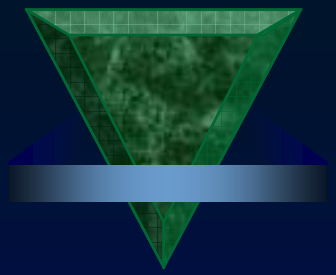
A Generalized Method of Differential Fault Attack Against AES Cryptosystem

Sharif University of Technology, Tehran, Iran

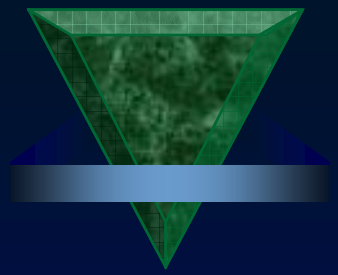
Amir Moradi, Mohammad T. Manzuri,

Mahmood Salmasizadeh



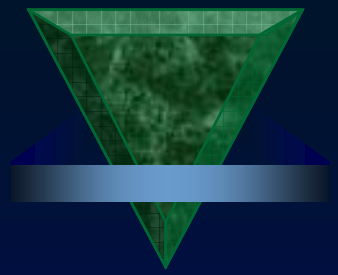
Outline

- History of DFA
- The proposed fault models
- The attack schemes that use our fault models
- Empirical results
- How we can use this approach to break AES without fault injection?
- Future works



History

- The first time, researchers of Bellcore in 1996 introduced a new attack method based on computational errors in implementation of RSA cryptosystem.
- The next year, Biham and Shamir extended their idea and used this method to attack DES and some other symmetric ciphers.
- They tried to inject faults, and they used the difference between faulty Ciphertext and fault free Ciphertext. Thus, they called it Differential Fault Attack (DFA).

