A lightweight-friendly modification of GOST block cipher

Andrey Dmukh Denis Dygin Grigory Marshalko

Abstract

We study the possibility of GOST block cipher modification in such way, that it would resist Isobe and Dinur-Dunkelman-Shamir attacks, and, at the same time, would be still lightweight-friendly.

Keywords: GOST 28147-89, lightweight cryptography, FPGA, ASIC

1 Inroduction

GOST 28147-89 block cipher [1] is known for more than 20 years and with proper s-boxes believed to be secure.

In 2011 two attacks were published by Isobe in [3] and by Dinur, Dunkelman and Shamir in [4], which reduced the time complexity of key recovering down to 2^{225} with data complexity 2^{32} ([3]) or down to 2^{192} with data complexity 2^{64} ([4]). Both attacks exploited the simplicity of GOST key schedule.

Note, that, first, time complexity 2^{192} is considered nowadays greater than enough, and, second, for a block cipher with n-bit block length, encryption of $2^{\frac{n}{2}}$ plain texts leads, due to birthday paradox, to general distinguishing attack. This is interpreted as an a priori insecurity of a block cipher when such amount of data is encrypted [6]. At the same time, the results of [3] and [4] caused considerable reputation damage to GOST. Particularly, this prevented GOST from being standardized by ISO.

At the same time this simplicity make GOST extremely lightweight-friendly (see, for example, [2]). Specifically, the results of this article show,

that GOST with 256 bit key length has the complexity of hardware implementation comparable with the complexities of implementations of algorithms with significantly shorter key lengths (see fig. 1)

