Fault Analysis of Kuznyechik

Riham AlTawy, Onur Duman, and Amr M. Youssef

Abstract

Kuznyechik is an SPN block cipher that has been chosen recently to be standardized by the Russian federation as a new GOST cipher. In this paper, we present two fault analysis attacks on two different settings of the cipher. The first attack is a differential fault attack which employs the random byte fault model, where the attacker is assumed to be able to fault a random byte in rounds seven and eight. Using this fault model enables the attacker to recover the master key using an average of four faults. The second attack considers the cipher with a secret sbox. By utilizing an ineffective fault analysis in the byte stuck-at-zero fault model, we present a four stage attack to recover both the master key and the secret sbox parameters. Our second attack is motivated by the fact that, similar to GOST 28147-89, Kuznyechik is expected to include the option of using secret sbox based on the user supplied key to increase its security margin. Both the presented attacks have practical complexities and aim to demonstrate the importance of protecting the hardware and software implementations of the new standard even if its sbox is kept secret.

Keywords: Kuznyechik, Differential fault analysis, Ineffective fault analysis, GOST-Grasshopper.

1 Introduction

A draft for a new block cipher called Kuznyechik (Grasshopper in Russian) was presented at CTCrypt 2014 [19]. This new cipher is the result of a project for a new standard for block cipher encryption algorithm [2] published by the Russian Federation. Kuznyechik is intended to accompany the current Russian encryption standard GOST 28147-89 [1] as a new member of the GOST family of ciphers [2]. Although the current standard is considered a lightweight cipher [17], and only theoretical attacks on the full round cipher have been presented [12, 10], it operates on 64-bit blocks of data which is not sufficient for the current requirements [19]. Hence, the need arose for a new standard with a larger block length which is intended to supersede the current GOST 28147-89 cipher in the future. Recently, a meet in the middle attack on a reduced round version of Kuznyechik was presented in [4]. In this paper, we analyze the resistance of the cipher to fault analysis attacks where the standard sbox is used in the first case and a secret sbox is employed in the second one.

Fault analysis is an implementation dependent attack where the attacker applies some kind of physical intervention during the computation of the internal state of the primitive which corrupts random or known bits in the state. Consequently, the attacker observes the correct and faulty outputs and performs her analysis to gain non negligible information about the secret material embedded in the hardware. Fault injection can be done in many ways which include power glitches, clock pulses, and laser radiation [20, 9].

Fault analysis was first introduced when Boneh *et al.* showed how the private key of the RSA-CRT-algorithm can be successfully recovered by observing the correct ciphertext and then injecting a fault and acquiring the faulty ciphertext [6]. Afterwards, the idea was generalized by Biham and Shamir with the introduction of differential fault analysis (DFA) [5]. DFA combines fault analysis with differential cryptanalysis where the difference between faulty and genuine ciphertexts is exploited. DFA attacks have been widely used for the analysis of block ciphers and hash functions (e.g., see [11, 21, 14, 3]). In particular, AES has received a lot of attention with regards to DFA where some of the works used fault injection in the encryption process [21, 11], and others attacked the key schedule [13].

Contrary to DFA, ineffective fault analysis (IFA) [8] is another form of fault analysis which deduces information about the secret material when the induced fault has no effect on the output. In other words, we consider a fault injection successful when both the faulty and original ciphertexts are equal. Accordingly, one knows that the value of the faulted data is similar to the genuine one. The use of IFA is particularly interesting because a common countermeasure to detect fault injections is the use of dual execution branches where the encryption process is executed twice and the output is withheld if a difference in the two ciphertexts is detected. Accordingly, using IFA in our analysis easily bypasses this countermeasure because we gain knowledge of a fault injection only when the two ciphertexts are equal, which is the case that is not detected by the use of dual execution branches. Additionally, the stuck-at-zero fault model assumes that multiple bits are reset to zero by a fault injection. The practical feasibility of bit-reset fault injections has been demonstrated in a set of experiments [18, 15]. In fact, during experimenting with laser fault injection, the rate of occurrence of multiple bit-reset faults was reported to be much higher than that of bit-flip faults [18].

In addition to its use in the analysis of IDEA [7], IFA has been used to reverse engineer AES with secret parameters in [8]. The idea of reverse engineering using fault analysis to recover the adopted secret sbox has also been applied to the current Russian standard GOST 28147-89 [22], where the authors presented three algebraic fault analysis attacks to recover different combinations of its secret parameters.