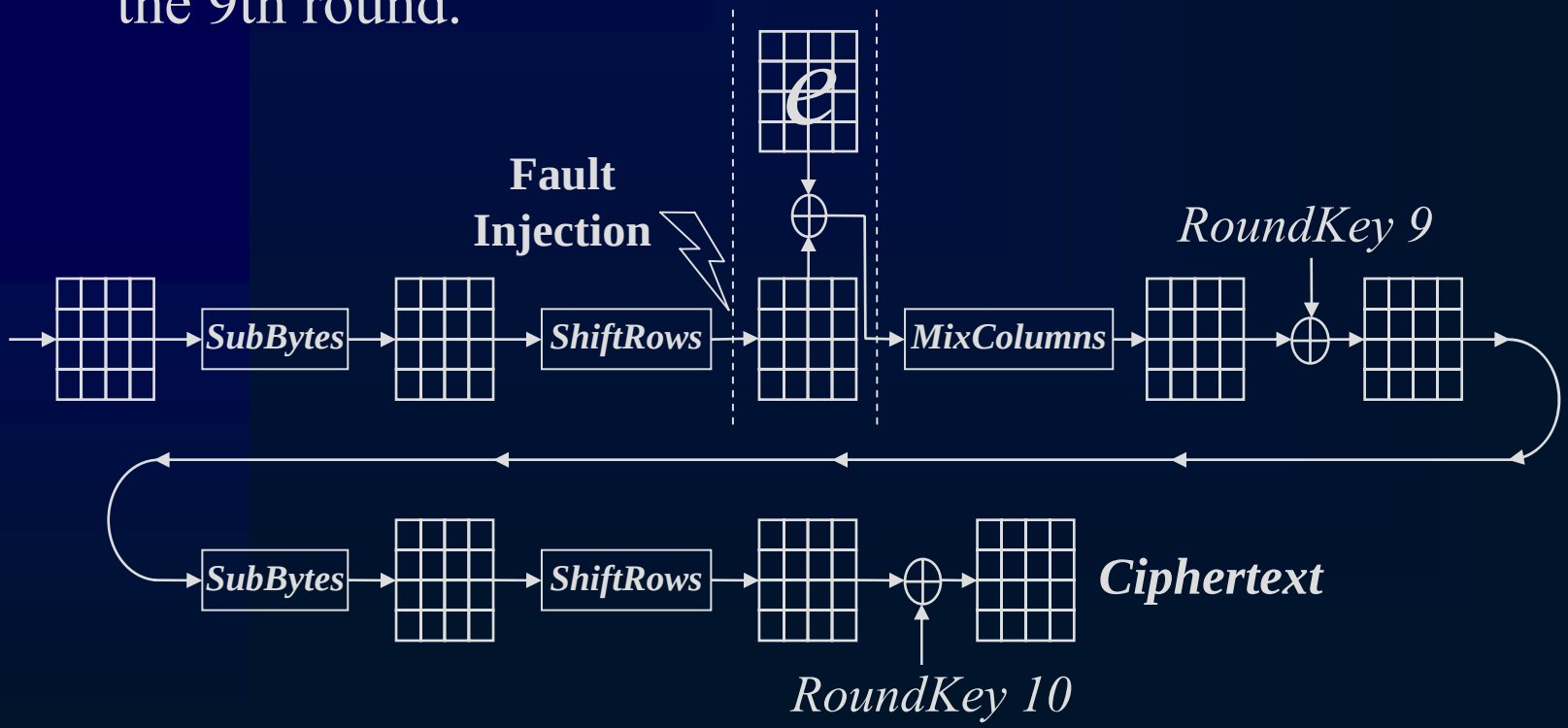


History (cont'd)

- All previous techniques assumed very specific models for fault location and value. Such attacks in real world are applicable only with sophisticated equipments such as narrow Laser beam.
- We present two general models for fault occurrence in AES cryptosystem which neither of them needs any sophisticated equipment.
- The first model covers 1.55% of all possible faults between the beginning of AES-128 and the input of *MixColumns* in round 9, and the reminder (98.45% of them) are covered with the second one.

Fault Models

- We assumed any type of fault appears as a random data to be added to the original data in the input of *MixColumns* of the 9th round.



The First Fault Model

$$M\left(A \oplus \begin{bmatrix} e_1 & 0 & 0 & 0 \\ e_2 & 0 & 0 & 0 \\ e_3 & 0 & 0 & 0 \\ e_4 & 0 & 0 & 0 \end{bmatrix}\right) = M(A) \oplus \begin{bmatrix} e'_1 = 2 \bullet e_1 \oplus 3 \bullet e_2 \oplus e_3 \oplus e_4 & 0 & 0 & 0 \\ e'_2 = e_1 \oplus 2 \bullet e_2 \oplus 3 \bullet e_3 \oplus e_4 & 0 & 0 & 0 \\ e'_3 = e_1 \oplus e_2 \oplus 2 \bullet e_3 \oplus 3 \bullet e_4 & 0 & 0 & 0 \\ e'_4 = 3 \bullet e_1 \oplus e_2 \oplus e_3 \oplus 2 \bullet e_4 & 0 & 0 & 0 \end{bmatrix}$$

- In the first model we suppose that at least one of the bytes e_1 to e_4 is zero.

$$FM_1 = \left\{ \varepsilon : (e_1, e_2, e_3, e_4) \middle| \exists\, e_i = 0; (1 \le i \le 4) \right\}$$

The First Fault Model (cont'd)

- In other words, at least one byte of *MixColumn* (in one column only) is fault free, but we don't know any other thing about occurred faults such as locations and values. In consequence, this model covers one byte, two bytes and three bytes fault(s) among four bytes of each column.

$$CR_1 = \frac{\binom{4}{1} \times 255 + \binom{4}{2} \times 255^2 + \binom{4}{3} \times 255^3}{256^4 - 1} = 0.0155$$

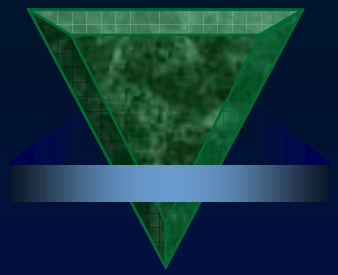
The Second Fault Model

- The second model is the complement of the first one i.e., in the second model, all four bytes of one column should be faulty.