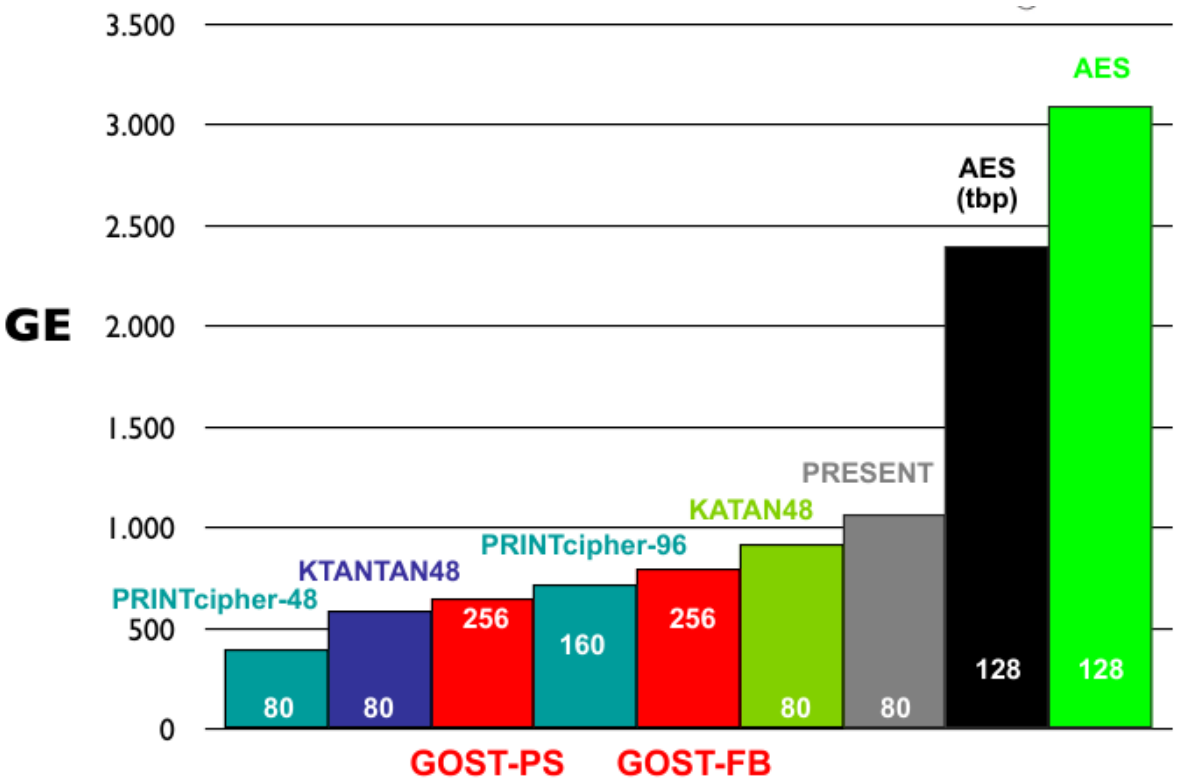

Figure 1: Comparison of lightweight block ciphers hardware implementation from [2]

Further we present a slightly modified version of GOST block cipher, which, we believe, would resist mentioned attacks, and, at the same time, would be still lightweight-friendly.

2 A brief description of GOST block cipher

Here we briefly recall the description of GOST block cipher (see [1]), which is two branch Feistel network. We denote the hole transformation as $F : V_{64} \times V_{256} \to V_{64}$, $F(P, K) = C$, where P - plaintext, K - cipher key, $K = (K_0, \ldots, K_7) \in V_{256}$, $K_i \in V_{32}$, $i \in \overline{0, 7}$, C - cipher text.

For a fixed key K mapping F is a bijection and a composition of 32 mappings: $F(X) = F_1 \cdot F_2 \cdot \dots \cdot F_{32}(P) = F_{32}(\dots F_2(F_1(P)) \dots)$.

Mapping F_i, $i = \overline{1,32}$, ($i-$th iteration) depends on the round key M_i and defined as follows.

Let mapping $F^*(X, M) = F^*((x^{(2)}, x^{(1)}), M) = (y^{(2)}, y^{(1)}) = Y$, $F^* : V_{64} \times V_{32} \to V_{64}$, is set by the following equations: $y^{(1)} = x^{(2)}$, $y^{(2)} = L(\Pi(x^{(2)}[+]M)) \oplus x^{(1)}$, where linear mapping L - is a rotation to the left by 11 bits, and Π - is a non-linear substitution layer, consisting of eight 4-bit s-boxes: $\Pi = (\Pi_0, \dots, \Pi_7)$, $\Pi_i : V_4 \to V_4$, $i = \overline{0,7}$. $[+]$ – addition modulo 2^{32}.

Let mapping $T : V_{64} \to V_{64}$ is defined as $T(X) = T((x^{(2)}, x^{(1)})) = (x^{(1)}, x^{(2)}), \quad x^{(1)}, x^{(2)} \in V_{32}$.

Then for $i = \overline{1,31}$: $F_i(X) = F^*(X, M_i)$ and $F_{32}(X) = T(F^*(X, M_{32}))$.

As a result, we could write mapping F as:

$$F(., K) = F^*(., M_1) \cdot F^*(., M_2) \cdot \dots \cdot F^*(., M_{32}) \cdot T$$

Round keys M_i, $i \in \overline{1,32}$, are equal to $M_i = K_{(i-1) \bmod 8}$ for $i \in \overline{1,24}$ and $M_i = K_{32-i}$ for $i \in \overline{25,32}$.

A schematic description of i-th ($i \in \overline{1,31}$) iteration of GOST is on fig. 2.

3 Isobe and Dinur-Dunkelman-Shamir attacks

Isobe attack exploits the fact that round keys at rounds 1-8, 9-16, 17-24, are applied in the straight order $K_0, \dots, K_7$, and at rounds 25-32 – in the reverse order - $K_7, \dots, K_0$.

GOST enciphering of plaintext P could be represented as

$$C = G \cdot G \cdot G \cdot H \cdot T(P),$$

where $G = F_1 \dots F_8$, $H = F_8 \dots F_1$. It's easy to see that $F_i^{-1} = TF_iT$, $T^2(X) = X$. Then we could proceed as follows

$$HT = F_8 \dots F_1 T = F_8 TTF_7 \dots TF_2 TTF_1 T = F_8 TF_7^{-1} \dots F_1^{-1}.$$

And finally, for $G^3(X) = Y = (y, y)$, we have $HT = G^{-1}$.