Fault analysis attacks vary in the number of required faults depending on the employed fault model. Generally, all models assume that the attacker has access to the physical device, and is able to reset it to the same unknown initial settings as often as needed. Furthermore, different assumptions with respect to the amount of control the attacker has over the position and the Hamming weight of the induced faults are employed.

In this work, we present two fault analysis attacks on Kuznyechik. The first attack

is a differential fault analysis attack that adopts the random byte fault model which is considered the most practical fault model. Using this model, the attacker is assumed to be able to fault a random state byte in rounds seven and eight. In this attack, we adapt the attack presented on AES in [16] to analyze Kuznyechik by using an equivalent representation of the last round which enables us to bypass the optimal diffusion effect of the last linear transformation. Our tweak enables a practical and efficient retrieval of the last round key and hence, one can peel off the last round and retrieve the round key used in the second to last round. The knowledge of the last two round keys allows us to invert the key schedule and recover the master key. The second attack is an ineffective fault analysis attack that considers Kuznyechik with a secret sbox. This attack employs a stuck-at-zero fault model where the attacker is assumed to be able to rest the value of a state byte to zero and observe the output of the cipher. Accordingly, one can verify that the value of the genuine state byte is zero when both the faulty and original outputs are equal. Our attack utilizes some of the approaches used to analyze AES with secret parameters [8]. However, unlike AES where the sbox is derived by utilizing the relations between different round keys, we propose a four stage approach where we employ an iterative stage in which we efficiently solve a system of linear equations in $GF(2^8)$ to retrieve multiple sets of candidate parameters. Afterwards, for each candidate set we recover the first two round keys which subsequently enables the retrieval of a set of candidate master keys. Finally, we filter the acquired set of master keys by testing them with a known plaintext-ciphertext pair to recover the right master key and the secret sbox entries. Our second attack demonstrates that trying to increase the security of implementations by having transformations with private parameters is not an adequate measure for protecting the implementation against fault analysis attacks. This fact is particularly interesting for Kuznyechik which, similar to GOST 28147-89, is expected to allow deployment with a secret sbox.

The rest of the paper is organized as follows. In the next section, the description of the Kuznyechik block cipher along with the notation used throughout the paper are provided. Afterwards, in section 3, we provide a detailed description of our differential fault analysis of the cipher. Our four stage ineffective fault analysis attack on Kuznyechik with a secret sbox is given in section 4. Finally, the paper is concluded in section 5.

2 Specification of Kuznyechik

Kuznyechik [19, 2] is an SPN block cipher that operates on a 128-bit state. The cipher employs a 256-bit key which is used to generate ten 128-bit round keys. As depicted in Figure 1, the encryption procedure updates the 16-byte state by iterating the round function for nine rounds. The round function consists of:

- SubBytes (S): A nonlinear byte bijective mapping.
- Linear Transformation (L): An optimal diffusion operation that operates on a 16-byte input and has a branch number = 17. This transformation can also be seen as

a row left multiplication by a 16×16 byte matrix whose coefficients $\alpha_{i,j}$ denote the coefficient at row, i, and column, j, for $i, j = 0, 1, \cdots, 15$.

- Xor layer (X): Mixes round keys with the encryption state.

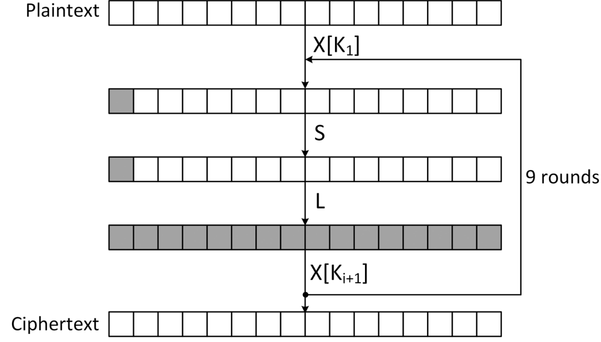

Figure 1: Encryption procedure

Additionally, an initial XOR layer is applied prior to the first round. The full encryption function where the ciphertext C is evaluated from the plaintext P is given by:

$$C = (X[K_{10}] \circ L \circ S) \circ \cdots \circ (X[K_2] \circ L \circ S) \circ X[K_1](P)$$

In our first attack, we use an equivalent representation of the last round function. The representation exploits the fact that both the linear transformation, L, and the Xor operation, X, are linear and thus, their order can be swapped. One has to first Xor the data with an equivalent round key, then apply the linear transformation, L, to the result. We evaluate the equivalent round key after the last round r by $EK_{r+1} = L^{-1}(K_{r+1})$. For further details regarding the employed sbox and linear transformation, the reader is referred to [19].

