6.857 Computer & Network Security Final Project: Differential Fault Analysis

Jered J. Floyd (jered@mit.edu)
Kevin E. Fu (fubob@mit.edu)
Peter Sun (petersun@mit.edu)
Instructor: Ron L. Rivest

December 19, 1996

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 several publications. Biham and Shamir named this form of attack Differential Fault Analysis (DFA). This paper summarizes the existing research on DFA, describes and analyzes a differential attack on RC5 that requires only 32 unique faults per partial round, and introduces a practical implementation for mounting such an attack.

1 Introduction

The origins of the threat model behind Differential Fault Analysis (DFA) can be traced back to Anderson and Kuhn's paper on tamper resistance. The first results of their findings were presented at the Isaac Newton Institute, Cambridge, in June of 1996, but withheld from publication due to agreements with the manufacturer of one of the security processors described in the paper [1] [2]. In the report, Anderson and Kuhn outline various methods of compromising smartcards and other security devices and warn manufacturers of the existing

weaknesses. Due to economic considerations, most of today's electronic wallets are constructed with little protection against a concerted physical assault; even attackers with few resources and limited sophistication can cause operational anomalies or can take advantage of known weaknesses in these cards.

A typical smartcard consists of an 8-bit microprocessor along with several memory components such as ROM, RAM and EEPROM. Both passive and active attack techniques can induce random hardware faults. For example, some chips are known to be susceptible to power and clock transients. Other faults may be due to error, generated from the right combination of inputs and operations on those inputs. An attacker can usually create a fault by operating the card in a timed and repeatable fashion. A more aggressive technique against recent, more securely manufactured devices involve bombarding the cards with UV light, strong magnetic pulses, or ionizing radiation to induce malfunction.

Anderson and Kuhn's report validates the practicality of chip-level extraction of cryptographic key material. The threat model assumes that attackers have possession of one or more of the target devices and have the necessary capabilities to cause transient faults (i.e., faults that only affect the data in the current state and do not otherwise effect the device). These assumptions imply that the attacker can input a predefined plaintext and derive different ciphertexts by causing single bit flips in the registers holding the key or by deleting specific operations from the program counter during computation. These corrupted ciphertexts are then compared with the correct outcome, and difference vectors are generated. The differences are analyzed in accordance with the structure, and implementation of the algorithm. Given enough faults, the key is recreated.

This paper achieves three goals. First, it presents a chronological overview of the research on DFA for both public- and private-key cryptosystems with known and unknown algorithms. Second, it details an original differential attack on the RC5 algorithm and demonstrates key extraction with only 32 faults per partial round. Third, it outlines a practical implementation for carrying such an assault on a device encoded with RC5.

Section 2 details the various research efforts on fault-based cryptanalysis. Section 3 presents an attack on RC5 and includes an overview, a description, an analysis, and some proposed workarounds. Section 4 describes an efficient, real-world implementation of our attack. Section 5 summarizes the paper and adds a few concluding remarks.

2 Chronology of Research on DFA

Research on Differential Fault Analysis has been produced at an remarkable pace. This section highlights some of the major breakthroughs and provides a fairly detailed description of each known attack.

2.1 September 26, 1996: Boneh, DeMillo and Lipton

Bellcore releases a publicity announcement to the *New York Times* that proposes a fault-based assault against devices that use public-key cryptography. Boneh, DeMillo, and Lipton – the three pioneering scientists – observe that hardware devices may occasionally induce errors that leak information about the underlying cryptography. They point out that this leakage permits the modulus of some implementations of RSA and Rabin signatures to be easily factored.

Their attack on RSA is as follows: Let $n = pq$, where n will be the product of two large primes, p and q. The encryption of a message m is, therefore, $m^e \bmod n$ where e is the private key. Since the exponentiation of the m is computationally expensive, some implementations of RSA use repeated squaring to first compute $E_1 = m^e \bmod p$ and $E_2 = m^e \bmod q$. The final signature E is created from a linear combination of the two sub-signatures using the Chinese Remainder Theorem (CRT).