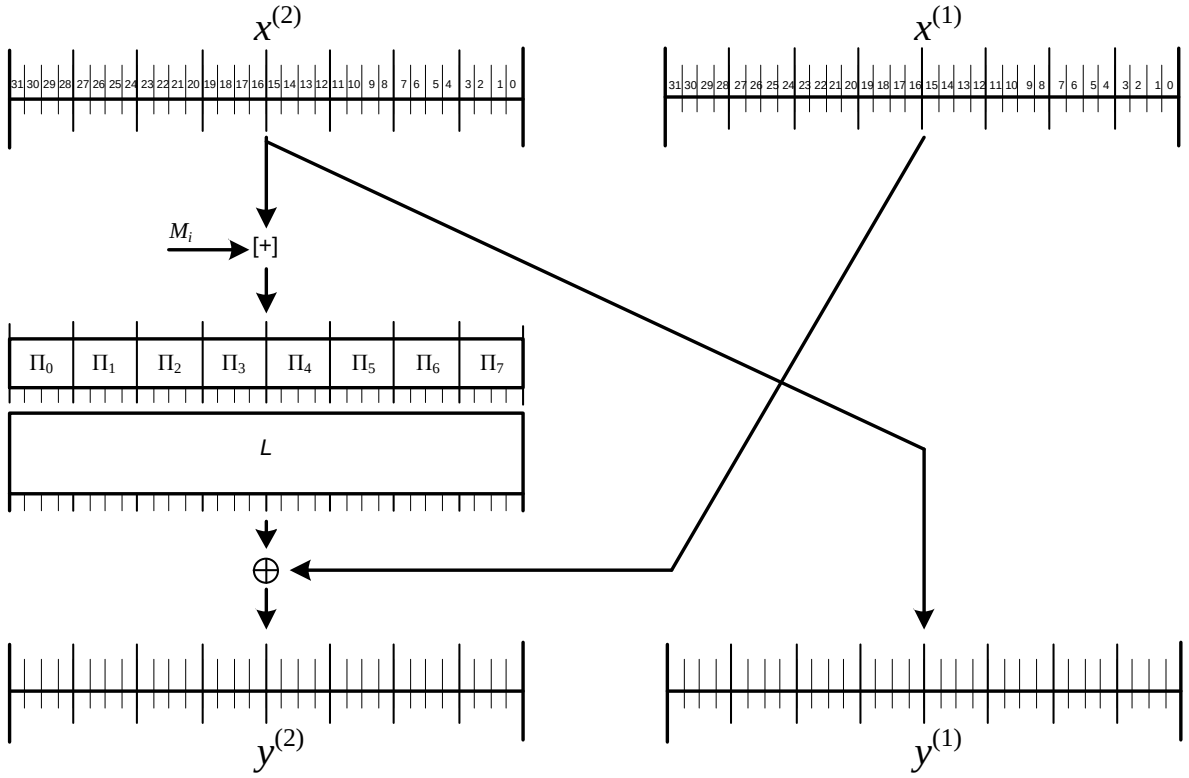

Figure 2: i-th ($i \in \overline{1,31}$) iteration of GOST

If the the described property holds for some ciphertext $C' \in V_{64}$, then, for corresponding plaintext $P' \in V_{64}$ $G^2(P') = C'$, or, in other words, encryption with 32 rounds is equivalent to the encryption with 16 rounds GOST. In [3] this property of GOST is noted as a reflection property. In case of a random choice of plaintext P the probability to get fixed point C' is 2^{-32}. Now, if C' – is a fixed point, then we can apply, so called, reflection-meet-in-the-middle attack to $G^2(P) = C'$ with time complexity 2^{193}. The overall time complexity of Isobe attack is 2^{225} with data complexity 2^{32} pairs of plain/cipher text and success probability $0,63$.