Key schedule: The ten 128-bit round keys are derived from the 256-bit master key by undergoing 32 rounds of a Feistel structure function. The first two round keys, K_1 and K_2, are derived directly from the master key, K, as follows: $K_1 \parallel K_2 = K$. As depicted in Figure 2, each pair of subsequent round keys is extracted after eight rounds of execution. During each round, the same round function used in the encryption procedure is applied to the right half of the input to the Feistel round. However, round constants are used with the X operation instead of round keys. The 128-bit round constants C_i are defined as follows: $C_i = L(i),\ i = 1, 2, \cdots, 32$. Let $F[C](a, b)$ denote $(LSX[C](a) \oplus b, a)$, where C, a, and b are 128-bit inputs. The rest of the round keys are derived from the first two round keys, K_1 and K_2, as follows:

$$(K_{2i+1}, K_{2i+2}) = F[C_{8(i-1)+8}] \circ \cdots \circ F[C_{8(i-1)+1}](K_{2i-1}, K_{2i}),\ i = 1, 2, 3, 4.$$

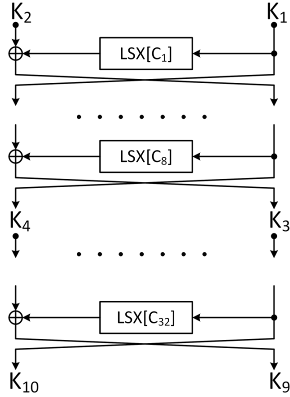

Figure 2: Key schedule

Notation The following notation is used throughout the paper:

- x_i, y_i, z_i: The 16-byte state after the X, S, L operation, respectively, at round i.
- $x_i[j]$: The jth byte of the state x_i, where $j = 0, 1, \cdots, 15$, and the bytes are indexed from left to right.
- Δx_i, $\Delta x_i[j]$: The difference at state x_i, and byte $x_i[j]$, respectively.
- (C, C'): A pair of ciphertexts where C denotes the original ciphertext and C' denotes the faulty one.
- $P[j]$: The jth byte of the plaintext.

3 A Differential Fault Analysis Attack on Kuznyechik

In this attack, we adopt a random byte fault model where the attacker is assumed to be able to fault a random byte in the states before the linear transformation in round eight to a random value. As depicted in Figure 3, a successful fault injection occurs in either x_8 or y_8. The exact position of the fault cannot be determined from the observed ciphertexts due to the optimal diffusion properties of L and it has no added value to the steps of the attack. The attack starts by building a table for all the possible 16×255 differences in z_8 which result from propagating a random error at any of the 16 byte positions in either x_8 or y_8 through the linear transformation layer. Then using the observed ciphertext pair (C, C'), one guesses the last round key, and evaluates the difference at x_9. The evaluated difference is then checked against the differences in the stored table. Successful key candidates result in a match and are consequently stored in another table for further filtration with another (C, C') pair. It is expected that the number of remaining candidate keys after trying N ciphertext pairs (C, C') is given by $256^n(n \times 255^{1-n})^N = 256^{16}(16 \times 255^{-15})^N$

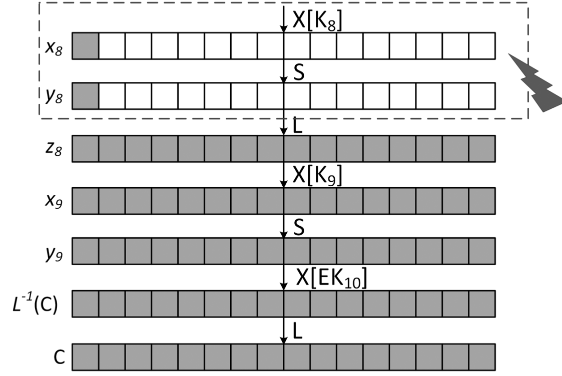

Figure 3: Fault injection in round eight

[16], where n denotes the number of bytes in the state. Accordingly, two ciphertext pairs are required to retrieve the last round key. A naive implantation of this attack requires guessing the 128-bits of the last round key when testing the first ciphertext pair which renders its complexity unpractical. However, if the last round does not contain a linear transformation, one can guess independent key bytes which reduces the complexity as in the case AES and Khazad [16]. Kuznyechik employs a linear transformation in its last round. Accordingly, as depicted in Figure 3, we adopt an equivalent representation by which we evaluate an equivalent key, $EK_{10} = L^{-1}(K_{10})$ and swap the order of the linear transformation and the key mixing operations. Hence, we can apply a two fault practical attack and recover the master key. In what follows, we give the steps of the attack.