$E = m^e \bmod n$ is formed as follows: Let a be an integer that satisfies $a \equiv 1 \pmod{p}$ and $a \equiv 0 \pmod{q}$ and let b be an integer that satisfies $b \equiv 0 \pmod{p}$ and $b \equiv 1 \pmod{q}$. By CRT, it is obvious that

$$E = aE_1 + bE_2 \pmod{n} \tag{1}$$

Bellcore's attack takes advantage of CRT's linear structure. Let the correct signature be represented as E in (1). Let $\hat{E}$ be a faulty signature that is computed as

$$\hat{E} = a\hat{E}_1 + b\hat{E}_2 \pmod{n} \tag{2}$$

Without loss of generality, assume that a fault occurs during the computation of $\hat{E}_1$ and not $\hat{E}_2$. Observe then that $\hat{E}_2 = E_2$, and

$$E - \hat{E} = (aE_1 + bE_2) - (a\hat{E}_1 + b\hat{E}_2) = a(E_1 - \hat{E}_1) \tag{3}$$

If $E_1 - \hat{E}_1$ is not divisible by p, then

$$\gcd(E - \hat{E}, n) = \gcd(a(E_1 - \hat{E}_1), n) = q \tag{4}$$

Having q means factoring n is trivial. Once we have the factors the remaining keys in the system can be computed.

2.2 October 18, 1996: Biham and Shamir

Following the assumptions of the Bellcore scientists and the notion of transient faults alluded by Anderson and Kuhn, Biham and Shamir announce the first analysis of a secret key algorithm, Data Encryption Standard (DES), and gave the attack the name Differential Fault Analysis [3]. Their proposed algorithm finds the last DES subkey given less than 200 ciphertexts. For simplicity, their attack focuses on the right half of the data and assumes that one bit of data in one of the 16 rounds is flipped. Furthermore, they require that both the bit position and the round number be chosen with a uniform probability distribution.

Let c be the correctly encrypted ciphertext and c' be a faulted ciphertext. The attack is as follows: After inducing a fault, the round in which the fault occurred is identified. A fault in the right half of the last round produces only an one bit difference on the right half of c' when compared c. In addition, the left half of the ciphertext differs only in the output bits of the S-boxes. Therefore, the six key bits of these S-boxes can be guessed, and certain values can be discounted when they do not satisfy the cryptanalysis. Biham and Shamir claim that approximately four 6-bit values remain for each active S-box.

Faults in previous rounds can be analyzed by their effects on the round that they occur and also the S-boxes in subsequent rounds. For example, a fault in round 15 would produce

a c' with a right half that equals the output difference of the f function in that round. Cryptanalysis is applied when we guess the single bit fault and compare its output difference to the expected difference. We can also verify if 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 (which is the XOR of the left half of the ciphertext with the fault). This analysis causes additional values to be discarded in the last round. The process is repeated for each of the S-boxes in the preceding rounds using counting methods to extract the key.

This attack on the last subkey uncovers 48 of the 56 key bits. The remaining 8 bits can be discovered using a brute force attack or by removing the last round and repeating the same analysis. The first approach is suitable for quickly cracking traditional DES while the latter is more appropriate in assaulting triple DES or DES with independent subkeys.

2.3 October 23, 1996: National University of Singapore

Motivated by the work of Boneh, DeMillo, Lipton, Biham, and Shamir; six scientists at the University of Singapore make a research announcement detailing a specific bit fault attack against RSA in a tamperproof device [4].