In [4] Dinur, Dunkelman and Shamir proposed two methods for GOST key recovery, which are actually could be considered as further development of Isobe attack. They are based on the meet-in-the-middle attack, which is applied not for a single pair of plain/cipher text on 16 rounds (as Isobe

attack), but for two pairs on 8 iterations.

The first attack exploits the following property. Let P is a fixed point for the first 8 rounds, then it is a fixed point for 24 rounds, and after 32 rounds we get some cipher text C. Then, considering GOST key schedule, we get two pairs of plain/ cipher text for 8 rounds: (P, P) and $(\widetilde{C}, \widetilde{P})$ ($\widetilde{C}$ and $\widetilde{P}$ are obtained from C and P by 32-bit block permutation). For a random key the probability that P is a fixed point for 8 rounds of GOST is 2^{-64}. Therefore, the probability that for the maximum amount of data (2^{64} plain texts) we get a fixed point for 8 rounds is $1 - \left(1 - \frac{1}{2^{64}}\right)^{2^{64}} \approx 1 - e^{-1} \approx 0,63$. The overall time complexity of the attack is $1,5 \cdot 2^{192}$ with data complexity 2^{64} and success probability $0,63$.

The second attack needs 2^{32} pairs of plain/cipher text to get a fixed point for $G \cdot T \cdot G^{-1}$ with probability $0,63$. Let C' is a fixed point for $G \cdot T \cdot G^{-1}$. Then we have correspondence (P, C') between input and output for 16 rounds of GOST. We can run an exhaustive search for 2^{64} outputs Z after 8 rounds of GOST, and, consequentially, we get 2 pairs (P, Z) and (Z, C') for 8 rounds. Finally, we apply a meet-in-the-middle attack for 8 rounds. The overall time complexity of the attack is $1,5 \cdot 2^{224}$ with data complexity 2^{32} and success probability $0,63$. We can also reduce the amount memory from 2^{64} down to 2^{36} by means of guess-and-determine technique (here we can construct a tree of possible keys with partial guessing).

4 2-GOST – a modified GOST 28147-89

The described attacks exploits the simplicity of GOST key schedule, therefore we have to change it in order to withstand them. At the same time it is likely to keep the simplicity of a lightweight hardware implementation. That is way we retain the same principle for key schedule as in GOST: round key $M_i = K_j, j \in \overline{0,7}$ is used for round i, $i \in \overline{1,32}$. In table 1 2-GOST key schedule is presented. In a cell, corresponding a row $i \in \overline{0,3}$ and a column $j \in \overline{0,7}$ we have a round key $M_{8 \cdot i + j + 1}$. A set of s-boxes is considered as a long term key, so it's not specified by the standard. Nevertheless, we usually consider it known during cryptanalysis. More than

i/j	0	1	2	3	4	5	6	7
0	K_0	K_1	K_2	K_3	K_4	K_5	K_6	K_7
1	K_3	K_4	K_5	K_6	K_7	K_0	K_1	K_2
2	K_5	K_6	K_7	K_0	K_1	K_2	K_3	K_4
3	K_6	K_5	K_4	K_3	K_2	K_1	K_0	K_7

Table 1: 2-GOST key schedule

that, although it is usually consists of 8 different s-boxes, the standard doesn't require this.

It's better to have as few different s-boxes as possible for better implementation (in the best case – one). But as we would show further we can use even two s-boxes and still consider 2-GOST as a lightweight cipher. So we decided to use two different s-boxes for variety in the following manner: $\Pi_0 = \Pi_1 = \Pi_2 = \Pi_3 = \pi_1$, $\Pi_4 = \Pi_5 = \Pi_6 = \Pi_7 = \pi_2$, where π_1 = (6,A,F,4,3,8,5,0,D,E,7,1,2,B,C,9) , and π_2 = (E,0,8,1,7,A,5,6,D,2,4,9,3,F,C,B)

These substitutions have been previously proposed during ISO standardization process, and have minimal linear and differential characteristics, and in terms of [7] could be considered as "optimal".

