Differential Fault Analysis
Eltayeb S. Abuelyaman and Balasubramanian Devadoss
School of Information Technology
Illinois State University
Normal, IL, 61790
Esabuel@ilstu.edu

Abstract: Commercial ventures and financial institutions have proposed and are relying upon smartcards and other security processors as a method for storing and transacting electronic currency. As users begin to accept electronic wallets as a viable option for storing their assets, the security community has placed these devices under closer scrutiny. The idea of using computational faults to break tamper resistant cryptographic devices has been recently highlighted in lot of researches. Biham and Shamir named this form of attack as Differential Fault Analysis (DFA). This paper summarizes the existing research on DFA, describes and analyzes a differential attack on various algorithms like DES, RSA, IDEA, RC5, DSA and other vulnerable ciphers. This paper also discusses some of the ways to prevent the Differential Fault Analysis.

0. Introduction
Side-channel attacks occur when an attacker is able to use some additional information leaked from the implementation of a cryptographic function to cryptanalyze the function. Given enough side-channel information, it is trivial to break a cipher. An attacker who can, for example, learn every input into every S-box in every one of DES's rounds can trivally calculate the key. Differential Fault analysis (DFA) fall under side channel attacks. The purpose of this paper is to outline the Differential Fault Analysis (DFA) method of attacking cryptographic algorithms. This paper is intended to provide a survey of DFA in terms of the how the attack is accomplished and which algorithms are vulnerable to this form of attack. The document is divided into four parts the first part provides an introduction to fault analysis. The second deals with differential fault analysis technique. The third part outlines the attacks on various algorithms. The fourth part looks at the ways that this attack can be avoided.

1. Overview of Fault Analysis
Fault Analysis relates to the ability to investigate ciphers and extract keys by generating faults in a system that is in the possession of the attacker, or by natural faults that occur. Faults are most often caused by changing the voltage, tampering with the clock, or by applying radiation of various types. The attacks are based on encrypting the same piece of data (which are not necessarily known to the attacker) twice and comparing the results. A one-bit difference indicates a fault in one of the operations. Now a short computation can be applied for DES
for example to identify the round in which the error has occurred. A whole set of operations can be carried out to recover the DES sub-key which is the sub-key of the last round. When this sub-key is known the attacker can either guess the missing 8 bits or simply peel off the last round for which he knows the sub-key and perform the attack on a reduced DES.

Another type of fault analysis is the non-Differential Fault analysis, but this is based on causing permanent damage to devices for the purpose of extracting symmetric keys. It must be mentioned that a trait of such attacks is that they do not require correct cipher texts. This leads to the attacker being able to make use of natural faulty units, without himself tampering with them. [1]

Types of Faults

Transient faults Consider a certification authority (CA) that is constantly generating certificates and sending them out to clients. Due to random transient hardware faults the CA might generate faulty certificates on rare occasions. If a faulty certificate is ever sent to a client, that client will be able to break the CA’s system and generate fake certificates. Note that on various systems, a client is alerted when a faulty certificate is received.

Latent faults Latent faults are hardware or software bugs that are difficult to catch. As an example, consider the Intel floating point division bug. Such bugs may also cause a CA to generate faulty certificates from time to time.

Induced faults When an adversary has physical access to a device she may try to purposely induce hardware faults. For instance, one may attempt to attack a tamper-resistant device by deliberately causing it to malfunction. The erroneous values computed by the device enable the adversary to extract the secret stored on it. [2]

2. Differential Fault Analysis (DFA)

In 1996, a new attack on cryptographic devices was proposed by researchers at Bellcore. This attack depends on introducing errors into key-dependent cryptographic operations through physical intrusion. Soon after, the initial Bellcore work which focused on public-key techniques was extended and applied to secret-key encryption techniques. It also motivated a series of discussions on the capabilities of secure hardware as a means of keeping the details of certain cryptographic algorithms confidential, and a variety of different threat models have now been considered as a result of their work. The reliance of many security systems on the use of secure hardware or secure processing makes a full evaluation of the potential of fault analysis very important. For developers and users alike an increased awareness of the threat posed by new and novel methods of cryptanalysis allows the development of more secure cryptographic implementations. In this note we will summarize these recent results and in particular we will assess their practical significance when applied to RSA and DES. [3]

3. Differential Fault Analysis on Various Algorithms

In the next subsection we will discuss Differential Fault Attack (DFA) on DES.
3.1. **DFA on DES** The attack follows the Bellcore fundamental assumption that by exposing a sealed tamperproof device such as a smart card to certain physical effects (e.g., ionizing or microwave radiation), one can induce with reasonable probability a fault at a random bit location in one of the registers at some random intermediate stage in the cryptographic computation. Both the bit location and the round number are unknown to the attacker.