Their attack is as follows: Assume that n (the product of two primes p and q) is a 512-bit number. Let e be the encryption key which is known and d be the secret exponent which is stored in the smartcard. The well-known RSA encryption formula (expressed here for reference) is

$$c = m^e \bmod n \tag{5}$$

where c is the ciphertext and m is the plaintext. Furthermore, the following binary representation for the secret exponent is chosen as

$$d = (d_{511})(d_{510})...(d_i)...(d_1)(d_0) \tag{6}$$

where each d_i value is one bit and all the d_i values are concatenated together. In addition, let the ciphertext be denoted as

$$c_0 = c, c_1 = c^2, c_2 = c^{2^2}, ..., c_{511} = c^{2^{511}} \tag{7}$$

The plaintext can then be expressed as

$$m = (c_{511}^{d_{511}})(c_{510}^{d_{510}})...(c_i^{d_i})...(c_1^{d_1})(c_0^{d_0}) \tag{8}$$

Two ways of determining the secret exponent d using DFA are presented. First, suppose that there is a one bit fault in the representation of d in (6). The attacker can then ask the device to decrypt c with the faulted d. Naming the flipped bit d_i', the output of the decryption will be

$$m' = (c_{511}^{d_{511}})(c_{510}^{d_{511}})...(c_i^{d_i'})...(c_1^{d_1})(c_0^{d_0}) \tag{9}$$

With this output the attacker can compute a difference ratio using the original plaintext m. The ratio is

$$\frac{m'}{m} = \frac{c_i^{d_i'}}{c_i^{d_i}} \tag{10}$$

If $\frac{m'}{m} = \frac{1}{c_i}$, then $d_i = 1$ and if $\frac{m'}{m} = c_i$, then $d_i = 0$. The attacker can reconstruct one bit of d by computing c_i and $\frac{1}{c_i}$ for $i = 0, 1, ..., 511$ and comparing it to (10). This process can be repeated with other random faults to find the remaining bits of d.

The second fault model proposes an attack that causes an single bit error in the ciphertext. Let this new ciphertext be c_i'. The decryption of c_i' is

$$m' = (c_{511}^{d_{511}})(c_{510}^{d_{510}})...(c_i'^{d_i})...(c_1^{d_1})(c_0^{d_0}) \tag{11}$$

The following ratio can be computed by the attacker assuming that d_i is 1:

$$\frac{m'}{m} = \frac{c_i'^{d_i}}{c_i^{d_i}} = \frac{c_i'}{c_i} \tag{12}$$

If the attacker computes all 262,144 outcomes of $\frac{c_i'}{c_i}$ and compares it with $\frac{m_i'}{m_i}$, then finding a match means that position i is 1. Again, this process is repeated until d is found.

2.4 October 30, 1996: Biham and Shamir

A few days after the announcement by the group in Singapore, Biham and Shamir release a notice that extends their fault model to include smartcards that use an unknown cryptosystem [5]. Their attack assumes that the device uses asymmetric memories such as EEPROM cells in which faults are more likely to change a 1 bit to a 0 bit. They also assume that the attack can be performed while the device is disconnected from power so that only one bit

is flipped. A third major constraint is that the attacker must be able to load new keys into the device.

The attack is broken into two stages. First, the plaintext m_0 is encrypted a number of times with a fault applied to key k_0 after each round (when the power is disconnected). This generates a sequence of distinct ciphertexts $c_0, c_1, ..., c_f$ with a corresponding set of keys $k_0, k_1, ..., k_f$ (faults that do not cause a different ciphertext/key pair are discarded). Since there are approximately $\frac{n}{2}$ 1 bits in key k_0, the value f is approximately $\frac{n}{2}$. After all the 1 bits in the key have been flipped to 0, further faults will not cause new ciphertexts.

Second, the key can be reconstructed by using the ciphertexts generated in the first stage, and by deterministically feeding the device keys that we believe produced those ciphertexts. We know that the all-zero key k_f will generate the ciphertext c_f and that c_{f-1} is generated by a key that is different from k_f by a single 1 bit. This key, k_{f-1}, can be sought by feeding the n different possibilities into the device and isolating the one that generates c_{f-1}. This process is repeated to find the keys $k_{f-2}, k_{f-3}, ..., , k_1, k_0$. Once k_O is found, the original key is recovered. The search at each stage is O(n) and the overall complexity of the algorithm is at most O(n^2).