5 Lightweight hardware implementation

As a measure for 2-GOST lightweight hardware implementation assessment we use post-synthesis figures from [2] for serialized and round-based implementations of GOST. We restrict ourselves to serialized implementation, since it has minimal number of gate equivalents (GE).

Poschmann et al. in [2] use Synopsys DesignCompiler CAD for GOST implementation synthesis. The use of such CAD implies rather deep understanding of ASIC engineering, so, we used a little bit more common Xilinx ISE 9.2 CAD for FPGA synthesis. Our synthesis figures could be considered as upper bounds for ASIC implementation. We performed simulation for Virtex5 5vlx330ff1760-2, which has one of the maximal CLBs in the Virtex5 family and maximal frequency 500 Mhz. This reduces loss

in performance in case of maximum workload of FPGA.

Figure 3 shows a lightweight hardware architecture of GOST 28147-89 with one s-box from [2], blue rectangle is a key schedule.

Intermediate encryption vectors are stored in two 32-bit registers **L** and **R**. On initial signal **rst** 4 bits of plain texts **OT** are loaded into least significant bits of **R**; 4 most significant bits of **R** are loaded into least significant bits of **L**, and other bits of **L** and **R** are shifted to the most significant bits by 4; as a result after 16 cycles both registers would store 64 bits of plaintext. On "end of ciphering" **rdy** signal cipher text vectors **CT** are generated in the same way with backward shift. When ciphering, 4 least significant bits of **R** are added to the corresponding key bits (and with carry bit from previous 4-bit block) and 4 least significant bits of **L**, the result is loaded into 4 least significant bits of **L**. Finally, both **L** are **R** are rotated to the most significant bits by 4. After 8 cycles the values in **L** and **R** are interchanged, with 11-bit rotation, as in [2]. For key schedule we use a 2-bit counter **CurrSerie** for selecting proper 4-bit key block (e.g. row number from table 1), and two 3-bit counters **CurrRound** (e.g. column number from table 1) and **CurrTick** - the number of 4-bit subblock of a 32-bit key, selected by **CurrSerie** and **CurrRound**.

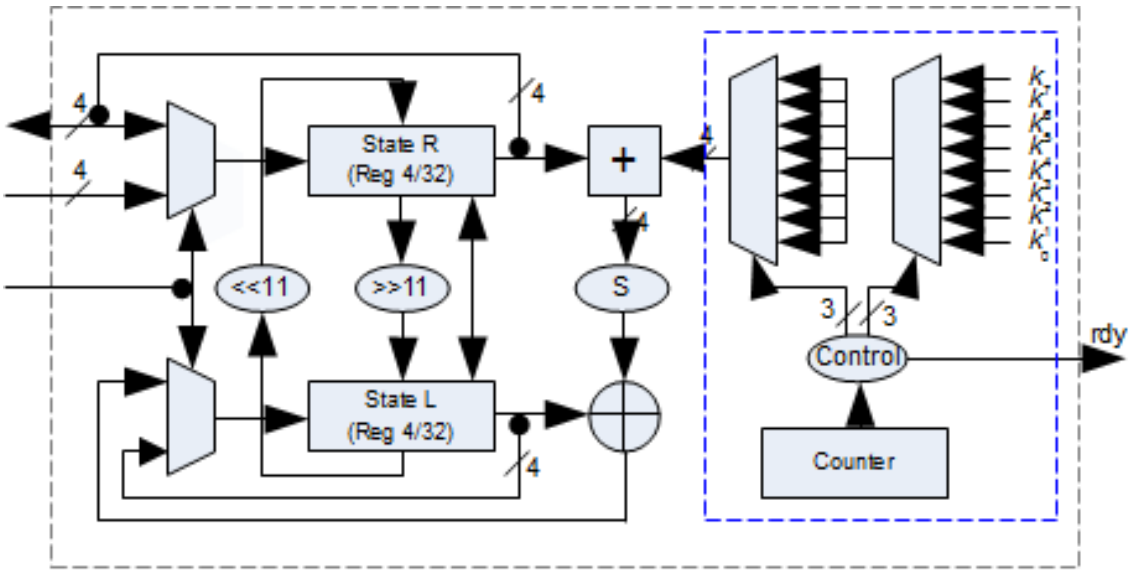

Figure 3: Architecture of serialized implementation of GOST 28147-89 with one s-box [2]