$$FM_2 = \left\{ \varepsilon : (e_1, e_2, e_3, e_4) \middle| \forall\, e_i \neq 0; (1 \leq i \leq 4) \right\}$$

- All four bytes of one column are influenced by the occurred fault.

$$CR_2 = \frac{255^4}{256^4 - 1} = 0.9845$$



Fault Models

- All possible faults can be covered by one of the two presented models and there is no fault that is not included in one of these two models.
- The intersection of the two presented models is empty and the union of them is all possible faults which can occur in four bytes ($256^4 - 1$).
- Any occurred fault in other units of the encryption algorithm from the beginning of the algorithm up to *MixColumns* of round 9 can be considered as another fault occurred in *MixColumns* input of the 9^{th} round, then it's coverable with one of the illustrated models. None of previous fault models against AES had this capability.

Attack Methods

- At the first, we generate two set $S_1$and S_2. These two sets can be generated using function *MixColumns* independent of plaintext and key.

$$S_1 = \{ \varepsilon' : (e'_1, e'_2, e'_3, e'_4) \mid \forall\, e'_i \neq 0; (1 \le i \le 4), \exists\, \varepsilon : (e_1, e_2, e_3, e_4) \in FM_1; MixColumn(\varepsilon) = (\varepsilon') \}$$

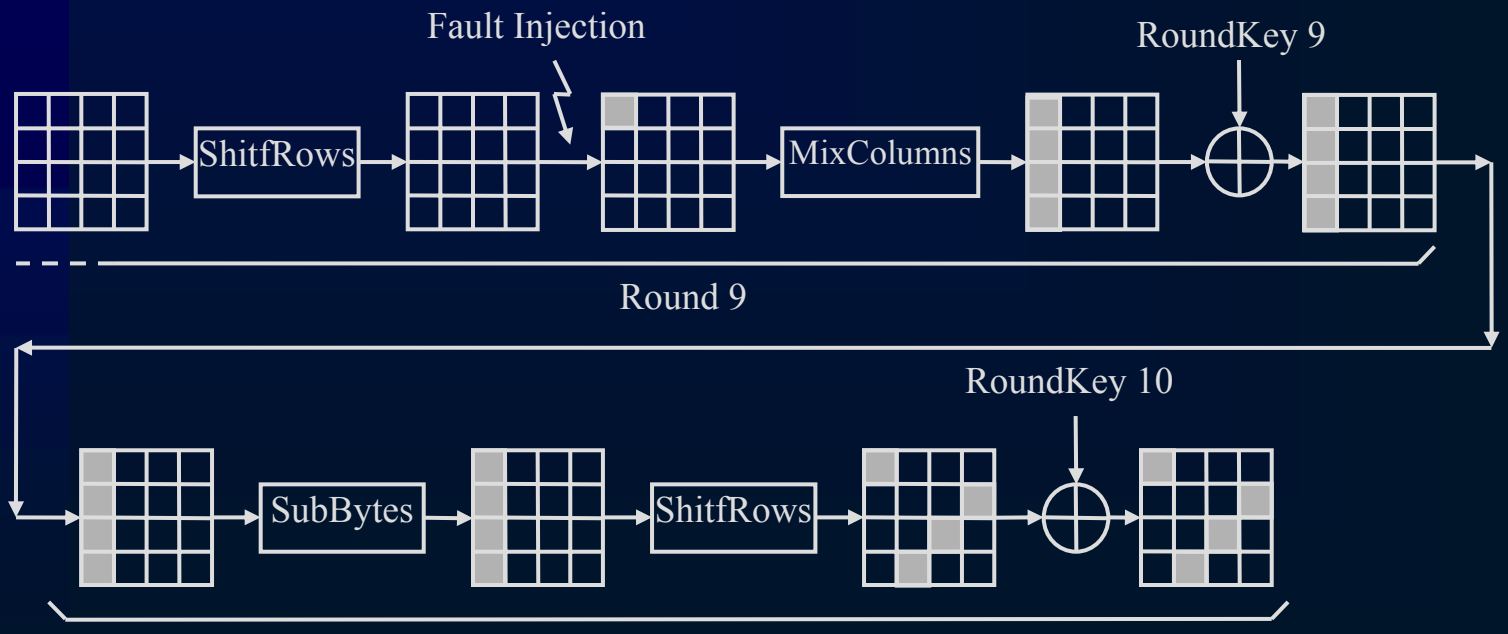
$$S_2 = \{ \varepsilon' : (e'_1, e'_2, e'_3, e'_4) \mid \forall\, e'_i \neq 0; (1 \le i \le 4), \exists\, \varepsilon : (e_1, e_2, e_3, e_4) \in FM_2; MixColumn(\varepsilon) = (\varepsilon') \}$$