2.5 November 20, 1996: Anderson and Kuhn

The attacks proposed by the Bellcore scientists, the National University of Singapore researchers, Biham, and Shamir elicits criticism from the smartcard community. Skeptics suggest that the theoretical assumptions of the fault models have no sound practical basis. Since most smartcards also hold application software in EEPROM, they claim that errors

caused by ionizing radiation would more likely cause a system crash rather than a bit flip in the key. Anderson and Kuhn answer the critics by proposing a new threat model that attacks small fragments of code rather than the memory storing the key [2].

Anderson and Kuhn assert that the simplified version of Bellcore attack on RSA (due to Lenstra) is perfectly suited for a glitch attack. A glitch attack causes faulty behavior in a microprocessor by either applying a clock pulse that is faster than normal or by manipulating transients in the power supply. When this anomaly occurs during the phase of RSA where the primes p and q are combined by the Chinese Remainder Theorem, the factor n may be easily factored. Furthermore, since only one RSA signature is needed, the attack can be performed online quickly.

Similarly, DES can be assaulted by causing instructions to fail. By removing one of the XOR operations - which combines round keys with the inputs to the S-boxes - in the last two rounds, we can obtain output ciphertexts that differ from the original by two or three S-boxes. Cryptanalysis will then allow us to extract five bits of information about the eight key bits that were not XOR'ed. Therefore, eight faults yield a total of 40 bits of the key; the remainder can be found by a brute-force search. Since most smartcard manufacturers keep program routines in ROM and provide toolkits for developers that document the storage of functions such as DES, targeting individual instructions is quite trivial.

Another way to extract keys from a card is through the ROM overwrite attack. A laser cutter applied to the surfaces of ROM chips can cause single bit changes. One such useful change would be to reduce the number of rounds by making a jump instruction unconditional. A similar assault can be mounted on EEPROMs by using microprobing needles to set or

reset target bits.

Anderson and Kuhn also proposed that instruction faults are useful in reverse engineering unknown block ciphers. The following is their description of how to reverse engineer RC5:

> Removing the last operation - the addition of key material - yields an output in which the right hand side is different (it is $(B \oplus A) <<< A$). This suggests, correctly, that the cipher is a balanced Feistel network without a final permutation. Removing the next operation - a shift - makes clear that it was a 32-bit circular shift but without revealing how it was parameterized. Removing the next operation - the XOR - is transparent, and the next - the addition of key material in previous round - yields an output with the values A and B in the above expression. It thus makes the full structure of the data-dependent rotation clear. The attacker can now guess that the algorithm is defined by
>
> $$A = ((A \oplus B) <<< B) \text{ op key}$$
> $$B = ((B \oplus A) <<< A) \text{ op key}$$
>
> Reverse engineering RC5's rather complex key schedule (and deducing that 'op' is actually $+$) would require single-stepping back through it separately; but once we know that 'op' is $+$, we can find the round key bits directly by working back through the rounds of the encryption [2, p.1].

2.6 Status of DFA Research

The following table is a duplicate of the chart located at Bellcore's research archives [6]. It has been updated to reflect the results of this paper and should be an accurate status of the

research into fault-based cryptanalysis as of December 10, 1996. This also adds our attack on RC5.

Protocol	Attack Known?	Number of Faults Needed
Public-Key for Signatures		
RSA	Yes	Many
RSA/CRT	Yes	One
Rabin	Yes	One
Elliptic (Mod p)	No	
Elliptic (Mod n)	Yes	One
DSS	No	
Public-Key for Authentication		
Fiat-Shamir	Yes	Few (~5)
Schnorr	Yes	Many
Guillou-Quisquater	Yes	Many
Secret Key		
DSS	Yes	200
RC5	Yes	32 per partial round

3 RC5 Differential Fault Analysis

In this section, an attack on RC5 is expressed in detail. First, we briefly describe the RC5 block cipher. Second, we define our fault model. Third, an analysis of the attack is presented in two sections: identification and difference analysis. We conclude with a few thoughts about prevention.

3.1 Overview of RC5

RC5 is a block cipher with several variable parameters: block size, key size, and the number of rounds. The RC5 cipher consists of a key expansion stage and an encryption stage. We assume the reader is already somewhat familiar with RC5. There are several publica-

tions which explain RC5 in great detail [8][9]. Our cryptanalysis ignores key expansion and concentrates on the functions in the encryption stage (i.e., XOR, +, <<<, etc.).