1. Store in a table T all the possible 16×255 differences in z_8.

2. Consider two ciphertext pairs (C_1, C'_1), and (C_2, C'_2), for each ciphertext pair, compute $EC_i = L^{-1}(C_i)$, and $EC'_i = L^{-1}(C'_i)$, for $i = 1, 2$.

3. For each value of the possible 2^{16} values of $EK_{10}[0]||EK_{10}[1]$, compute

   $$S^{-1}(X[EK_{10}[0]||EK_{10}[1]](EC_1[0]||EC_1[1])) \oplus S^{-1}(X[EK_{10}[0]||EK_{10}[1]](EC'_1[0]||EC'_1[1])),$$

   $$S^{-1}(X[EK_{10}[0]||EK_{10}[1]](EC_2[0]||EC_2[1])) \oplus S^{-1}(X[EK_{10}[0]||EK_{10}[1]](EC'_2[0]||EC'_2[1])).$$

   Match the resulting two differences from both ciphertext pairs with the two left most bytes of the differences in T. If a match occurs, add $EK_{10}[0]||EK_{10}[1]$ to another table T_k.

4. For each $EK_{10}[0]||EK_{10}[1]$ in table T_k:

   - Remove $EK_{10}[0]||EK_{10}[1]$ from T_k and extend it by one byte $EK_{10}[2]$.
   - For all the 2^8 values of $EK_{10}[2]$, compute

     $$S^{-1}(X[EK_{10}[1]||EK_{10}[2]](EC_1[1]||EC_1[2])) \oplus S^{-1}(X[EK_{10}[1]||EK_{10}[2]](EC'_1[1]||EC'_1[2])),$$

     $$S^{-1}(X[EK_{10}[1]||EK_{10}[2]](EC_2[1]||EC_2[2])) \oplus S^{-1}(X[EK_{10}[1]||EK_{10}[2]](EC'_2[1]||EC'_2[2])).$$

- Match the resulting two differences from both ciphertext pairs with the second and third bytes of the differences in T. If a match occurs, add $EK_{10}[0]||EK_{10}[1]||EK_{10}[2]$ to T_k.

5. Repeat step 4 until the length of the candidate keys in T_k is 16 bytes.

6. Using the two equivalent ciphertext pairs, exhaustively verify which of the remaining keys in T_k produces a difference that matches any of the ones in the precomputed table T.

We have simulated the procedure for the last round key recovery using 100 randomly generated keys. The use of two faults resulted in an avarage of 462.86 remaining candidates in T_k after step 5 of the above procedure. Then the remaining keys were exhastively tested using the same original-faulty ciphertext pairs to recover K_{10}. The attack requires two faults injected in either x_8 or y_8 to recover K_{10}. Afterwards, using the knowledge of K_{10}, one can peel off the last round and repeat the attack by injecting an additional two faults in either x_7 or y_7 to recover K_9. Finally, the knowledge of K_9 and K_{10} allows us to invert the key schedule and compute the master key.

4 Ineffective Fault Analysis Attack on Kuznyechik with a Secret Sbox

In this analysis, we consider the case when Kuznyechik is deployed with a secret sbox. Additionally, we assume that the same sbox is used in the key schedule operation. A similar setting is employed with the current standard GOST 28147-89 and it is expected that users will be allowed to use secret sboxes with Kuznyechik as well. Utilizing secret parameters is assumed to increase the security margin of the employed primitive and makes it harder to cryptanalyze. For that reason, customized primitives with secret parameters are used in military products, gaming systems, and pay TV.

Our attack applies an ineffective fault analysis using stuck-at-zero faults on Kuznyechik. The adopted fault model assumes that the attacker is able to reset a given state byte to zero, hence the attacker can verify if the original byte is zero or not by checking if both the original and faulty ciphertexts are equal. In other words, a successful fault injection takes place when the observed genuine and faulty ciphertexts are similar (i.e., $C = C'$).