$$|S_1| = \binom{4}{1} \times 255 + \binom{4}{2} \times 255^2 + \binom{4}{3} \times 255^3 = 66,716,670$$

$$|S_2| = 255^4 = 4,228,250,625$$

Attack Methods (cont'd)

- After *MixColumns* of round 9 each byte of its output affects on one byte of Ciphertext independent of other bytes, because the *MixColumns* of round 10 is omitted. In fact it causes the success of these attacks. As a result, we could consider each column of *MixColumns* output in round 9 independently.



Attack Methods (cont'd)

- A : output of *MixColumns* in round 9, *AddRK* : *AddRoundKey*

$$Ciphertext = ShiftRows\left(SubBytes\left(A \oplus RoundKey_9\right)\right) \oplus RoundKey_{10}$$

$$AddRK\left(\begin{bmatrix} A_1 \oplus e'_1 \\ A_2 \oplus e'_2 \\ A_3 \oplus e'_3 \\ A_4 \oplus e'_4 \end{bmatrix}, \begin{bmatrix} K9_1 \\ K9_2 \\ K9_3 \\ K9_4 \end{bmatrix}\right) = AddRK\left(\begin{bmatrix} A_1 \\ A_2 \\ A_3 \\ A_4 \end{bmatrix}, \begin{bmatrix} K9_1 \\ K9_2 \\ K9_3 \\ K9_4 \end{bmatrix}\right) \oplus \begin{bmatrix} e'_1 \\ e'_2 \\ e'_3 \\ e'_4 \end{bmatrix}$$

$$SubBytes\left(\begin{bmatrix} B_1 \oplus e'_1 \\ B_2 \oplus e'_2 \\ B_3 \oplus e'_3 \\ B_4 \oplus e'_4 \end{bmatrix}\right) = SubBytes\left(\begin{bmatrix} B_1 \\ B_2 \\ B_3 \\ B_4 \end{bmatrix}\right) \oplus \begin{bmatrix} e''_1 \\ e''_2 \\ e''_3 \\ e''_4 \end{bmatrix}$$

Attack Methods (cont'd)

$$ShiftRows \quad : \quad \begin{bmatrix} D_1 \\ D_{14} \\ D_{11} \\ D_8 \end{bmatrix} \oplus \begin{bmatrix} e''_1 \\ e''_2 \\ e''_3 \\ e''_4 \end{bmatrix} = \begin{bmatrix} C_1 \\ C_2 \\ C_3 \\ C_4 \end{bmatrix} \oplus \begin{bmatrix} e''_1 \\ e''_2 \\ e''_3 \\ e''_4 \end{bmatrix}$$

$$AddRK\left(\begin{bmatrix} D_1 \oplus e''_1 \\ D_{14} \oplus e''_2 \\ D_{11} \oplus e''_3 \\ D_8 \oplus e''_4 \end{bmatrix}, \begin{bmatrix} K10_1 \\ K10_{14} \\ K10_{11} \\ K10_8 \end{bmatrix}\right) = AddRK\left(\begin{bmatrix} D_1 \\ D_{14} \\ D_{11} \\ D_8 \end{bmatrix}, \begin{bmatrix} K10_1 \\ K10_{14} \\ K10_{11} \\ K10_8 \end{bmatrix}\right) \oplus \begin{bmatrix} e''_1 \\ e''_2 \\ e''_3 \\ e''_4 \end{bmatrix}$$