The encryption begins by splitting the input bytes into two blocks. RC5 assumes a little-endian convention by placing the first byte in low-order bit positions of the register A, the last byte in the high-order bit positions of the register B, etc. After the key is expanded into an array of keys S, RC5 computes:

$A = A + S_0$
$B = B + S_1$

For i=1 to num_of_rounds do:
$A = (A \oplus B) <<< B + S_{2i}$
$B = (B \oplus A) <<< A + S_{2i+1}$

The final A and B values compose the ciphertext. The decryption follows in a similar manner:

For i=num_of_rounds downto 1 do:
$B = ((B - S_{2i+1}) >>> A) \oplus A$
$A = ((A - S_{2i}) >>> B) \oplus B$

$B = B - S_1$
$A = A - S_0$

3.2 RC5 Fault Model

Our goal is to find the RC5 S-table entries, which are the expanded RC5 key. We choose typical values, 64 bit blocks and 12 encryption rounds, for our statistics, but the attack shown here applies to any variant of RC5. The expanded key consists of 26 32-bit subkeys (each round has two subkeys and there are two initialization subkeys). Before we can proceed with the analysis, we will make a few definitions:

A **half-round** is the operation $X = (Y \oplus X <<< Y) + S_i$. One RC5 round consists of two half-rounds: one with $X = A$, $Y = B$, and the other with $X = B$, $Y = A$.

A **register fault** is the flipping of exactly one bit of a register.

A **useful fault** is a register fault occurring directly before the final RC5 half-round.

In our fault model, we assume that only one register fault per encryption is induced. This fault occurs in either register A or B.

3.3 Analysis

Our attack determines subkey one by one, attacking the last half-round and then using an inductive step to unroll RC5 to the top. We begin by breaking the 26th half-round key. After finding S_{26}, we are able to partially decrypt the ciphertext. We can use this partially decrypted ciphertext to break S_{25}. To generalize, when a procedure finds S_i, this newly acquired subkey can be used to find S_{i-1}. Each subkey is found by identifying and analyzing *useful faults*.

3.3.1 Identification of Useful Faults

We determine the bits of the expanded key by first identifying a fault, and then determining a bit of the key by analyzing the difference between the faulted and normal ciphertext. Because our attack works on half-rounds, we do not need to distinguish between:

$$A = (B \oplus A) <<< B + S_{2i}$$

and

$$B = (A \oplus B) <<< A + S_{2i+1}$$

To simplify the notation in our analysis, we use a different representation for the rounds in RC5. First, two registers X and Y acquire the values of A and B. The generalized operations XOR, bit rotation, and addition are then performed sequentially. We then swap registers X and Y and repeat the computation. The process is as follows:

$(X, Y) = (A, B)$
A fault may occur here
$X = X + S_0$
swap(X,Y)
A fault may occur here
$X = X + S_1$
swap(X,Y)

For i=2 to num_of_rounds*2+1 do
    A fault may occur here
    $X = (Y \oplus X) <<< Y + S_i$
    swap(X,Y)

There are two types of useful faults: one modifies register Y (case mY) directly before the last half round, the other modifies register X (case mX), also directly before the last half round. Let (X, Y) be the expected ciphertext and (X', Y') be the faulty ciphertext.

In the mY case, Y and Y' differ by 1 flipped bit and X and X' differ by 2^x (i.e., $X - X' = 2^x$ or $X' - X = 2^x$, taking into account underflow and overflow). In our analysis, we do not utilize this kind of fault, however an expanded analysis could use this to reduce the number of total faults necessary to break an RC5 key. In the mX case, $Y = Y'$ and X and X' differ by 2^x. We utilize this kind of fault to extract the expanded key.