It is further assumed that the attacker is in physical possession of the tamperproof-device, so that he can repeat the experiment with the same cleartext and key but without applying the external physical effects. As a result, he obtains two ciphertexts derived from the same (unknown) cleartext 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. For the sake of simplicity, we assume that one bit of the right half of the data in one of the 16 rounds of DES is flipped from 0 to 1 or vice versa, and that both the bit position and the round number are uniformly distributed. In the first step of the attack the round in which the fault occurred is identified. If the fault occurred in the right half of round 16, then only one bit in the right half of the ciphertext 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 of these S boxes. In such a case, the six key bit of each such S box in the last round can be guessed, and any value which disagrees with the expected differences of these S boxes discarded (e.g., differential cryptanalysis). 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 (e.g., 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 analyze the faults in the 14th 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, 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 14th, 15th, or 16th round. This attack finds the last sub-key. Once this sub-key is known, we can proceed in two ways: We can use the fact that this sub-key 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 sub-key to peel up the last round (and remove faults that we already identified), and analyze the preceding rounds with the same data using the same attack. This latter approach makes it possible to attack triple DES (with 168 bit keys), or DES with independent sub-keys (with 768 bit keys). [5]

3.2. **Differential Fault Analysis on RSA**

Direct attacks on the famous RSA cryptosystem seem to require that one has to factor the modulus. **The attack on RSA algorithm is as follows:**

Let $n$ be the product of two primes $p$ and $q$ in RSA, $e$ the public exponent which is publicly known and
Let \( M \) be a plaintext, then the corresponding ciphertext is \( C = M^e \mod n \). Denote the binary representation of the private exponent as
\[
d = d(t-1) \parallel d(t-2) \parallel ... \parallel d(i) \parallel ... \parallel d(1) \parallel d(0),
\]
where:
\( d(i) \), takes value 1 or 0, is the \( i \)th bit, \( t \) is the number of bits of \( d \).
\( x \parallel y \) denotes concatenation of \( x \) and \( y \).

Further, we denote \( C(0) = C \), \( C(1) = C^2 \mod n \), \( C(2) = C^{(2^2)} \mod n \), ..., \( C(t-1) = C^{(2^{(t-1)})} \). Given \( C \) and \( d \), the corresponding plaintext \( M \) can be expressed as
\[
M = (C(t-1)d(t-1))(C(t-2)d(t-2))...(C(i)d(i))...(C(1)d(1))(C(0)d(0)) \mod n.
\]

**Attack 1:**
Suppose that one bit in the binary representation of \( d \) is changed from 1 to 0 or vice versa, and that the faulty bit position is randomly located. An attacker arbitrarily chooses a plaintext \( M \) and computes the ciphertext \( C \). He then applies external physical effects to the tamperproof device and at the same time asks the device to decrypt \( C \). Assuming that \( d(i) \) is changed to its complement \( d(i)' \), then the output of the device will be
\[
M' = (C(t-1)d(i-1))(C(t-2)d(i-2))...(C(i)d(i))...(C(1)d(1))(C(0)d(0)) \mod n.
\]
Since he now possesses both \( M \) and \( M' \), he can compute
\[
M'/M = C(i)^{d(i)/C(i)^{d(i)}} \mod n.
\]
If \( M'/M = 1/C(i) \mod n \), then \( d(i) = 1 \), and if \( M'/M = C(i) \mod n \), then \( d(i) = 0 \).
The attacker can pre-compute \( C(i) \) and \( 1/C(i) \mod n \) for \( i = 0, 1, ..., t-1 \), and compares \( M'/M \mod n \) to these values in order to determine one bit of \( d \). He repeats the above process using either the same plaintext/ciphertext pair or using different plaintext/ciphertext pairs until he finds enough information to obtain \( d \).