In case of GOST 28247-89 key schedule the counters work as follows: **CurrTick** is incremented every cycle. If **CurrTick** is equal to 7, then, if **CurrSerie** is equal to 3, **CurrRound** is decremented, otherwise, **CurrRound** is incremented. Besides, if **CurrRound** is equal to 0, **CurrSerie** is incremented. If **CurrSerie** is equal to 0, **rdy** signal is sent.

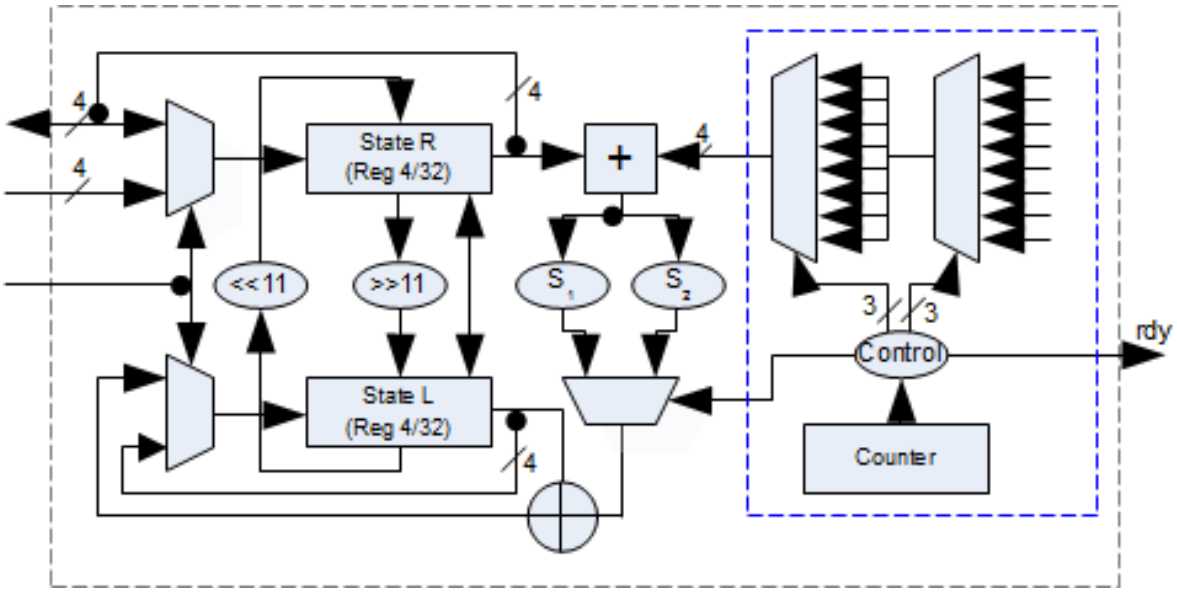

Figure 4: Architecture of serialized implementation of 2-GOST

If the key is an external signal for the ciphering modulus, the synthesis result for this architecture shows 722 GE. In [2] another variant is discussed, when a fixed key is directly stored in the modulus. In this case CAD synthesizes a ROM block consisting of LUT-cells (8 32-bit words), and key schedule implementation falls down to 180 GE. The overall estimate for an external key is 2165 GE, and 1584 GE for a fixed key.