First we show that $Y = Y'$ and X and X' differing by 2^x identifies a mX fault. Since

Y and Y' are equal, the fault did not happen in the Y register. Furthermore, since Y and Y' are equal, the fault most likely occured before the last half round, otherwise we would expect Y and Y' to be significantly different. If X and X' differ by 2^x, then the error clearly occured in the X register, before the last half round. It is possible the some single bit fault much earlier could result in similar modification of the output, misleading this recognition. The chance of this happening is very slim compared to the incidence of genuine useful faults; roughly as probable as finding a collision in a secure hashing function. Care should be taken during implementation, however, to discard faults that conflict with known bits of the key. If X and X' differ by a non-power of two, then we ignore the faulted ciphertext for this half-round.

3.3.2 Utilization of fault mX

Each new useful fault mX reveals a bit of $(Y \oplus X) <<< Y$ and thus of X, since Y is provided by the output of the round. Consider the final unbroken half round:

$$X' = ((Y \oplus X) <<< Y) + S_i$$

Having identified an mX fault, we find S_i given:

- If 1 bit in X was flipped, 1 bit in $(Y \oplus X) <<< Y$ is flipped.
- Thus, $((X \oplus Y) <<< Y)$ is $\pm 2^n$ what it should be (keeping overflow in consideration when dealing with actual values)
- $((X \oplus Y) <<< Y) + S_i$ is $\pm 2^n$ what it should be.
- We can find bits of $((Y \oplus YX) <<< Y)$ from whether it is high or low, as follows:
  If $((Y \oplus X) <<< Y)$ is 2^n too high, bit x was previously a 0.
  If $((Y \oplus X) <<< Y)$ is 2^n too low, bit x was previously a 1.

- From our computed $((X \oplus Y) <<< Y)$, and our known output X', subtract to get S_i.
- We now know both Y and S_i, so find X (via standard decryption).
- Repeat for all previous half-rounds.

RC5 is a bitwise algorithm *except* for the addition of the S-boxes. Because of this step, there is the possibility of overflow, which will may sometimes make a value that is 2^n high appear to be some non-power of two too low, or vice versa. Since a mX bit error can only induce an error offset of a power of two, if we choose the wrong comparison (higher vs. lower), then the difference will most likely (possibly always) not be a power of two. In such a case we must check if the value has wrapped past zero in one direction or the other.

Here's a quick example with 4 bit words. The correct $X' = 0b1100$ while Y is irrelevant for computing the S values. In our last half-round, we find that the faulted ciphertext reveals $Y = Y'$ and $X' = ((Y \oplus X) <<< Y) + S_i \neq 0b1100$. Bad X's due to useful fault mX yield:

X'	Error	Bit Disclosure
0b1110	2^1 too high	bit 1 is 0
0b0100	2^3 too low	bit 3 is 1
0b1011	2^0 too low	bit 0 is 1
0b0000	2^2 too high	bit 2 is 0

So, $((Y \oplus X) <<< Y) = 0b1001$.

From $S_i = X' - (Y \oplus X) <<< Y)$; $S_i = 0b1100 - 0b1001 = 0b0011 = 3$! Note that this method requires every possible useful fault mX to occur (all 32 in a half-round). For the 12 round, 64-bit block size RC5, we need 832 unique mX faults. Since the faults happen randomly in any of the 1664 possible fault locations, the DFA may take quite a long time to compute. However, we may be able to remove fault randomness by synchronizing the DFA to the smartcard clock. If we count clock cycles, we may be able to more reliably cause a

fault in a predetermined half-round.

3.4 Prevention and Workarounds

Several researchers have advised smartcard makers to compute the encryption twice and then verify that the two encryptions match. This significantly reduces the possibility of generating a faulted ciphertext.

Another method works if we can slow down the smart card encryption process. For instance, the smartcard clock could cycle more slowly or the smartcard could randomly insert several NOP's. If the smartcard is slow enough, then encryption may take 1-2 seconds versus a few milliseconds. This added delay does not hurt normal use. Humans will not notice the difference. But the delay will affect cryptanalysis if several thousand or million faulty ciphertexts are needed. Of course, if an attacker can get around the smartcard security to induce faults, then the attacker might also be able to avoid the delays by modifying the clock or annulling NOP instructions[2].