**Attack 2:**

For the sake of simplicity, here we assume that in decrypting a ciphertext, the tamperproof device first computes the data sequence \( C(i) \) and then computes
\[
M = (C(t-1)d(t-1))(C(t-2)d(t-2))...(C(i)d(i))...(C(1)d(1))(C(0)d(0)) \mod n.
\]
Suppose that the one bit error is induced in \( C(i) \), \( i = 0, 1, ..., t-1 \).
Suppose that the one bit error is in \( C(i) \), we denote the corrupted value as \( C(i)' \). Then the output from the tamperproof device is
\[
M' = (C(t-1)d(i-1))(C(t-2)d(i-2))...(C(i)'d(i))...(C(1)d(1))(C(0)d(0)) \mod n.
\]
The attacker computes \( M'/M = C(i)'d(i)/C(i)d(i) \mod n \) (note that \( d(i) \) in this ratio must be 1). He can compute all possible \( C(i)'/C(i) \mod n \) values in advance and store them somewhere. The attacker compares all these values with \( M'/M \mod n \). Once a match is found, he knows \( i \) and then knows that \( d(i) \) is 1.
The above procedure is repeated until enough information is obtained to determine \( d \).

The two examples show that one bit fault at certain location and time can compromise the private key.
This compromise gradually increases as the above procedures are repeated using one or multiple pairs of M and C.

**Attack to RSA with Chinese Remainder Algorithm:**

The signature S of a message M equals \( M^d \mod n \) and thus \( S^e \mod n \) is again equal to M \( \mod n \). It is well known that S can be computed by computing \( u = M^d \mod p \) and \( v = M^d \mod q \), and by combining u and v using the Chinese Remainder algorithm. If a fault occurs in the course of the computation of the signature, the resulting value, denoted as \( S' \), will most likely not satisfy \( M = S'^d \mod n \). If, however, the fault occurred only during the computation of say, u, and if v and the Chinese Remainder Algorithm were carried out correctly, then the resulting faulty signature \( S' \) satisfies \( S'^e = M \mod q \) but the same congruence \( \mod p \) does not hold. Therefore, q divides \( S'^e - M \) but p does not divide \( S'^e - M \), so that a factor of n may be discovered by the recipient of the faulty signature \( S' \) by computing the greatest common divisor of n and \( S'^e - M \).

This attack is very powerful and it requires only one faulty signature. On the other hand, it applies only to the case when RSA is implemented based on the Chinese Remainder Algorithm. [4]

### 3.3. Attacks on IDEA

The timing attack on IDEA uses three different group operations on pairs of 16-bit numbers, namely:

- Bit-by-bit exclusive OR of two 16-bit (numbers \( 0 \ldots (2^{16} +1) \))
- Addition of integers modulo \( 2^{16} \) (numbers \( 0 \ldots (2^{16} +1) \))
- Multiplication of integers modulo \( 2^{16} +1 \) (numbers \( 1 \ldots 2^{16} \)). The number 0 of the preceding two operations is treated as \( 2^{16} \) here.

Multiplication of two non-zero numbers modulo \( 2^{16} +1 \) can easily be carried out by first computing the 32-bit "normal" product and then subtracting the high significant 16 bits from the low significant bits (and adding \( 2^{16} +1 \) when the difference is negative). Mathematically speaking:

\[
ab \mod (2^{15} +1) = ((ab \mod 2^{15}) - (ab \div 2^{15})) \mod (2^{15} +1)
\]

Multiplication by 0 (or, equivalently, by 16 2) represents a special case here. Since

\[
2^{16} \mod (2^{16} +1) = -1
\]

We have

\[
2^{16} k \mod (2^{16} +1) = -k = 2^{16} +1 - k
\]

i.e., a simple subtraction.

Therefore, when doing the multiplication, we first check whether one of the factors equals 0 and then apply the appropriate computation.

