES key scheduling is applied with two 28-bit shift registers, C and D, as in its original definition. The key rotates every round, and is restored to its original state at the end of encryption, since the total number of shifts during the 16 rounds is 28. If the faults affect the shifts of these registers, then in the following encryptions the key is changed to a *related* key. Related key cryptanalysis[3], or differential related key cryptanalysis[7] might be applied with DFA in such cases. We expect that linear cryptanalysis[12] can also be combined with DFA in some cases (in a similar way to differential-linear cryptanalysis[10]), especially when the identification of the fault position is highly reliable (or when the fault positions might be chosen by the attacker).

Variants of DFA attacks can in some cases also derive the keys of modes of operation in which only part of the ciphertext is known to the attacker. This is similar to the situation studied in [18] for differential cryptanalysis of the CFB mode of operation.

3 Breaking Unknown Cryptosystems

In this section, we introduce a variant of DFA that can find the secret keys of unknown cryptosystems, even if they are sealed inside tamper-resistant devices, and nothing is known about their design. In this attack, we assume a slightly different fault model: the main assumption behind this fault model is that the cryptographic key is stored in an asymmetric type of memory, in which induced faults are much more likely to change a one bit into a zero than to change a zero bit into a one (or the other way around). CMOS registers seem to be quite symmetric, but most types of non-volatile memory exhibit some degree of asymmetry. For example, a one bit in an EEPROM cell is stored as a small charge on an electrically isolated gate. If the fault is induced by external radiation (e.g., ultraviolet light), then the charges are more likely to leak out of the gate than to be forced into the gate.

To make the analysis simpler, we assume that we can apply a low level physical stress to the tamper-resistant device when it is disconnected from power, whose only possible effect is to occasionally flip one of the one bits in the key register to a zero. The plausibility of this assumption depends on numerous physical and technical considerations, which are beyond the scope of this paper.

We further assume that we are allowed to apply two types of cryptographic functions to the given tamper-resistant device: We can supply a plaintext m and use the current key k stored in the non-volatile memory of the device to get a ciphertext c, or we can supply a new n-bit key k' which replaces k in the non-volatile memory.

The cryptanalytic attack has two stages: In the first stage, we keep the original unknown secret key k stored in the tamper-resistant device, and use it to repeatedly encrypt a fixed plaintext m_0. After each encryption, we disconnect the device from power and apply a gentle physical stress. The resultant stream of ciphertexts is likely to consist of several copies of c_0, followed by several copies of a different c_1, followed by several copies of yet another c_2, until the sequence stabilizes on c_f. Since each change is likely to be the result of one more key bit flipping from one to zero (thus changing the current key k_i into a new variant k_{i+1}), and since there are about $n/2$ one bits in the original unknown key k, we expect f to be about n/2, and c_f to be the result of encrypting m_0 under the all-zero key k_f.

In the second stage of the attack, we work our way backwards from the known all-zero key k_f to the unknown original key k_0. Assuming that we already know some intermediate key k_{i+1}, we assume that k_i differs from k_{i+1} in a single bit position. If we knew the cryptographic algorithm involved, we could easily try all the possible single bit changes in a simple software simulation on a personal computer, and find the (almost certainly unique) change which would give rise to the observed ciphertext c_i. However, we do not need either a simulator or knowledge of the cryptographic algorithm, since we are given the real thing in the form of a tamper-resistant device into which we can load any key we wish, to test out whether it produces the desired ciphertext c_i. We can thus proceed deterministically from the known k_f to the desired k_0 in $O(n)$ stages, trying $O(n)$ keys at each stage. The attack is guaranteed to succeed if the fault model is satisfied, and its total complexity is at most $O(n^2)$ encryptions.

This seems to be the first cryptanalytic attack which makes it possible to find the secret key of a completely unknown cryptosystem in polynomial time (quadratic time in our case). It relies on a particular fault model which is stronger than the transient fault model described above, and which has not been experimentally verified so far.

This attack can be extended to asymmetric fault models in which some of the faults change zeroes to ones, but most of the faults change ones to zeroes. In this case, the stream of ciphertexts does not stabilize, and continues to fluctuate between a relatively small number of ciphertexts (generated from low Hamming weight keys). The attack thus requires an additional stage, in which we try to encrypt the plaintext m_0 with randomly chosen low Hamming weight keys. By the birthday paradox, if enough ciphertexts are generated in the first and second stages, we are likely to find a common ciphertext, and thus a common key, k_l. k_l is known and is a result of a biased random walk from the original key k_0, where every two consecutive keys have a single-bit difference. The key k_0 can then be recovered in a similar way to the original attack.

In this attack, there exists a Hamming weight w such that about half of the time the random walk uses keys with Hamming weight smaller or equal to w. A simple calculation shows that this Hamming weight is about $w = \frac{np}{1+p}$, where p is the ratio between the probability of a zero to one fault in a single zero bit and the probability of a one to zero fault in a single one bit. The number of keys with an Hamming weight smaller or equal to w is $N_w = \binom{n}{w} + \binom{n}{w-1} + \ldots + \binom{n}{0}$. We compute $\sqrt{2N_w/n}$ values from the random walk, and encrypt m_0 under about $\sqrt{nN_w}$ random keys with Hamming weight bounded by w. By the birthday paradox we can find with a reasonable probability a common ciphertext produced by a common key k_l.

Numeric example: consider 64-bit keys and a ratio $p = 1/7$ between the probability of a zero to one fault and the probability of a one to zero fault. In this case $w = 8$, and the number of keys with a lower or equal Hamming weight is $N_8 = \binom{64}{8} + \binom{64}{7} + \ldots + \binom{64}{1} + \binom{64}{0} \approx 2^{32}$. Then we compute about 2^{14} values in the random walk, of which about 2^{13} are with Hamming weight smaller or equal to 8. We encrypt m_0 under 2^{19} random keys with Hamming weight smaller or equal to w, and expect to find one common key. The last stage of the attack finding k_0 requires to find an average of 2^{13} keys, each by an average of 32 trial encryption. Thus, the complexity of this attack is 2^{19}.

4 Reconstructing Unknown Ciphers

In this section we show how to reconstruct the full structure of unknown ciphers hidden in tamper-resistant devices. We assume that the attacked cipher is a DES-like cipher, which encrypts by applying some initial permutation and then applying a round function several times. The round function applies some function F to the right half of the data, XORs the result to the left half, and exchanges the roles of the halves. Finally, some final permutation is performed.

In our attack we reconstruct the cipher in several steps. In each step we receive some additional information on the unknown structure of the cipher: we start from the final permutation and continue backwards through the rounds.

Note that the representation of the ciphers is not unique, and thus we cannot identify the exact original definition of the cipher. Instead, we actually find a family of representations, of which the original representation is a member. We can then choose any member of the family as an equivalent representation of the cipher.

In the first step of the attack we study some information on the final permutation: We identify which ciphertext bits come from the right half and which from the left half of the last round, i.e., which bits affect which in the last round. We encrypt several plaintexts several times, and collect pairs of ciphertexts consisting of the real ciphertext of the plaintext and a faulty ciphertext resulting from some fault during encryption of the same plaintext. We identify the faults which occur in the last round (or two) by counting the bits differing between the real ciphertext and the faulty ciphertext. We use the pairs in which the number of differing bits in the ciphertexts is smaller than a threshold (typically about a quarter of the blocksize), in which case, it is almost certain that the fault occurred in the last round, or in the preceding one. Each fault in the last round differs in one bit in the right half and several bits in the left half. Therefore, the bits of the left half differ more frequently than the bits of the right half. We can thus identify the bits of the right half as those which differ in the least number of pairs.

Once we identify the bits of the right half and the bits of the left half, we can observe which bits of the left half are affected by each bit of the right half via the function F in the last round. In DES-like ciphers the F function is composed of S boxes, and each S box takes a few of the bits of the right half, and affects a few of the bits of the left half. We can easily identify the number of S boxes, and the input and output bits of each S box: we just choose all the pairs which differ by only one bit of the right half, and for each such bit we find the set of all the bits of the left half which differ in those pairs. This information suffices to find the number of S boxes, their sizes, and to identify the input and output bits of each S box.

Then, we reconstruct the content of the S boxes, with the specific unknown key mixed into the input of the S boxes and some unknown value mixed into their output (i.e., we can identify the table $T(x)$, where $T(x) = S(x \oplus k) \oplus u$, where k is the (actual) subkey, and u is derived from the right half of the preceding round). This can be done since the pairs give us the difference $T(x) \oplus T(y)$, for two known values x and y. This reconstruction is very effective. For example, if the unknown cipher is DES, we miss information on only $6 + 4 = 10$ bits out of the $64 * 4 = 256$ unknown bits of each S box, and if the unknown cipher is LOKI[6], we miss information on only $12 + 8 = 20$ bits out of the $2^{12} * 8 = 32768$ bits of each S box of LOKI. These missing bits do not reduce the success of the attack since we actually find all the information we need for peeling off the last round: the subkeys are already mixed into the S boxes and the extra constants are counted naturally as parts of the subkeys when analyzing the preceding rounds. This way we can fully analyze the whole cipher, and receive its full description with the specific key mixed into the S boxes.

Till now we identified the S boxes up to XOR with some unknown constants, some of which are subkeys. We can further identify the S boxes and the subkeys by analyzing encryptions under several keys and comparing the differences between the retrieved tables T. In DES and LOKI the key scheduling algorithm and the S boxes can be easily identified by such comparisons. In these ciphers, where the key scheduling actually select key bits into the subkey (rather than making a more complex computation), the selection pattern of the key bits can be identified by analyzing the effect of different keys on the T boxes. Then, the effect of the subkeys on the T boxes can be removed, resulting with the original S boxes and the key scheduling algorithm.

In some ciphers, the function F is not a composition of several S boxes as in DES and LOKI, but may compute more complex operations. In this case we can model the function F by a most general structure: for each possible subkey, the F function is modeled as an arbitrary function. In this case we can view the F function as applying one huge S box. The number of input and output bits of this huge S box is only half the block size, and thus even though the identification of this whole S box requires much work (about 2^{36} in the case of a 16-round 64-bit DES-like cipher), it can still be done for any particular key. The effect of the key on this S box can be removed in most cases after repeating this computation for several keys.

We have implemented major parts of this attack on a personal computer, using DES as the unknown cipher. Our implementation required only about 500 faulty ciphertexts to identify the bits of the right half, and up to 5000 faulty ciphertexts to identify the input and output bits of the S boxes, without reconstructing their actual entries. Their complete reconstruction would require about 10000 faulty ciphertexts. Note that the complexity of this S box reconstruction crucially depends on the size of the S boxes: in DES there are 64 entries in each S box, and thus about 64 faults in the input of an S box in different ciphertexts suffice to reconstruct the S box (except the 10 missing bits). In LOKI there are 4096 entries in each S box, and thus the number of required faulty ciphertexts is much larger. If we view the F function as one large S box, the number of entries of this huge S box is 2^{32}, and thus the number of required faulty ciphertexts in this case is huge, but still practical.

5 Non-Differential Fault Analysis

The main criticism against differential fault analysis was the transient fault model that was claimed to be unrealistic. Although we still believe that an attack based on radiation-induced faults is possible, we decided to devise a more practical fault model that will hopefully be less controversial.

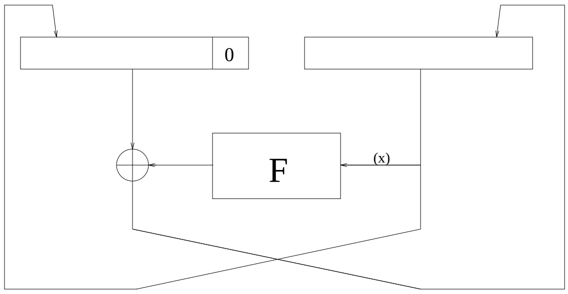

Figure 1. An Iterated Hardware Implementation of DES.

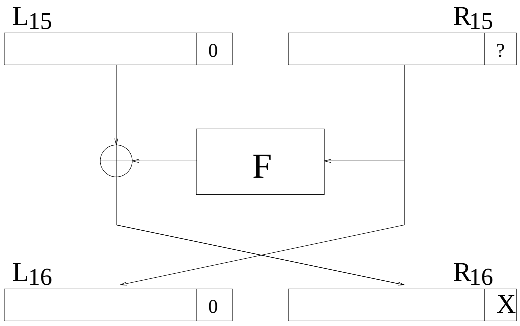

Figure 2. The Last Round in the Iterated Implementation of DES.

The fault model considered in this section assumes that we can cut a wire or destroy a memory cell in a register using a narrow laser beam directed into a carefully chosen location in the silicon. As a result, during computation the values entering the affected location can be considered to be permanently stuck at a fixed value, and can no longer affect the rest of the computation. A similar fault model is considered in [2] where it is used in conjunction with micro-probing to mount other attacks on smartcard-based cryptosystems.

5.1 The Basic Attack

Based on this fault model we develop a *pure* ciphertext only attack, which, unlike all other ciphertext only attacks, does not make any assumption on the statistical distribution of the plaintexts. Moreover, the ciphertext should only be received after the permanent fault had already been generated; non-faulty ciphertexts are not required at all in this attack. Therefore, this attack will still work if faulty smartcards, which already have the appropriate faults due to natural reasons, are given to the attacker.

We describe this attack against a smartcard implementation of DES. We assume that DES is implemented in hardware as a single round, which is used 16 times as is described in Figure 1. We generate a permanent fault in the least significant bit of the left-half register, either by destroying the bit or by cutting the wire which enters or leaves this cell, and assume that the value of this bit is permanently stuck at zero. Figure 2 describes the values computed by this iterated implementation in the last round.

For any ciphertext the least significant bit (LSB) of the right half of the ciphertext (L_{16}) is zero, and the LSB of L_{15} is also zero. The LSB of the output of the F-function of the last round equals the LSB of the left half of the ciphertext (X, since the LSB of L_{15} is zero). This bit is the output of S7 in the last round. The input of S7 is formed by XORing 6 known ciphertext bits with six key bits. We can try all the 64 possible values of the six key bits, and discard any value which does not predict the LSB of the right half of the ciphertext. Each ciphertext is used to discard about half of the remaining key values. Thus, given about six ciphertexts (on whose plaintexts the attacker has no information of any kind) the right value of the six key bits can be identified. The other bits of the last subkey can be found by inducing additional faults, or by enumerating and verifying additional key bits.

5.2 Attacks on Unrolled Implementations

The attack becomes easier if DES is implemented as 16 separate hardware rounds. In this case, a permanent fault induced in one round would not directly affect the other rounds. In this attack, it suffices to destroy the LSB of register L_{15}. In particular, if we destroy the whole L_{15} in this case, we can find the key with only two ciphertexts. We can even reduce the number of ciphertexts required to find 32 bits of the last subkey uniquely to exactly one (rather than only in average) by also destroying the first and sixth input bits entering the S boxes (after they are XORed with the subkey). In this case, the F function becomes reversible, and thus its remaining inputs can be computed uniquely from the outputs.

In an alternative attack, we can destroy all the bits of the subkey registers, except in the first two rounds. Given the ciphertext, remove the last 14 rounds by decrypting with zero subkeys. Equivalently, we can destroy the outputs of the F-function in rounds 3-16, so that the outputs become zero, or even cut the cipher after the 14'th round. In all these cases, the problem is reduced to attacking a two-round DES, which can be done using one ciphertext.

We can even attack the independent-key variant of any iterated cipher. We simply remove a round at a time and prepare the required encrypted data for later analysis. After all the rounds are removed, we find the subkey of the last removed round using the prepared data, and use this found subkey in attacking the second-to-last removed round, etc., till we find all the subkeys. Note that we get two equations on each F-function from two consecutive removed rounds, and thus we get sufficient information for finding the 48-bit subkeys, although each equation leaks only 32 bits.

5.3 Additional Attacks on Iterated Implementations

All the above attacks find the key bits *horizontally*, i.e., a subkey at a time. In iterated hardware implementations we can do the following *vertical* attack which finds the first bits of all the subkeys, then the second bits of all the subkeys, etc. In this attack we first permanently destroy the plaintext to be zero, and encrypt the zero plaintext under the unknown key. In the main step of the attack, we cut the wire leaving the last bit of the subkey in the iterated hardware implementation of the rounds, and encrypt the zero plaintext under the resultant subkeys. Then, we cut the second-to-last wire, etc. When all the subkey bits had been cut, we get the zero plaintext encrypted under the zero subkeys. We can now find the first bits of the subkeys (total of 16 bits in a 16-round cipher) by trying all their possible values (2^{16} trials), and comparing the resulting ciphertexts to the last received ciphertext. Then, we try the values of the second bits of the subkeys, etc, till we find all the bits of the subkeys. For DES in particular, we can use the fact that the key is divided into two halves. We can cut the first 23 bits of the subkey before they are XORed to the right half of the data. Then, we cut the 24'th bit. Then again we cut the next 23 bits. Given the ciphertexts of the destroyed (i.e., 0) plaintext before and after each of these events (four ciphertexts), we can exhaustively search for the 16 key bits which still affect the computation after the last event. When found, we can complete the 12 key bits of the right half of the key, and then search for the 16 key bits of the right half which affect the encryption after the first event. Finally, we complete the remaining 12 key bits.