- $(e''_1, e''_2, e''_3, e''_4)$ presented on output of *SubBytes* does not have any linear relation with (e'_1, e'_2, e'_3, e'_4) (errors on its input). But each e''_i relates to only e'_1 and the non linearity of this relation is very high. *ShiftRows* and *AddRoundKey* are linear functions, thus $(e''_1, e''_2, e''_3, e''_4)$ appears exactly on Ciphertext but in (1, 14, 11, 8) locations respectively.

Attack Methods (cont'd)

$$\begin{bmatrix} e''_1 \\ e''_2 \\ e''_3 \\ e''_4 \end{bmatrix} = \begin{bmatrix} FFC_1 \\ FFC_{14} \\ FFC_{11} \\ FFC_8 \end{bmatrix} \oplus \begin{bmatrix} FC_1 \\ FC_{14} \\ FC_{11} \\ FC_8 \end{bmatrix}$$

- We know that ε'' is the difference at the output of *SubBytes*. So, we generate set *EI*.

$$EI = \{(\varepsilon' : (e'_1, e'_2, e'_3, e'_4), \iota : (I_1, I_2, I_3, I_4)) \mid$$

$$SubBytes\left(\begin{bmatrix} I_1 \\ I_2 \\ I_3 \\ I_4 \end{bmatrix}\right) \oplus SubBytes\left(\begin{bmatrix} I_1 \oplus e'_1 \\ I_2 \oplus e'_2 \\ I_3 \oplus e'_3 \\ I_4 \oplus e'_4 \end{bmatrix}\right) = \begin{bmatrix} e''_1 \\ e''_2 \\ e''_3 \\ e''_4 \end{bmatrix} \}$$

Attack Methods (cont'd)

- But all values of ε' are not useful then we generate set *I*.

$$I = EI \cap S_1 = \left\{ \iota : (I_1, I_2, I_3, I_4) \middle| \exists\, \varepsilon' ; \varepsilon' \in S_1 \wedge (\varepsilon', \iota) \in EI \right\}$$

- In other words, set I contains all possible values for the first column of *SubBytes* input at the last round. Thus, we gather some faulty Ciphertexts caused by same plaintext and different faults that are covered by the first model. Then we will decrease the size of set *I* by repeating the proposed method using collected faulty Ciphertexts until set *I* has only one element. Now we know four bytes of *SubBytes* input at the last round and we know the fault free Ciphertext; thus, we can exploit the 10^{th} roundkey.