4 Outline of Implementation

This section outlines the equipment requirements, component interactions, and attack algorithm necessary for implementing the RC5 attack detailed in the previous section. In addition, it analyzes the attack algorithm and shows that it can be achieved in a realistic amount of time and space. The modest resources required for an assault demonstrate the feasibility of our threat model. The technical details of physically assaulting the smartcard is beyond the scope of this paper and this project. However, Anderson and Kuhn in [1] list

many techniques that can be adapted.

4.1 Equipment Requirements

Specific equipment depends on the construction of the smartcard, but the following are some essential devices:

- A sample smartcard (preferably several)
- A PC or workstation with external I/O interfaces
- Smartcard to computer interface (i.e., serial cable, PC card, etc.)
- A fault inducing device such as an ion gun for CMOS chips or an UV lamp for EEPROMs
- A PC controller interface for the fault device

4.2 Component Interaction

The computer serves as the central unit of attack. It sends input to the smartcard via the smartcard to computer interface. Using the same connection, it also retrieves the output. In addition, it controls the firing of the fault device and runs the attack program that computes the subkeys.

4.3 Attack Algorithm

The attack consists of three steps and begins with the initialization of the devices and algorithm data structures. The attack program creates the following data structures and variables (assume that the attack is on 64-bit blocks and 12 rounds of encryption):

$key[26]$	an array to hold all the subkeys
$fault\_ciphertexts[1664]$	an array to hold all the fault ciphertexts
$plaintext$	a variable to hold the plaintext
$ciphertext$	a variable to hold the original ciphertext
i	an counter variable
j	a second counter variable
$temp\_cipher1$	a temporary variable
$temp\_cipher2$	a second temporary variable
key_found	a boolean indicator variable

The attack program asks the user to input 8 bytes of data for encryption. This value is saved in the *plaintext* variable. The program then sends this value to the smartcard to encrypt. The resulting output is saved in the variable *ciphertext*. A loop sets all the elements in $key[i]$ and $fault\_ciphertexts[i]$ to zero.

The second step involves finding all the unique faults and hashing them into $fault\_ciphertext[1664]$. This involves the following loop:

$i \leftarrow 0$
while $i < 1664$
  do $temp\_cipher1 \leftarrow rc5\_fault\_encrypt(plaintext)$
    $j \leftarrow hash(temp\_cipher1)$
    if $fault\_ciphertext[j] = 0$
      then $fault\_ciphertext[j] \leftarrow temp\_cipher1$
        $i \leftarrow i + 1$
      else *continue*

Assume that we have a hash function $hash(x)$ that will take an x value and hash it into one of 1664 unique spots, and a $rc5\_fault\_encrypt(x)$ function that will randomly cause a register fault in one of the rounds and produce a faulty ciphertext.

The third step involves finding the subkeys. This is done by performing bit identification presented in the previous section for every bit of every subkey.

$i \leftarrow 0$

```
while i < 26
    do key_found ← FALSE
       temp_cipher1 ← decrypt(ciphertext, 26 − i)
       j ← 0
       while key_found ≠ TRUE
          do temp_cipher2 ← decrypt(fault_cipher[j], 26 − i)
             key_found ← identify(temp_cipher1, temp_cipher2, key[26])
             j ← j + 1
       i ← i + 1
```

Assume that we have a $decrypt(x, i)$ function that will decrypt a ciphertext x for i half-rounds and generate the output. Also assume that we have a $identify(x, y, key[i])$ function that will take two ciphertexts, x and y, and the subkey array $key[i]$ and do the proper comparisons outlined in the our attack analysis and find keys.

4.4 Algorithm Analysis

The average number of randomly faulted ciphertexts needed is bounded above by $n \ln n$ where n is the number all of different possible mX and mY faults. There is also a more complicated tight bound of how many ciphertexts are needed to find the $n/2$ required mX faults. We show the $n \ln n$ approximation by using a balls and bins analogy as described in [7][p.129].