Our attack recovers the master key and the sbox secret parameters in four stages. The first stage recovers the value of K_1 relative to the value of $S^{-1}(0)$. Afterwards, in the second stages, we form a system of equations and derive the values of 2^{16} candidates for the right sbox corresponding to all the possible values of $S^{-1}(0)||K_2[0]$. In the sequel, using the 2^{16} candidate sboxes, we recover the values of 2^{16} candidates for K_2 in the third stage . Finally, in the fourth step, we filter the 2^{16} candidate master keys and their corresponding sboxes using a known plaintext-ciphertext pair. In what follows we give the details of our four stage approach.

Recovery of $K_1 \oplus S^{-1}(0)$***:*** This stage recovers the value of the ith byte on $K_1[i]$ up to the constant value $S^1(0)$, for $i = 0, 1, \cdots, 15$, as follows:

1. Iteratively exhaust $P[i]$ and fault byte $y_1[i]$ until an ineffective fault is observed. The occurrence of the IF indicates that the original value of $y_1[i] = 0$.

2. Accordingly, the value of $K_1[i] \oplus S^{-1}(0) = P[i]$.

Applying the above two steps for all the values of i, we recover all of the bytes of K_1 up to the constant $S^{-1}(0)$. This step requires about $2^8 \times 16 = 2^{12}$ fault injections.

Retrieving 2^{16} ***candidates for the sbox:*** In this stage, we iteratively assumes all the possible values $S^{-1}(0)$ and $K_2[0]$ to derive the values of the secret parameters of 2^{16} candidate sboxes. The procedure for each $S^{-1}(0)$ and $K_2[0]$ guess is described as follows:

1. Let $a = S^{-1}(0)$ and hence $S(a) = 0$.

2. Evaluate the candidate value of K_1 by Xoring the value of $K_1[i] \oplus S^{-1}(0)$ recovered in the first stage of the attack, by the guessed value of $S^{-1}(0)$.

3. Repeat the following steps 2^8 times.

   (a) Let $P[0] = m_0 + K_1[0]$, $P[1] = a + K_1[1]$, and $P[i] = K_1[i] + S^{-1}(0)$, for $i = 2, 3, \cdots, 15$.

   (b) Iteratively exhaust m_0 and fault byte $y_2[0]$ until an ineffective fault is observed. The occurrence of the IF indicates that the original value of $y_2[0] = 0$.

   (c) Accordingly, after applying the $X[K_1]$, S, L, and $X[K_2]$ transformations, the value of the first byte of x_2 is given by $\alpha_{0,0}S(m_0) + \alpha_{1,0}S(a) + K_2[0]$.

   (d) Due to the IF which resulted by the current choice of m_0, we know that the value of $x_2[0] = S^{-1}(0)$. Accordingly, one can compute the value of $S(m_0)$ by

   $$S(m_0) = [K_2[0] + S^{-1}(0) + \alpha_{1,0}S(a)]\alpha_{0,0}^{-1}$$

   (e) Set $a = m_0$, and go to step 3 to find another m_0.

The obtained equations from the above procedure correspond to a linear system of equations over $GF(2^8)$ where the unknowns are the sbox entries. According to our experimental results, the system obtained using $P[0]$ and $P[1]$ (or any other pairs) is not a full rank and exhaustively enumerating all its possible solutions is computationally prohibitive. However, because of its structure, we are able to evaluate the values of the sbox entries that are uniquely determined by these equations using the above iterative procedure. When we use three pairs, the system was full rank for 99 out of 100 experiments. Consequently, to recover all the 256 entries of the candidate sbox, the above algorithm is repeated with three different plaintext byte positions. More precisely, after exhausting

$P[0]$ and iteratively setting $P[1]$ to the recovered value of a for 2^8 times, we repeat the exact procedure with $P[0]$ and $P[2]$, and finally with $P[1]$ and $P[2]$. However, in the latter two cases, we start the procedure from step 3, so we use the last recovered value of a and not the first point of entry as the first procedure. All in all, for the 2^{16} candidate sboxes, the attack requires $3 \times 2^{16} \approx 2^{17}$ faults and a time complexity of $2^{16}(3 \times 2^{16}) \approx 2^{33}$.