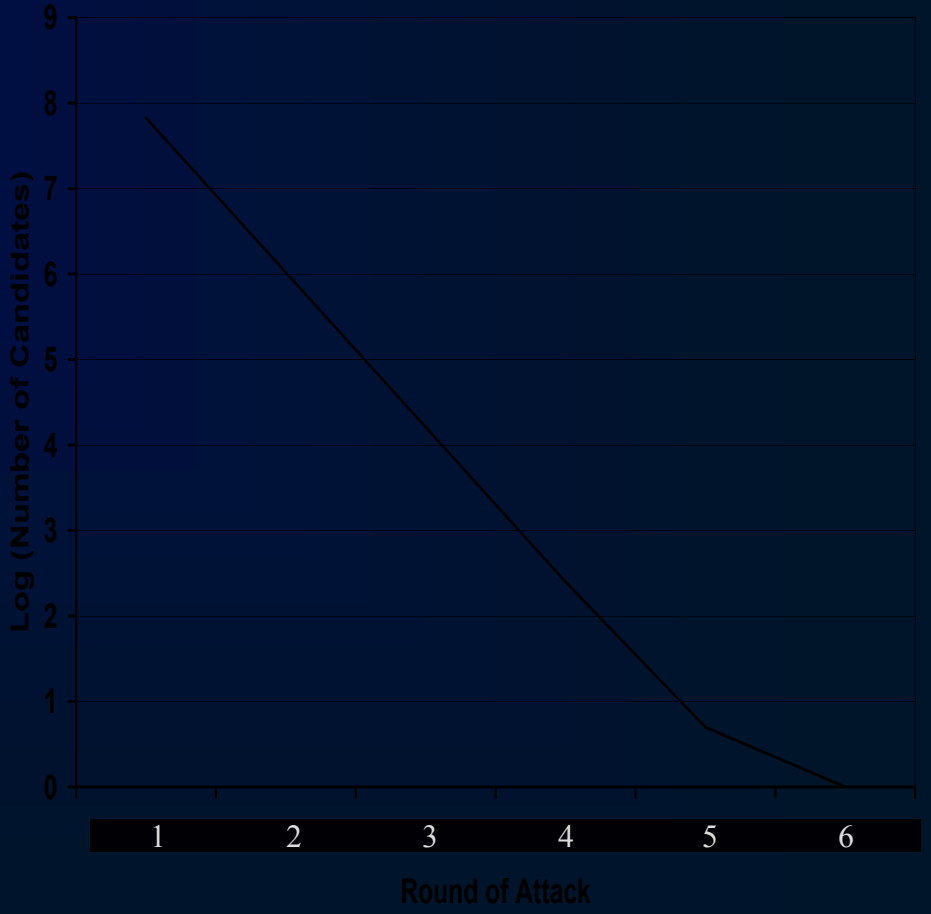
Attack Methods (cont'd)

- One of the advantages of this attack is that finding every four bytes of 10^{th} Roundkey can be processed separately and parallel. Also, we can employ four dedicated systems that each one tries to find four bytes of *K10*.

- The other method to attack is completely similar to the presented one but we assume occurred faults can be covered by the second fault model and we use S_2 for limiting (e'_1, e'_2, e'_3, e'_4) in *EI*. All other specifications and advantages of the first method are true for the second method.