Let a bin represent a unique fault and a ball represent the generation of a fault. How many balls (ciphertexts) must we randomly toss into the bins (sample space of possible faults) until all of the bins contain at least one ball (all mX and mY faults found)? Let us call a toss of a ball into an empty bin a "hit." We want to know the average number of t tosses needed to get n hits. We partition the t tosses into n stages. In the ith stage, $(i - 1)$

bins contain at least one ball and $(n-i+1)$ bins contain no balls. The probability of getting a hit for all tosses in the ith stage is $\frac{n-i+1}{n}$. Let t_i denote the number of tosses in the ith stage. Thus, the number of required tosses to get n hits is

$$t = \sum_{i=1}^{n} t_i$$

Since each random variable t_i has a geometric distribution with a probability of success $\frac{n-i+1}{n}$, the expected value of t_i is just $\frac{n}{n-i+1}$. And by the linearity of expectation,

$$E[t] = E[\sum_{i=1}^{n} t_i] = \sum_{i=1}^{n} E[t_i] = \sum_{i=1}^{n} \frac{n}{n-i+1} = n \sum_{i=1}^{n} \frac{1}{i} \simeq n(\ln n)$$

This follows from rearranging the summation and using an integral approximation. In the case of 12 rounds and 64-bit input blocks, there are 1664 faults. Thus we expect to need about $1664 \ln 1664 = 10^4$ ciphertexts. Since this is the big bottle neck, if each encryption takes one second, we'll have the key within about 3.4 hours. This is a pessimistic bound since we also require finding every mY fault. Therefore, the algorithm runs in $\Theta(n \log n)$ time where n is the number of encryptions and decryptions.

Since the number of different ciphertexts required is large, the space requirements for this algorithm is roughly $\Theta(n)$ where n is the number of faulty ciphertexts. Each ciphertext takes 8 bytes and we have 1,664 of them. Therefore, the program takes roughly 13 kilobytes of storage.

5 Conclusion

Differential Fault Analysis is a topic with significant ramifications for the smartcard and electronic banking industries. This paper attempts to provide a complete overview of the work done on this topic and adds to the existing work by presenting a original attack on RC5. This paper also moves a step further than most of the existing theoretical work by outlining a practical implementation of the attack.

We see that RC5 is as vulnerable to attack as any other block cipher. Our analysis shows that it can be broken with 32 faulted ciphertexts per round. This suggests that DFA may be a fundamentally universal attack. As more research is done on inducing actual physical faults and breaking other families of ciphers (such as stream ciphers), a generalized attack mechanism may be developed in the near future.

References

[1] R. Anderson, M. Kuhn. *Tamper Resistance—a Cautionary Note.* `http://www.fit.uni-erlangen.de/~mskuhn/tamper.html`, 1996.

[2] R. Anderson, M. Kuhn. *Improved Differential Fault Analysis.* `ftp://ftp.cl.cam.ac.uk/users/rja14/dfa`, 1996.

[3] E. Biham, A. Shamir. *A New Cryptanalytic Attack on DES.* `http://jya.com/dfa.htm`, 1996.

[4] F. Bao, R. Deng, Y. Han, A. Jeng, T. Nagir, D. Narasimhalu. *A New Attack to RSA on Tamperproof Devices.* `http://jya.com/isghak.htm`, 1996.

[5] E. Biham, A. Shamir. *The Next Stage of Differential Fault Analysis.* `http://jya.com/dfa.htm`, 1996.

[6] `http://www.bellcore.com/SMART/secwp.html`, 1996.

[7] T. Cormen, C. Leiserson, R. Rivest. *Introduction to Algorithms.* Cambridge, MA: The MIT Press, 1992.

[8] B. Schneier. *Applied Cryptography*. New York, NY: John Wiley & Sons, Inc., 1996.

[9] W. Baldwin, R. Rivest. *RFC2040 - The RC5, RC5-CBC, RC5-CBC-Pad, and RC5-CTS Algorithms*. `ftp://ds.internic.net/rfc/rfc2040.txt`, 1996.