In order to implement the key schedule from table 1 we need one more signal **CurrKeyNr**, which is the sum of $\pm$**CurrTick** and a **CurrSerie**-dependent constant. In this case CAD synthesizes ROM block consisting of LUT-cells (four 3-bits blocks) and a 3-bit adder. The summation sign is **CurrSerie**-dependent. Note, since the constant has a 2-bit address bus, then the first row in Table 1 could also be shifted by an arbitrary constant. The estimate of a key schedule implementation is 775 GE for an external

	CAD Xilinx ISE 9.2	CAD Xilinx ISE 9.2	CAD Synopsys DesignCompiler
	external key	fixed key	fixed key
GOST 28147-89	2137 GE	1556 GE	650GE [2]
2-GOST	2158 GE	1570 GE	750 GE

Table 2: Summary of results

key and 208 GE – for a fixed key. The overall estimate of an external key implementation is 2204 GE and 1644 GE – for a fixed key implementation.

In contrast to ASIC, where GE estimate depends on the properties of a specific set of s-boxes, FPGA CAD implements s-box as a ROM block consisting of LUT-cells (16 4-bit words), that's why the final estimate doesn't depend on the s-box choice. If we use two different s-boxes subject to the most significant bit of, CAD synthesizes two ROM blocks and demultiplexer (MUX). The overall estimate for such modulus is 2176 GE for an external key and 1672 GE for a fixed key. Note, that for an external key the estimate turn out even less than for an algorithm with a single s-box: CAD differently synthesizes algorithms, and differently assess the use of logical elements, even for the same FPGA blocks.

Now we can deduce that the proposed modification slightly increases GE estimation for FPGA: for less than 2% for an external key and for less than 6% for a fixed key. The maximum frequency for both implementations exceeds 300 MHz, due to additional control of dependent levels of logic between triggers with intermediate computations, that is more than 50% of maximum FPGA frequency. If we do not need such frequency, GE estimate could be lowered.

From our figures and figures presented in [2], we can estimate the ASIC implementation of 2-GOST. We need 30 GE for additional 4-bit s-box, 23 GE for 3-bit counter and 15 GE for MUX. Finally, we have that for 2-GOST ASIC implementation we need no more than 750 GE.

6 Security evaluation of 2-GOST

Both Isobe and Dinur-Dunkelman-Shamir attacks exploit the reflection property for the last 16 iterations. For the proposed algorithm the probability of the corresponding event is negligible: $P\{K_0 = K_2 = K_4 = K_6, K_1 = K_3 = K_5 = K_7\} = 2^{-192}$ (if keys are selected at random).

The first Dinur-Dunkleman-Shamir method works if $K_0 = K_2 = K_4 = K_6 = K_1 = K_3 = K_5 = K_7$. The probability of such event is 2^{-224}.

Since the new key schedule could be represented as a concatenation of different shifts of $(K_0, \ldots, K_7)$, 2-GOST (together with original GOST) is subjected to related-key attacks. At the same time, such attacks are difficult for practical implementation, since the probabilities of relations are negligible (see, for example, [5]), when keys are selected randomly.

2-GOST can effectively resist linear and differential cryptanalysis, since we use "optimal" s-boxes.

References

[1] National Bureau of Standards. Federal information processing standard – Cryptographic protection – Cryptographic algorithm. GOST 28147-89, 1989.

[2] A. Poschmann, S. Ling, H. Wang. 256 Bit Standardized Crypto for 650 GE - GOST Revisited. CHES 2010, LNCS 6225, pp. 219-233, 2010.

[3] T. Isobe. A Single-Key Attack on the Full GOST Block Cipher. LNCS v. 6733, p. 290-305. Springer, 2011.

[4] I. Dinur, O. Dunkelman, A. Shamir. Improved Attacks on Full GOST. Cryptology ePrint Archive, Report 2011/558, 2011. http://eprint.iacr.org/.

[5] V.I. Rudskoy. On several approaches for the estimation of the effectiveness of related-key attacks. (In Russian) RusCrypto'2011. http://www.ruscrypto.ru/.

[6] ISO/IEC JTC 1/SC 27 Standing Document 12 (SD12) on the Assessment of cryptographic algorithms and key lengths.

[7] G. Leander, A. Poschmann, On the classification of 4-bit S-boxes, WAIFI, Springer-Verlag, p. 159-176, 2007.