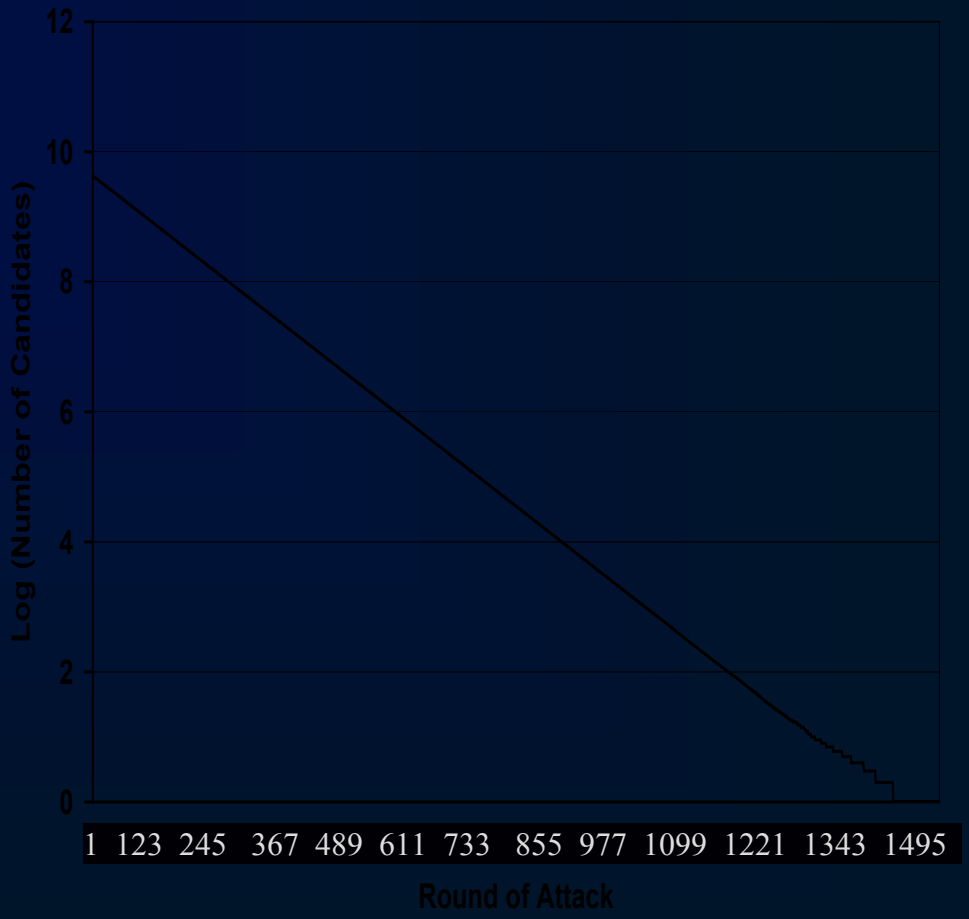
Experimental Results

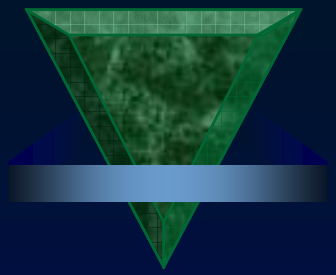
- At the first, we implemented the first method of attack. We started with the first column of *MixColumn* input in round 9 and we selected faulty Ciphertexts that all four bytes in 1, 14, 11 and 8 locations are different with fault free Ciphertext. In this situation, we ran the attack algorithm to 1000 encryption unit with different random generated keys. In average 6 faulty Ciphertexts were needed to find all four bytes of 10th RoundKey and the needed time is not considerable (10 seconds).



Experimental Results (cont'd)

- S_2 has more elements and calculating of intersection between S_2 and *EI* needs more time comparing to the first method. On the other hand, S_2 needs 15.5 GB memory. After improving, optimizing and using memory management techniques on the implementation of the attack, we succeeded to do it with 762.5 MB memory and in almost 2 hours. We should specify that the simulations have been done using Visual C++ on a 2GHz *centrino* with 1GB memory. We applied this attack to AES with 100 random keys. Each attack needed 1495 faulty Ciphertexts and 2 hours in average to find four bytes of *K10*.





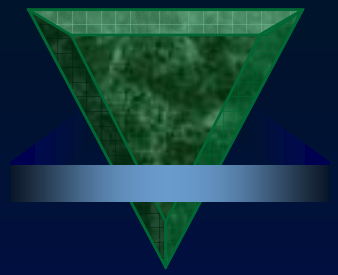
Using Fault Attack Assumption to Break AES

- In proposed methods we supposed faults occur only on internal values, but we assumed RoundKeys and *KeyExpansion* unit is completely fault free. As previously described, any fault that happen before the MixColumns of round 9 is coverable with one of our proposed fault models.
- We can suppose fault occurred on the beginning of the encryption algorithm means plaintext. Thus, changing in plaintext that leads to different Ciphertexts can be assumed as a fault that occurred in the plaintext and is covered by one of our two models.
- Then that's enough to know that the caused difference in *MixColumns* input of round 9 is coverable with which of our fault models.

Future Works

- We are working on designing a method to generate some Ciphertexts that we know which model covers the difference between each of them. Also, we are trying to construct a test method to know the difference between two Ciphertexts at *MixColumns* input in round 9 is coverable with which fault models. Then, by finding any method or designing a rule, we will break AES with 128-bit key and its period will be finished.

- Additionally, we don't need to know plaintexts and if we can find a method to distinguish and classify the different Ciphertexts based on *MixColumns* input of round 9, we will have a successful Ciphertext Only Attack and it's not necessary to run a Known Plaintext Attack.



Questions ?