The subtraction in the case when one of the factors is 0 takes much less time than the multiplication in the "normal" case. It is precisely this property of different computation times which the attack makes use of. In a first step, the multiplicative subkeys of the output transform are attacked. This is done by generating many plaintext-ciphertext pairs using a
randomly chosen key to start with. In order to find the second multiplicative subkey of the output transform, the ciphertexts are grouped into $2^{16}$ groups ordered by the last 16 bits of the ciphertext and the computation times measured. Normally there will be one group which shows a lower average computation time than the other $2^{16}$ groups. This is the group that has a zero input to the last multiplier, and from the known output and input the corresponding subkey may be computed. The first multiplicative subkey of the output transform is determined in a similar way.

In the following steps, the additive subkeys of the output transform and the first multiplicative subkey of the MA box are attacked, giving a total of 80 bits of subkey. The remaining 48 bits may then be determined by brute force. The important point is that, in all steps, use is being made of the fact that multiplication time differs when one of the factors is zero. Similarly, by looking at the power consumption of the IDEA engine, one can determine whether one of the inputs to a multiplier is zero and exploit this fact in order to determine the key.\[8\]

### 3.4. Attack on DSA

In the DSA, to generate a private and public key pair, we first choose a prime number $p$ such that $p = zq+1$ for a reasonably large prime $q$. We then compute $g \equiv b^{(p-1)/q}$ mod $p$, where $b$ is any number less than $p$ such that $(b^{(p-1)/q}$ mod $p)$ is greater than 1. The signer’s private key is $x$, a random number less than $q$, and the public key is $y \equiv g^x$ mod $p$, $g$, $p$, $q$).

To sign a message $m$, the signer first picks a random $k$ that it is less than $q$. Then she computes

$$w \equiv g^k \text{ mod } p \text{ mod } q \quad \text{and} \quad s \equiv (e+wx)/k \text{ mod } q,$$

where $e=\text{h}(m)$ with $\text{h}$ being a secure one way hash function that outputs a number less than $q$. The signature is the pair $w$ and $s$. To verify the signature, the verifier confirms that

Where $u = 1/s \text{ mod } q$.

The attacker applies external physical effects to the tamper resistance device and at the same time asks the device to sign a message $m$. During the process of calculating $s$, we assume that the $i$th bit of $x$ is changed from $x_i$ to its complement $x_i$. Let $x'$ denote the corrupted $x$ due to the flip of $x_i$ then the outputs of the device will be

$$w \equiv g^k \text{ mod } p \text{ mod } q \quad \text{and} \quad s' \equiv (e+wx')/k \text{ mod } q$$

Using $w$, $u' = 1/s'$ mod $q$, $m$, and the signer’s public key $(y, p, g, q)$, the attacker can compute $e=\text{h}(m)$ and

$$T \equiv g^{(u' e \text{ mod } q)} y^{(u' w \text{ mod } q)} = g^{(u' (e+wx)\text{ mod } q)} \text{ mod } p \text{ mod } q.$$  

Let $R_i \equiv g^{(u' w2^i \text{ mod } q)} \text{ mod } p \text{ mod } q$ for $i=0,1,2,3,\ldots,t-1$. Then we have

$$T/R_i \equiv g^{(u' e+w(x-2^i)) \text{ mod } q)} \text{ mod } p \text{ mod } q.$$  

It's easy to show that:

$$T/R_i \equiv w \text{ mod } p \text{ mod } q,$$  

if $x_i=0$ $T/R_i \equiv w \text{ mod } p \text{ mod } q$, if $x_i=1$

So by iterating through different $i$ and matching $w$ with the $T/R_i \text{ mod } p \text{ mod } q$ and $T/R_i \text{ mod } p \text{ mod } q$, the attacker can discover the value of $x_i$.\[8\]
3.5. **Attack on RC5** There may be cases when we can learn the internal flags states of processors. For example, we may be able to determine the state of carry flag after each half-round of RC5. Recall that an RC5 round looks like

\[ R_i = (R_{i-1} \oplus L_{i-1}) \ll L_{i-1} + S_{2i-1} \\
L_i = (L_{i-1} \oplus R_i) \ll R_i + S_{2i} \]

Where \( S_i \) denote the rounds subkeys, \((L_0,R_0)\) is the original plaintext, and \((L_r,R_r)\) is the output of the last round. There is one final transformation

\[ R_{r+1} = (R_r \oplus L_{4r}) \ll L_r + S_{2r+1} \\
L_{r+1} = L_r \]

Before we obtain the resulting ciphertext \((L_{r+1},R_{r+1})\) (this last information makes the encryption and decryption process identical).