An additional attack against iterated implementations of DES ignores the data rather than the key. In this attack we cut the 32-bit data input of the F-function (this position is denoted by (x) in Figure 1), such that the output of the F-function becomes dependent only of the subkey and independent of the plaintext. If the actual plaintext is unknown, we also destroy the plaintext register, i.e., set it to zero permanently. The key scheduling algorithm of DES divides the key into two halves, each of them affects only four S boxes in each round (S1–S4, and S5–S8). As a result, 32 bits of the ciphertext depend only on one 28-bit half of the key, and the other 32 bits of the ciphertext depend only on the other 28-bit half of the key. Thus, given the ciphertext of the zero plaintext (or similarly of any plaintext) under the modified cipher, we can easily search for the two halves of the key (2^{28} trials for each, each trial costs about 1/4 encryption).

5.4 Remark

Note that in this model it is easy to apply a chosen plaintext attack without choosing any plaintext — we just destroy the plaintext register. In such a case, even a fault tolerant design in which the smartcard encrypts the plaintext several times and compares the results, will not detect any difference between the ciphertexts.[2] Moreover, if the verification is done by decrypting the ciphertext, the result register can also be destroyed, and thus will always be the same as the plaintext register.

6 Acknowledgements

We would like to gratefully acknowledge the pioneering contribution of Boneh, Demillo, and Lipton, whose ideas were the starting point of our new attack.

[2]Designers should be very careful with such designs, since under some fault models, the comparison of the results might leaks information about the key, which wouldn't be leaked otherwise.

References

[1] Ross Anderson, Markus Kuhn, *Tamper Resistance - a Cautionary Note*, proceedings of the Second Usenix Workshop on Electronic Commerce, pp. 1–11, November 1996.

[2] Ross Anderson, Markus Kuhn, *Low Cost Attacks on Tamper Resistant Devices*, proceedings of the 1997 Security Protocols Workshop, Paris, April 7–9, 1997.

[3] Eli Biham, *New Types of Cryptanalytic Attacks Using Related Keys*, Journal of Cryptology, Vol. 7, No. 4, pp. 229–246, 1994.

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

[5] Dan Boneh, Richard A. Demillo, Richard J. Lipton, *On the Importance of Checking Cryptographic Protocols for Faults*, Lecture Notes in Computer Science, Advances in Cryptology, proceedings of EUROCRYPT'97, pp. 37–51, 1997.

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

[7] John Kelsey, Bruce Schneier, David Wagner, *Key-Schedule Cryptanalysis of IDEA, G-DES, GOST, SAFER, and Triple-DES*, Lecture Notes in Computer Science, Advances in Cryptology, proceedings of CRYPTO'96, pp. 237–251, 1996.

[8] Paul C. Kocher, *Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems*, Lecture Notes in Computer Science, Advances in Cryptology, proceedings of CRYPTO'96, pp. 104–113, 1996.

[9] Xuejia Lai, James L. Massey, Sean Murphy, *Markov Ciphers and Differential Cryptanalysis*, Lecture Notes in Computer Science, Advances in Cryptology, proceedings of EUROCRYPT'91, pp. 17–38, 1991.

[10] Susan K. Langford, Martin E. Hellman, *Differential-linear cryptanalysis*, Lecture Notes in Computer Science, Advances in Cryptology, proceedings of CRYPTO'94, pp. 17–25, 1994.

[11] John Markoff, *Potential Flaw Seen in Cash Card Security*, New York Times, September 26, 1996.

[12] Mitsuru Matsui, *Linear Cryptanalysis Method for DES Cipher*, Lecture Notes in Computer Science, Advances in Cryptology, proceedings of EUROCRYPT'93, pp. 386–397, 1993.

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

[14] Shoji Miyaguchi, *FEAL-N specifications*, technical note, NTT, 1989.

[15] Shoji Miyaguchi, *The FEAL cipher family*, Lecture Notes in Computer Science, Advances in Cryptology, proceedings of CRYPTO'90, pp. 627–638, 1990.

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

[17] National Bureau of Standards, *Data Encryption Standard*, U.S. Department of Commerce, FIPS pub. 46, January 1977.

[18] Bart Preneel, Marnix Nuttin, Vincent Rijmen, Johan Buelens, *Cryptanalysis of the CFB Mode of the DES with a Reduced Number of Rounds*, Lecture Notes in Computer Science, Advances in Cryptology, proceedings of CRYPTO'93, pp. 212–223, 1993.

[19] Ronald L. Rivest, *The RC5 Encryption Algorithm*, proceedings of Fast Software Encryption, Leuven, Lecture Notes in Computer Science, pp. 86–96, 1994.

[20] Bruce Schneier, *Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish)*, proceedings of Fast Software Encryption, Cambridge, Lecture Notes in Computer Science, pp. 191–204, 1993.

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