Recovering the rest of K_2: The previous two stages resulted in 2^{16} candidate sboxes with their corresponding candidate K_1 and $K_2[0]$ values. Accordingly, in this stage, for each sbox out of the retrieved 2^{16} candidates, we recover the remaining fifteen bytes of K_2 as follows: for each byte i, for $i = 1, 2, \cdots, 15$.

1. Let $P[0] = m_0 + K_1[0]$, and $P[i] = K_1[i] + S^{-1}(0)$.
2. Iteratively exhaust m_0 and fault byte $y_2[i]$ until an ineffective fault (IF) is observed. The occurrence of the IF indicates that the original value of $y_2[i] = 0$.
3. Evaluate $K_2[i] = \alpha_{0,i} S(m_0) + S^{-1}(0)$.

This stage requires $2^8 \times 15 \approx 2^{12}$ fault injections and a time complexity of $2^{16} \times 2^8 \times 15 \approx 2^{28}$ to recover the 2^{16} candidate for the remaining fifteen bytes of K_2.

Recovering the master key: In this stage, we test the 2^{16} candidate sets parameters, where each set consists of an sbox and its corresponding master key $K_1||K_2$ against a known plaintext-ciphertext pair. More precisely, using a candidate sbox and its corresponding $K_1||K_2$, one can encrypt a given plaintext and compare the computed ciphertext to the one generated by the attacked device. This stage requires a time complexity of 2^{16}.

Our four stage approach has a practical complexity which was verified by our simulation where the retrieval of the secret parameters of the sbox and corresponding keys require about $2^{12} + 2^{17} + 2^{12} \approx 2^{17}$ ineffective faults and a time complexity of $2^{12} + 2^{33} + 2^{28} + 2^{16} \approx 2^{33}$. This complexity is justified considering that the use of a secret 8-bit sbox and a 256-bit key increases the size of the secret information to about 1940 bit. Indeed, the security level of Kunyechik in this setting is expected to be very high and consequently the practicality of our attack proves its worthiness.

5 Conclusion

In this paper, we have presented fault analysis attacks on the new draft of the Russian encryption standard, Kuznyechik, in two different settings. Our first attack is a differential fault analysis attack that utilizes the random byte fault model. In this attack, we employ an equivalent representation of the last round by which we bypass the effect of the optimal diffusion of the last linear transformation. This tweak enables an efficient and practical

recovery of the master key using four faults. Our second attack is a four stage ineffective fault analysis of Kuznyechik when employing secret sbox. We first recover several sets of candidate secret sbox parameters and their corresponding master key. In the sequel, we filter these sets by testing them against a known plaintext-ciphertext pair to recover the right secret sbox and master key.

Our attack works when we assume that the same secret sbox is used in the key schedule operation. It is interesting to investigate how this approach can be extended in different cases where different sboxes are employed for each byte position, or when the sbox used in the key schedule is different than the one used in the encryption process.

While these attacks may not present direct threat to the theoretical security of Kuznyechik, they serve as a cautionary example to demonstrate the importance of protecting different implementations of the new standard, even if its sbox is not publicly known.

References

[1] GOST 28147-89. Information Processing Systems. Cryptographic Protection. Cryptographic Transformation Algorithm. *(In Russian)*.

[2] The National Standard of the Russian Federation GOST R 34._-20_. Russian Federal Agency on Technical Regulation and Metrology report, 2015. http://www.tc26.ru/en/standard/draft/ENG_GOST_R_bsh.pdf.

[3] AlTawy, R., and Youssef, A. M. Differential fault analysis of Streebog. In *ISPEC* (2015), J. Lopez and Y. Wu, Eds., vol. 9065 of *Lecture Notes in Computer Science*, Springer, pp. 35–49.

[4] AlTawy, R., and Youssef, A. M. Meet in the middle attacks on reduced round Kuznyechik. Cryptology ePrint Archive, Report 2015/096, 2015. http://eprint.iacr.org/.

[5] Biham, E., and Shamir, A. Differential fault analysis of secret key cryptosystems. In *CRYPTO* (1997), J. Kaliski, BurtonS., Ed., vol. 1294 of *Lecture Notes in Computer Science*, Springer, pp. 513–525.

[6] Boneh, D., DeMillo, R., and Lipton, R. On the importance of checking cryptographic protocols for faults. In *EUROCRYPT* (1997), W. Fumy, Ed., vol. 1233 of *Lecture Notes in Computer Science*, Springer, pp. 37–51.

[7] Clavier, C., Gierlichs, B., and Verbauwhede, I. Fault analysis study of IDEA. In *CT-RSA* (2008), T. Malkin, Ed., vol. 4964 of *Lecture Notes in Computer Science*, Springer, pp. 274–287.

[8] Clavier, C., and Wurcker, A. Reverse engineering of a secret AES-like cipher by ineffective fault analysis. In *IEEE workshop on Fault Diagnosis and Tolerance in Cryptography* (2013), pp. 119–128.

[9] Courbon, F., Loubet-Moundi, P., Fournier, J. J., and Tria, A. Adjusting laser injections for fully controlled faults. In *Constructive Side-Channel Analysis and Secure Design* (2014), E. Prouff, Ed., Lecture Notes in Computer Science, Springer, pp. 229–242.

[10] Dinur, I., Dunkelman, O., and Shamir, A. Improved attacks on full GOST. In *FSE* (2012), A. Canteaut, Ed., vol. 7549 of *Lecture Notes in Computer Science*, Springer, pp. 9–28.

[11] Giraud, C. DFA on AES. In *AES* (2005), H. Dobbertin, V. Rijmen, and A. Sowa, Eds., vol. 3373 of *Lecture Notes in Computer Science*, Springer, pp. 27–41.

[12] Isobe, T. A single-key attack on the full GOST block cipher. In *FSE* (2011), A. Joux, Ed., vol. 6733 of *Lecture Notes in Computer Science*, Springer, pp. 290–305.

[13] Kim, C., and Quisquater, J.-J. New differential fault analysis on AES key schedule: Two faults are enough. In *CARDIS* (2008), G. Grimaud and F.-X. Standaert, Eds., vol. 5189 of *Lecture Notes in Computer Science*, Springer, pp. 48–60.

[14] Kircanski, A., and Youssef, A. M. Differential fault analysis of Rabbit. In *SAC* (2009), M. J. Jacobson, V. Rijmen, and R. Safavi-Naini, Eds., vol. 5867 of *Lecture Notes in Computer Science*, Springer, pp. 197–214.

[15] Moro, N., Dehbaoui, A., Heydemann, K., Robisson, B., and Encrenaz, E. Electromagnetic fault injection: Towards a fault model on a 32-bit microcontroller. In *IEEE workshop on Fault Diagnosis and Tolerance in Cryptography* (2013), pp. 77–88.

[16] Piret, G., and Quisquater, J.-J. A differential fault attack technique against SPN structures, with application to the AES and Khazad. In *CHES* (2003), C. Walter, C. Koç, and C. Paar, Eds., vol. 2779 of *Lecture Notes in Computer Science*, Springer, pp. 77–88.

[17] Poschmann, A., Ling, S., and Wang, H. 256 bit standardized crypto for 650 GE GOST revisited. In *CHES* (2010), S. Mangard and F.-X. Standaert, Eds., vol. 6225 of *Lecture Notes in Computer Science*, Springer, pp. 219–233.

[18] Roscian, C., Sarafianos, A., Dutertre, J.-M., and Tria, A. Fault model analysis of laser-induced faults in SRAM memory cells. In *IEEE workshop on Fault Diagnosis and Tolerance in Cryptography* (2013), pp. 89–98.

[19] Shishkin, V., Dygin, D., Lavrikov, I., Marshalko, G., Rudskoy, V., and Trifonov, D. Low-Weight and Hi-End: Draft Russian Encryption Standard. In *CTCrypt* (2014), pp. 183–188.

[20] Skorobogatov, S., and Anderson, R. Optical fault induction attacks. In *CHES* (2003), B. Kaliski, C. Koç, and C. Paar, Eds., vol. 2523 of *Lecture Notes in Computer Science*, Springer, pp. 2–12.

[21] Tunstall, M., Mukhopadhyay, D., and Ali, S. Differential fault analysis of the Advanced Encryption Standard using a single fault. In *Information Security Theory and Practice* (2011), C. Ardagna and J. Zhou, Eds., vol. 6633 of *Lecture Notes in Computer Science*, Springer, pp. 224–233.

[22] Zhao, X., Guo, S., Zhang, F., Wang, T., Shi, Z., and Gu, D. Algebraic fault analysis on GOST for key recovery and reverse engineering. In *IEEE workshop on Fault Diagnosis and Tolerance in Cryptography* (2014), pp. 29–39.