RC5 is a variable width cipher usually the width is set so that \( R_i \) (resp. \( L_i \)) fits into one register. Therefore the addition performed may cause the carry flag to be set and we denote \( C_{2i}(C_{2i}) \) the value of the carry flags after the addition computing \( R_i \) (resp. \( L_i \)).

With the high probability, an attacker can reconstruct \( S_{2r-1} \) given several ciphertext pairs \((L_{r+1,j}, R_{r+1,j})\) and the corresponding carry flags \( c_{2r-1,j} \). Once an attacker determines \( S_{2r+1} \), they can strip off the last half round and reapply our attack (reversing left and right).

Each of the two possible values of the msb(SK\(_{2r-1}\)) lead to exactly one value of SK\(_{2r+1}\). Hence an attacker could simply guess the most significant bit of the SK\(_{2r+1}\) for all \( i \leq r \), for each of the subkeys and perform a few trial decryption. If \( r \) is small (say 32), this is certainly feasible. However, in the event that \( r \) is too big, we must attempt to find msb(SK\(_{2r-1}\)) by other means[10].

3.6. **Other Vulnerable Ciphers** Differential Fault Analysis can break many additional secret key cryptosystems, including SKIPJACK and Feal. Some ciphers, such as Khufu, Khafre and Blowfish 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 by some technical details from the implementation described above for DES. [5]

3.7. **DFA on Unknown Cryptosystems.**

The next step of this attack is to break unknown systems. Biham and Lipton proposed a method of doing this.

The cryptanalytic attack has two stages:

1. In the first stage of the attack, the original unknown secret key \( k \) was stored in the tamperproof device, and used to repeatedly encrypt a fixed cleartext \( m_0 \). After each encryption, the device was disconnected from power and a gentle physical stress applied. The resultant stream of ciphertexts was 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 stabilized on \( c_f \). Since each change is likely to be the result of one more key bit flipping from 1 to 0 (thus changing the current key \( k_i \) into a new variant \( k_{i+1} \)), and since there are about \( n/2 \)
1 bits in the original unknown key k, f was expected to be about n/2, and c_f to be the result of encrypting m_0 under the all-zero key k_f.

2. The second stage worked backwards from the known all-zero key k_f to the unknown original key k_0. Assuming that some intermediate key k_{i+1} was already known, an assumption that k_i differs from k_{i+1} in a single bit position was made. If the cryptographic algorithm involved was known, all the possible single bit changes could be tried in a simulation and the (almost certainly unique) change that would give rise to the observed ciphertext c_i found. The simulator or knowledge of the cryptographic algorithm is not needed, since the real thing is given in the form of a tamperproof device into which any key can be loaded to test whether it produces the desired ciphertext c_i. This is done starting form 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^3) encryptions. [5]

4. Preventing Differential Fault Analysis

Side-channel attacks like differential fault analysis can be defeated by carefully designing the software/hardware to either reduce the amount of side-channel information that leaks or make the leakage irrelevant.

Denying an attacker the ability to monitor the internal states can defeat processor flag based side-channel attack on RC5 encryption algorithm and Hamming weight based side-channel attack against Data Encryption Standard (DES) encryption algorithm. Concurrent Error Detection (CED) followed by suppression of the corresponding output has been suggested as an approach to tolerate fault-based side-channel cryptanalysis – On detecting a faulty computation the key is protected by suppressing the cipher text. CED can be performed by straightforward duplication of encryption (decryption) hardware and comparison or by use of spares.[7]

5. Conclusion:

In this paper we discussed about the basics of Fault analysis, different fault analysis technique, attack on various crypto algorithms such as DES, RSA, IDEA, RC5, DSA and at last we discussed about the various ways to prevent the differential fault analysis. The attack can and may already be prevented by well known hardware implementation techniques, and can also be prevented by simple modifications to the cryptographic processing. However the significance of these attacks, and their relevance to high security applications, should not be overlooked.

6. References:


