WAS Science Nature
WASSN

ISSN: 2766 - 7715
WAS PRESS

Article

Enhanced Version of GOST Cryptosystem for Lightweight Applications

Bassam. W. Aboshosha[1,*], Mohamed M. Dessouky[2], Rabie. A. Ramadan[3] , Ayman El-Sayed[2] and Fatma H. Galal[2]

[1]*Department of Computer Engineering, Higher Institute of Engineering, Elshorouk Academy, Cairo , Egypt.*
[2]*Department of Computer Engineering, Menofia University, Menoufia, Egypt.*
[3]*Computer Engineering Dept. Cairo University, Cairo, Egypt and University of Hail, Hail , KSA.*
***Correspondence:** bassam.ahmed32@gmail.com
Received: 05.05.2018; **Accepted:** 07.06.2018; **Published:** 15.06.2018

Abstract: Recently, constraint devices applications are widely extended in different fields. These devices are connected to serve billions of users. A well-known example of these structures is the Internet of Things (IoT). Due to the spreading of this technology, new security risks threatening the secrecy and privacy are generated. Therefore, confidentiality and protection of data communication must be concerned. A lightweight cryptographic algorithm is one of the most suitable solutions to save information in such environments. In this paper, we propose a rigid, lightweight, and energy-efficient security approach called E-GOST for the IoT systems. Furthermore, a new substitution box is suggested to be strong and immune box against the various types of attacks especially the side channel attack. E-GOST relies on the traditional GOST algorithm.

Keywords: Constraint devices; lightweight cryptographic; IoT; side channel attack; S-box; bit slice implementation.

1. Introduction

The revolution of Information and Communication Technology (ICT) is mainly based on constrained resources devices. These devices suffer from limited resources such as CPU capabilities, bandwidth, memory (ROM and RAM), and battery lifespan. There are many application areas in which such devices are cooperating to carry out specific tasks. One of the major application areas is the Internet of Things (IoT) [1]. IoT allows people to connect with anyone or anything at any time from anywhere. Under the concept of IoT, not only conventional types terminals including PCs, smartphones, and tablet PCs, but also automobiles, electrical home appliances, robots, and even facilities themselves will be connected to the Internet. These terminals are spreading throughout our living spaces and contributing to building the ubiquitous network described above. Under these circumstances, there is a definite need for security functions to protect personal information and privacy and guarantee the integrity of information exchanged on the network [2].

However, since providing such functions is not the main purpose of IoT services, the functions should not hinder the operations for the users. Moreover, it is unlikely that all IoT devices utilize high-performance CPUs. It should be assumed that some devices have poor computing resources with throughput and memory capacity inferior to conventional ICT terminals and battery-power restrictions on operating time. When the CPU and memory must be shared by many applications, cryptography with less CPU cost and memory consumption is sometimes demanded [3]. One example is the cryptography used in smartphones, tablet PCs, and smart TVs (high-performance television with an Internet connection). Furthermore, some devices that operate on batteries expect a cryptographic algorithm that consumes less power. For example, environment-measuring devices are often installed at locations where no utility power is available. Medical implant devices rely only on battery power and are required to be as small as possible to lessen their effects on the human body. Lightweight cryptography is expected to serve these applications [4].

Lightweight Cryptography has become one of the hot research topics. It is a relatively new science that is mainly sub-field from cryptography. It concerns new designs, adaptions or efficient implementations of cryptographic primitives and protocols. Due to the wide spreading of limited resources devices and its applications besides very strong attacker cryptanalysis techniques and that the nodes deployment in the harsh environment lead to new attacks —especially the possibility of physical attacks—there is a necessary need for lightweight security solutions that are tailored to the everywhere computing paradigm [5]. Lightweight cryptography is the part of cryptography using the most basic and simple computational operations to provide security systems. Yet, combining these operations to achieve a rigid and robust cryptosystem is a challenge that remains to be solved. Consequently, the research and development of lightweight cryptography for the implementation on devices with limited resources has been increased, and many cryptosystems have been proposed in research manuscripts [6]. In this paper, we follow the approach of adapting the design and the implementation of both hardware and software of the existing standardized block ciphers. A new version of GOST algorithm is proposed which called "E-GOST".

GOST block cipher is a Soviet and Russian government standard. It is the basis of most secure information systems in Russia. It has a simple structure suitable for compact hardware implementations. It has been classified as one of the ultra-lightweight block ciphers. Therefore, it is a target for the constrained environments. The immunity of any block cipher against several attacks depends basically on the rigidity of Substitution boxes (S-boxes) because it is the main non-linear component of a block cipher. The design and characteristics of S-boxes of a block cipher are central measures of resistance against all crypt-analytical techniques. Therefore, the analysis of an S-box is very important. In this paper, a discussion and analysis of the S-boxes of the Central Bank of the Russian Federation GOST version are introduced. Furthermore, a new substitution box is suggested as more strong, rigid, and immune boxes against the various attacks.

The paper is organized as follows: In the next section, we are going to introduce the GOST encryption algorithm briefly. Since the S-boxes was not specified in the original algorithm, this paper discusses the selection of an appropriate approach for S-boxes selection in subsequent Section 3. We also compare the linear and differential properties of the S-boxes as used by the Central Bank of the Russian Federation and the proposed S-box. The bit-slice implementation of the proposed S-box is given in section 4. The hardware implementation result of the proposed approach is presented in Section 5. Finally, this paper is concluded in Section 6.

2. GOST Encryption Structure

GOST algorithm is a symmetric block cipher, which conforms to Feistel scheme. 64-bit blocks of data are submitted to the input and converted into 64-bit blocks of encrypted data by 256-bit key [7]. Function F is applied in each round on the right side of plaintext messages ; it converts the plaintext with three cryptographic operations:

Adding data and subkey modulo 232.

Substitution of data using secret S-boxes.

Left cyclic shift by 11 positions.

Output of F-function is added modulo 2 to the left part of the plaintext, then right and left sides are swapped for next round. The overall process of the round function can be summarized in the formal notation as the following:

$$\boldsymbol{L}_{\boldsymbol{i+1}} = \boldsymbol{R}_{\boldsymbol{i}} \quad (1)$$

$$\boldsymbol{R}_{\boldsymbol{i+1}} = \boldsymbol{L}_{\boldsymbol{i}} \oplus (\boldsymbol{S}(\boldsymbol{K}_{\boldsymbol{i}} + \boldsymbol{R}_{\boldsymbol{i}} \ \boldsymbol{mod} \ \boldsymbol{2^{32}}) \ll \boldsymbol{11}) \quad (2)$$

where ⊕ denotes a bitwise exclusive OR and << a a rotation to the left by a bits.

The algorithm has 32 rounds. In the last round of encryption right and left parts are not swapped. The overall dataflow diagram of GOST is shown in Figure 1. GOST uses 8 S-boxes, which convert 4-bit input to 4-bit output. GOST has no predefined S-boxes unlike most encryption algorithms and any values can be used for them. The Secret Key is a series of eight 32-bit phrases: K1, K2, K3, K4, K5, K6, K7 and K8. One of these 32-bit phrases is used as a round subkey in every encryption round. When round subkey is calculated, the following principle is used:

Table 1. Key schedule in GOST

Rounds	**1 … 8**	**9 … 16**
Keys	**k1 k2 k3 k4 k5 k6 k7 k8**	**k1 k2 k3 k4 k5 k6 k7 k8**
Rounds	**17 … 24**	**25 … 32**
Keys	**k1 k2 k3 k4 k5 k6 k7 k8**	**k8 k7 k6 k5 k4 k3 k2 k1**

from round 1 to round 24 the order is straight; from round 25 to round 32 reversed order is used [7].

The S-boxes accept a four-bit input and produce a four-bit output. The S-box substitution in the round function consists of eight 4 × 4 S-boxes. The S-boxes are implementation-dependent – the same S-boxes must be used by parties wishing to secure their communications using GOST. The S-boxes can be kept confidential for additional safety. In the original standard where GOST was specified, no S-boxes were given, but the S-boxes used at the Central Bank of Russian Federation is published as a powerful example as shown in table 2.

Table 2. S-boxes of the Central Bank of Russian Federation

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
S1 (x)	4	A	9	2	D	8	0	E	6	B	1	C	7	F	5	3
S2 (x)	E	B	4	C	6	D	F	A	2	3	8	1	0	7	5	9
S3 (x)	5	8	1	D	A	3	4	2	E	F	C	7	6	0	9	B

S4 (x)	**7**	**D**	**A**	**1**	**0**	**8**	**9**	**F**	**E**	**4**	**6**	**C**	**B**	**2**	**5**	**3**
S5 (x)	**6**	**C**	**7**	**1**	**5**	**F**	**D**	**8**	**4**	**A**	**9**	**E**	**0**	**3**	**B**	**2**
S6 (x)	**4**	**B**	**A**	**0**	**7**	**2**	**1**	**D**	**3**	**6**	**8**	**5**	**9**	**C**	**F**	**E**
S7 (x)	**D**	**B**	**4**	**1**	**3**	**F**	**5**	**9**	**0**	**A**	**E**	**7**	**6**	**8**	**2**	**C**
S8 (x)	**1**	**F**	**D**	**0**	**5**	**7**	**A**	**4**	**9**	**2**	**3**	**E**	**6**	**B**	**8**	**C**

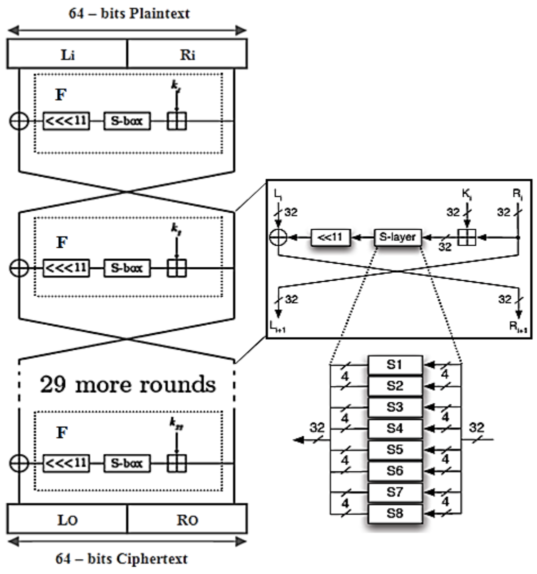


Figure 1. Feistel structure of the traditional GOST algorithm

3. The Properties of S-Boxes

The GOST standard does not specify a set of S-boxes. In fact, one aim of the designers was to have an encryption algorithm with a flexible security level.

Schneier [8] states that the Central Bank of Russian Federation has been using the S-boxes described in Table 2. This set of S-boxes serves as an instance of GOST, but the suitable selection of S-boxes is a design decision according to the standard. It is clear that the selection of the S-boxes has a significant influence on the cryptographic strength of the cipher, thus a careful selection is crucial. Please note that the standard does neither specify if the S-boxes used shall be different. Thus, with a small area footprint in mind, we opt for selecting one S-box that is used eight times in parallel which shown in table 3 – a similar approach as used in DESL/DESXL [9]. While DESL/DESXL lead to a slightly modified standard algorithm.

Table 3. The proposed S-box in E-GOST

x	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
S(*x*)	**8**	**7**	**3**	**C**	**D**	**B**	**4**	**1**	**6**	**A**	**9**	**F**	**0**	**5**	**E**	**2**

The linear and the differential properties are our main concern here and use the classification of 4-bit S-boxes published in [10] as a guideline for the selection of an appropriate S-box. In fact we have chosen to use our proposed S-box, since it has a strong immunity against both linear and differential cryptanalysis and has a low area footprint of 4-bit S-boxes beside the efficient bit-slice implementation especially on constraints microprocessors like x86 architectures. To verify the resistance of the algorithm to differential and linear cryptanalysis [11, 12] and other related attacks, it is necessary to calculate the Maximum Difference Propagation Probability (DPPmax) and the Maximum Input-Output Correlation (IOCmax) as well as the robustness of the S-box. The calculation of both DPPmax and IOCmax are achieved by first building the XOR distribution and Linear Approximation Tables respectively.

3.1. Linear Approximation Table (LAT)

For a given s-box constructed from a mapping $f : Z_2^n \rightarrow Z_2^m$, the linear approximation table entry $LAT(a \cdot b)$ is defined as:

$$LAT(a \cdot b) = \# \ \{x \ \epsilon \ Z_2^n \ | a \cdot x = b \cdot f(x)\} - \ 2^{n-1} \tag{3}$$

where $a \ \epsilon \ Z_2^n$, and $b \ \epsilon \ \{Z_2^m \ \}/0$, and $a \cdot x$ denotes the inner product of the vectors a and x evaluated over Z_2. The linear approximation entry with the maximum absolute values is denoted by L_{max}.

3.2. XOR Distribution Table

For a given s-box constructed from a mapping $f : Z_2^n \rightarrow Z_2^m$, The XOR table entry $N_{\Delta x \Delta y}$ is defined as:

$$\boldsymbol{N_{\Delta x \Delta y} = \ \# \ \{x \ \epsilon \ Z_2^n \ | f(x \oplus \Delta x) \oplus f(x) = \ \Delta y\}} \tag{4}$$

where $\Delta x \epsilon \ Z_2^n$, and $\Delta y \ \epsilon \ \{Z_2^m \ \}$. The entry $N_{oo} = 2^n$, is not taken into consideration as it does not have any cryptographic significance. For $\Delta x \neq 0$, the maximum entry in the XOR table is denoted by D_{max}. Table 3 and Table 4 show the linear approximation and the differential distribution of the Proposed S-box (SP) respectively. Also, appendix A and appendix B show the differential distribution and the linear approximation of the S-boxes used by the Central Bank of Russian Federation (S1 to S8) respectively.

From Table 5, where we summarized the linear and differential characteristics of these S-boxes, it becomes clear that the proposed S-box is stronger in both linear and differential cryptanalysis due to the strict design criteria. Therefore, in the following, we will also consider a GOST implementation that uses eight times the proposed S-box. We will refer to this variant with the term E-GOST while GOST-FB denotes the GOST variant that uses the S-boxes as used by the Central Bank of Russian Federation.

Table 3. Linear approximation table of the proposed S-box

8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

0	2	2	0	0	0	2	2	2	2	0	4	4	2	2	0
0	0	0	0	0	0	4	0	0	0	4	4	0	0	4	0
0	0	2	0	2	2	0	2	2	4	2	2	0	2	0	4
0	2	2	0	0	4	2	2	2	2	0	0	0	2	2	4
0	2	2	0	0	0	2	2	2	2	4	0	4	2	2	0
0	2	0	4	2	2	2	0	0	2	2	2	4	0	2	0
0	0	0	0	0	0	4	0	4	0	0	0	0	4	4	0
0	4	2	0	2	2	0	2	2	0	2	2	0	2	0	4
0	2	2	0	4	0	2	2	2	2	0	0	0	2	2	4
0	2	0	0	2	2	2	4	0	2	2	2	0	4	2	0
0	2	0	4	2	2	2	0	0	2	2	2	4	0	2	0
0	0	4	0	4	4	0	4	0	0	0	0	0	0	0	0
0	0	2	4	2	2	0	2	2	4	2	2	0	2	0	0
0	2	4	0	2	2	2	0	4	2	2	2	0	0	2	0
0	4	2	4	2	2	0	2	2	0	2	2	0	2	0	0

3.3. XOR Distribution Table

For a given s-box constructed from a mapping $\boldsymbol{f} : \boldsymbol{Z_2^n} \rightarrow \boldsymbol{Z_2^m}$, The XOR table entry $N_{\Delta x \Delta y}$ is defined as:

$$N_{\Delta x \Delta y} = \# \{x \ \epsilon \ Z_2^n \ | f(x \oplus \Delta x) \oplus f(x) = \ \Delta y\} \quad (4)$$

where $\Delta x \epsilon \ Z_2^n$, and $\Delta y \ \epsilon \ \{Z_2^m\ \}$. The entry Noo = 2n, is not taken into consideration as it does not have any cryptographic significance. For Δx $\neq 0$, the maximum entry in the XOR table is denoted by Dmax. Table 3 and Table 4 show the linear approximation and the differential distribution of the Proposed S-box (SP) respectively. Also, appendix A and appendix B show the differential distribution and the linear approximation of the S-boxes used by the Central Bank of Russian Federation (S1 to S8), respectively.

From Table 5, where we summarize the linear and differential characteristics of these S-boxes, it becomes clear that the proposed S-box is stronger both with regard to linear and differential cryptanalysis due to the strict design criteria. Therefore in the following we will also consider a GOST implementation that uses eight times the proposed S-box. We will refer to this variant with the term E-GOST while GOST-FB denotes the GOST variant that uses the S-boxes as used by the Central Bank of Russian Federation.

Table 4. XOR distribution table of the proposed S-box

I/O XOR Diff.	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	4	4	0	0	0	0	0	4	0	0	4
2	0	0	0	0	0	2	0	2	0	2	2	4	0	0	2	2
3	0	0	2	2	4	0	0	0	0	2	0	2	2	0	0	2
4	0	0	0	0	0	2	2	4	0	0	0	0	2	4	0	2
5	0	2	2	4	0	0	0	0	2	0	4	2	0	0	0	0
6	0	0	0	0	0	0	2	2	4	2	2	0	2	0	2	0
7	0	2	0	2	4	0	0	0	2	2	0	0	2	0	0	2
8	0	0	0	4	0	0	0	0	0	0	4	0	0	4	4	0
9	0	2	2	0	0	2	2	0	2	0	0	2	2	0	0	2
A	0	2	0	2	4	2	2	0	2	2	0	0	0	0	0	0
B	0	4	0	0	0	2	0	2	0	2	2	0	0	0	2	2
C	0	2	2	0	0	0	0	0	2	0	0	2	0	4	4	0
D	0	2	2	0	0	0	0	4	2	0	0	2	0	4	0	0
E	0	0	2	2	4	2	2	0	0	2	0	2	0	0	0	0
F	0	0	4	0	0	0	2	2	0	2	2	0	2	0	2	0

Table 5. Summary of the non-linear properties of GOST block cipher S-boxes and the proposed S-box

S - Box of Russian Central Bank Federation (GOST-FB)		
S-Box	D_{max}(XOR Dist. Table)	L_{max}(Linear App. Table)
S1	6	± 4
S2	6	± 6
S3	6	± 6
S4	6	± 6
S5	4	6
S6	6	6
S7	8	± 6

S8	8	± 6
The Proposed S - Box (E-GOST)		
S8	4	± 4

3.4. Robustness of E-GOST S-boxes

Seberry [13] defines an expression for the robustness, R, of an S-box. A measure of an S-box's resistance to differential cryptanalysis is robustness (R). Robustness is based on the Differential Distribution Table (DDT) two functions. The first is the number of nonzero components, N, in the DDT's first column (except the first component). These denote cases when an input change leads to no output change. Such events are a weakness because they decrease an algorithm's complexity and play a major role in differential cryptanalysis. The other feature is the largest value found in the DDT, L, other than the top left element. The robustness is given by equation 5:

$$R \ = (1 - N/2^n)(1 \ - L/2^n) \qquad (5)$$

where n is the number of input bits. The higher R is, the more difficult differential cryptanalysis is to perform. Table 6 shows that the proposed S-box has the highest value of R, and then the proposed S-box (SP) can be more resistance to differential cryptanalysis.

Table 6. Robustness of GOST block cipher S-boxes

S - Boxes of Russian Central Bank Federation			
S-Box	*nNZ*	D_{max}	R_S
S1	0	6	0.625
S2	0	6	0.625
S3	0	6	0.625
S4	0	6	0.625
S5	0	4	0.75
S6	0	6	0.625
S7	0	8	0.5
S8	0	8	0.5
S - Boxes of Most Recent Version			
S8	0	4	0.75

4. The Bit-Slice Implementation of The Proposed S-Box

Lightweight block ciphers are designed so as to fit into very constrained environments. They generally aim to be implemented efficiently on a large variety of platforms, but usually not really with software performance in mind. For classical lightweight applications where many constrained devices communicate with a server, it is also crucial that the cipher has good software performance on the server side. In this paper, we consider a context where we have very limited processing resources and throughput requirements. We suggest low-cost encryption schemes (i.e. tiny code size and memory) for processors with a restricted set of instructions (i.e. AND, OR, XOR gates, word rotation and modular addition). Recent work has shown that bit-slice implementations applied to different cryptographic algorithms led to very good software speeds, thus making lightweight ciphers interesting for cloud applications. In this article, we introduce a software implementation of lightweight ciphers on x86 architectures, with a special focus GOST algorithm. We apply to our portfolio of primitives the bit slice implementation trick for 4-bit SBox, which gives good performance, extra side-channels protection, and is quite fit for many lightweight applications.

The x86 processors, which can be found in nearly every personal computer, have some clearly distinguishing features when compared to more modern architectures. One of these is the small number of registers, only 8. Another is the instruction set, where almost all instructions always modify one of their input registers. If those essential properties of the x86 processors have been ignored, thus the result is a high so-called ’register pressure’, meaning compilers must put temporary variables in memory, issuing load and store instructions in addition to the actual computation. When required, the compiler also receives the copying values task.

Cryptographic algorithm implementation is typically optimized for one or more requirements such as latency, performance, power consumption, memory consumption, etc., but also criteria such as the expense of adding masking countermeasures to safeguard against side-channel attacks. It is worthwhile to spend time on this optimization, as the implementations are typically used many times. It is usually a hard problem to find an implementation that is theoretically minimal with respect to the criteria, e.g., general circuit minimization is complete [14]. However, for small functions this is still possible, using, for instance, bit slice solvers. Especially for building blocks that can be used in multiple cryptographic algorithms, such as S-boxes, it is useful to look at methods for finding minimal implementations with respect to some given criteria. we first discuss the problem of finding minimal implementations of the nonlinear functions. We give a sequence method “set of routines” for finding the shortest program to implement the proposed S-boxes.

If we restrict our S-box implementations to the AND; OR; MOVE; XOR; NOT operations, we only need to consider the number of ANDs and ORs. Optimizing for this goal is useful in the case of protecting against side- channel attacks using random masks, where nonlinear gates are typically more expensive to mask. There are also applications in multi-party computation and fully homomorphic encryption, where the cost of nonlinear operations is even more significant. Tiago et al.[15] presents another job where bit-sliced application and masking technique are used to avoid side channel assaults. Constant execution time can be accomplished in bit-sliced execution that helps safeguard against timing attacks. We present a method for finding efficient instruction sequences for the proposed S-box. There are only 8 registers for popular x86 processors, of which even fewer are accessible for computations. Also, the instructions are damaging, replacing one output input. Alternative variants of the S-box guidelines are provided, which require only 5 registers and use parallelism as well.

Another aspect is the encryption speed they allow in different applications. The goal of this work has been to find ways to improve the execution speed of the GOST algorithm on the x86 processors, including use of two-way parallel execution.

The 4-bit S-boxes are 16-element permutations and are performed through simple Boolean operations in a bit parallel (also known as bit slice) style. A 4-digit binary number can represent each number from 0 to 15, so these features map 4 input bits to 4 output bits. Now, we need some way to transform any 4 input bits into the corresponding 4 output bits using only those instructions available in the x86 instruction set, and in a bit parallel way. We’ll propose the S-box represented in table 3 as an efficient implantation:

Now rewrite x and S (x) in binary:

x_3	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
x_2	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
x_1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
x_0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
$S_3(x)$	1	0	0	1	1	1	0	0	0	1	1	1	0	0	1	0
$S_2(x)$	0	1	0	1	1	0	1	0	1	0	0	1	0	1	1	0
$S_1(x)$	0	1	1	0	0	1	0	0	1	1	0	1	0	0	1	1
$S_0(x)$	0	1	1	0	1	1	0	1	0	0	1	1	0	1	0	0

Each column in this table contains the bits of some value for x, as well as the bits of the corresponding S(x). The set of all columns contains all possible values for x. The number of columns is thus determined by the number of possible inputs and it is not related to the word length of any processor.

If we find a way of combining the xi rows by Boolean operations so that we get the S,i rows, then applying those operations to the bits of an input value x is equivalent to looking up S(x). To see how this is done, we will look at the execution of an instruction sequence for S2. The x86 instructions usable for the S-boxes are these:

Instruction	Effect	C expression
and a, b	a := a · b	a &= b
or a,b	a := a + b	a |= b
xor a,b	a := a ⊕ b	a ˆ= b
not a	a := a ⊕ 1	a = ˜a
mov a, b	a := b	a = b

Suppose we have 5 registers, named r0, . . . , r4, available for our computations, and 4 of them initially contain our 4 input bits (ri contains xi, 0 i 3). As r4 is not an input register, we just ignore its previous contents. Thus, we have this initial state:

R4																
R3	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
R2	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
R1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
R0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

The instruction sequence found by the search program (with two-way parallelism shown) is this:

mov r4, r1	and r1, r3
xor r1, r2	xor r3, r0
xor r3, r1	or r2, r4
xor r2, r0	xor r4, r3
mov r0, r2	or r2, r4
xor r2, r1	and r1, r0
xor r4, r1	xor r0, r2
xor r0, r4	not r4

Executing the first line of instructions makes the modifications r4 := r1; r1 := r1 · r3, giving us this new state:

R4	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
R3	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
R2	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
R1	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	1
R0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Next, we perform r1 := r1 ⊕ r2; r3 := r3 ⊕ r0.

R4	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
R3	0	1	0	1	0	1	0	1	1	0	1	0	1	0	1	0
R2	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
R1	0	0	0	0	1	1	1	1	0	0	1	1	1	1	0	0
R0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Now things get more interesting. Notice the values in the r3 row after r3 := r3 ⊕ r1; r2 := r2 + r4.

R4	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
R3	0	1	0	1	1	0	1	0	1	0	0	1	0	1	1	0
R2	0	0	1	1	1	1	1	1	0	0	1	1	1	1	1	1
R1	0	0	0	0	1	1	1	1	0	0	1	1	1	1	0	0
R0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

r3 is now the same as S2(x), one of our wanted output bits. Executing the next three lines of instruction pairs, we reach this state:

R4	0	1	1	0	1	0	0	1	1	0	1	0	0	1	0	1
R3	0	1	0	1	1	0	1	0	1	0	0	1	0	1	1	0
R2	0	1	1	0	0	1	0	0	1	1	0	1	0	0	1	1
R1	0	0	0	0	1	0	1	0	0	0	1	0	1	0	0	0
R0	0	1	1	0	1	0	1	0	0	1	1	0	1	0	1	0

Now r2 is the same as S1(x), The next two lines complete the work:

R4	1	0	0	1	1	1	0	0	0	1	1	1	0	0	1	0
R3	0	1	0	1	1	0	1	0	1	0	0	1	0	1	1	0
R2	0	1	1	0	0	1	0	0	1	1	0	1	0	0	1	1
R1	0	0	0	0	1	0	1	0	0	0	1	0	1	0	0	0
R0	0	1	1	0	1	1	0	1	0	0	1	1	0	1	0	0

Now after completing the work, note that the values in the registers are the same as S(x) according to the following manner:

Register	Equivalent
R4	S3
R3	S2
R2	S1
R0	S0

5. Hardware Implementations

There are different ways for implementing a cryptographic circuit. This paper considers three basic implementation architectures, unrolled, round, and serial as indicated in Figure 2. Efficient implementation has become one of the most challenges in different applications especially, constraint resources devices applications. Therefore, the following metrics have to be taken in consideration if the encryption schemes are implemented in hardware.

- Low gate area which is measured in Gate Equivalents (GE), both memory consumption and implementation size reflect the gate area.
- High throughput; reflects the encryption process speed. It measured in bits or bytes per second,
- Low latency, which defines the time taken to obtain the output of the circuit once its input has been set. It measured in seconds.
- Low power consumption; indicates the amount of power needed to use the circuit. It measured in Watts.

These four criteria compete with one another. For instance, a low latency tends to imply a higher area. Small implementation is also by far slow while the most energy efficient is the largest.

The optimal trade-off between these quantities is very much context dependent. In fact, primitives have been proposed that have been optimized for different corners of the design space: some allow a very low latency implementation, others a very small one (in terms of GE) which is concerned here, etc. Regardless of the exact platform, a given primitive may be implemented using different approach.

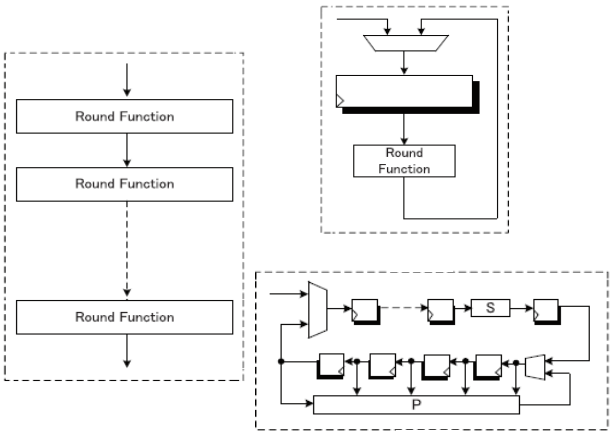
In the case of the hardware implementation of 12 algorithms; Two types of implementation were evaluated: one performing only the encryption operation and the other performing both encryption and decryption with the same module in which the operations are switched by a control signal. Table7 compares the implemented circuit sizes in terms of gate equivariant (area) excluding the interface circuits.

Let us consider unrolled implementation, the different block ciphers gave low latency however much larger than the requirements of the lightweight cryptography. Such unrolled implementations are popular when a low-latency is targeted as those allow a full evaluation in one clock cycle. The downside is then the far larger size of the circuit. Consequentially, serialized and round-based implementations are recommended for E-GOST lightweight hardware implementation. We restrict ourselves to serialized implementation, since it has minimal number of gate equivalents (GE). Beside a very small footprint implementation in hardware E-GOST has a rather high throughput giving good energy-per-bit. Due to the simplicity of the circuit logic involved, serialized implementations allow a far higher clock frequency so they may not be as slow as one might fear. Still, their main advantage is a much smaller circuit.

Figure 2. Basic Implementation Methods (3-a) unrolled, (3-b) round and (3-c) serial implementation.

5.1 Serial Implementation of E-Gost

Figure 3 shows fixed key lightweight hardware architecture of E-GOST with single S-box from [16]. It operates on 4-bit data path chunks; however, the permutation process is considered the most



implementation constraint (challenge). Where the rotation by 11-bit position can't applicable on 4-bit chunks, but instead we should to operate on the whole state. The proposed architecture takes 8 clock cycles to process all chunks of the state and to perform one round of E-GOST.

Then the content of the registers is swapped as it is required by the Feistel structure within one clock cycle, i.e. it operates on the whole state. This clock cycle is used to perform the 11-bit rotation, but in this architecture both halves of the state have XORed already. the right halves in the previous clock cycles have to be shifted by 11-bit positions to the right, before storing it as the new left halve. Then when the XOR sum of both halves is rotated by 11-bit positions to the left, the final step of one round of GOST is performed. In short, the following operations are carried out when the content of the registers is swapped:

$$L_{i+1} = R_i \ll 11 \quad (1)$$

$$R_{i+1} = (L_i \oplus S(K_i + R_i \ mod \ 2^{32})) \ll 11 \quad (2)$$

On the other hand, the key schedule of GOST is very simple: in every round a 32-bit chunk of the 256-bit key is used as the round key and for the last eight rounds the order is swapped (see Table 1). Thus, apart from a 32-bit wide 8- to-1 MUX to select the right round key there is only very little logic required for a round-based implementation. For a serialized implementation we need an additional 4-bit wide 8-to-1 MUX to select the right chunk of the round key. If the application requires the key to be updated, 256 additional flip-flops are required for storage. However, especially in passive RFID-tag scenarios it is very unlikely that the key needs to be changed. Therefore, the main target for our implementations are applications with a fixed key. Then only a small amount of (cheap) tie cells are required to hard-wire the key.

Table7. comparison of the implemented circuit sizes of 12 algorithms in terms of gate equivariant excluding the interface circuits.

Algorithm	Encryption Circuit Size without Interface (kgate)					
	Unrolled Enc	Unrolled Enc/Dec	Round Enc	Round Enc/Dec	Serial Enc	Serial Enc/Dec

AES(table) (128/128)	109.7	205.6	---	---	---	---
AES(comp) (128/128)	76.1	141.5	12.4	15.6	3.2	4.1
Camellia(comp) (128/128)	57.4	60.6	8.0	9.0	4.1	4.3
CLEFIA (128/128)	71.5	71.5	7.3	7.1	3.6	3.8
SIMON (128/128)	60.4	71.3	4.3	5.0	2.1	2.9
SPECK (128/128)	41.6	66.4	4.4	6.8	2.2	3.1
Midori (128/128)	31.8	52.9	4.3	5.6	2.2	2.6
TDES (64/168)	52.8	53.8	5.3	7.9	---	---
LED (64/128)	71.9	212.9	3.8	4.7	3.0	4.3
PRINCE (128/128)	7.8	8.1	2.7	3.0	1.6	1.8
SIMON (64/128)	21.8	25.4	3.2	3.9	1.7	2.5
SPECK (64/128)	17.4	27.8	3.2	4.6	1.8	2.7
Midori (64/128)	10.2	18.5	2.6	3.2	1.5	1.7
SIMON (64/96)	18.4	21.9	2.7	3.2	1.4	2.0
SPECK (64/96)	16.8	26.8	2.8	4.1	1.6	2.3
PRESENT (64/80)	22.0	42.1	2.2	2.9	2.0	2.8
PICCOLO (64/80)	17.4	21.1	1.6	1.9	1.1	1.3
TWINE (64/80)	17.8	23.9	2.7	2.9	2.4	2.5
SIMON (32/64)	7.8	9.2	1.7	2.1	1.0	1.4
SPECK (32/64)	7.0	11.2	1.7	2.4	1.1	1.6

Below we give such a breakdown of both GOST variants. Recall that GOST-FB refers to the variant that uses the set of S-boxes as used by the Central Bank of Russian Federation while E-GOST refers to the variant that uses the proposed S-box eight times.

The sBox Layer module in the serialized E-GOST is by far the smallest, because we only need to implement one S-box, while GOST-FB needs all 8 S-boxes and an additional MUX.

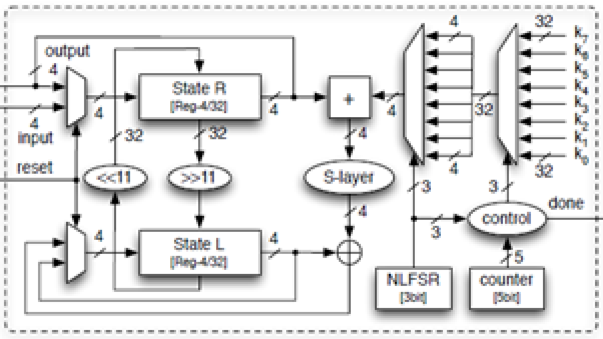


Figure 3. Architecture of a lightweight hardware architecture with a 4-bit data path for GOST.

As one can see, a serialized implementation leads to smaller area requirements, mostly due to a scaled down key addition module (saves 230 GE) and XOR gates (saves 70 GE). However, the serialized architecture also introduces some area overhead, because additional MUXes are required: one for the state register (15 GE) and one to select the right chunk of the round key (49 GE). If the GOST variant uses different S-boxes such as GOST-FB, but not GOST-PS, another MUX is required to select the correct S-box (52 GE). Furthermore, an NLFSR (23 GE) is required as a serial counter and the key addition requires a flip-flop to store the carry bit. The round counter is smaller in the round-based architectures, which we believe is because in the serialized architecture gated registers are required, but not in the round-based. Finally, table 7 states that the smallest footprint implementation in hardware of the algorithms is 1.0 kGE, however, E-GOST has a smaller footprint implementation. E-GOST requires 0.651 KGE.

6. Conclusions

We proposed a new version of GOST called E-GOST that uses a new cryptographically strong proposed S-box since the S-boxes are not specified in the original standard; we discuss the selection of an appropriate approach for the selection of S-boxes. We introduced an efficient bit-slice software implementation on x86 processors architecture as well. We also compare the linear and differential properties of the S-boxes as used by the Central Bank of Russian Federation and the proposed S-box. Finally, we show that E-GOST has the smallest footprint implementation in hardware.

Supplementary Materials: All data sets are included in the article.

Author Contributions: All the authors have contributed equally in the current work.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Serpanos, D., & Wolf, M. (2017). *Internet-of-things (IoT) systems: architectures, algorithms, methodologies*. Springer.
2. Ziegler, S. (Ed.). (2019). *Internet of Things Security and Data Protection*. Springer.
3. Ziegeldorf, J. H., Morchon, O. G., & Wehrle, K. (2014). Privacy in the Internet of Things: threats and challenges. *Security and Communication Networks*, *7*(12), 2728-2742.
4. Serpanos, D. N., & Voyiatzis, A. G. (2013). Security challenges in embedded systems. *ACM Transactions on embedded computing systems (TECS)*, *12*(1s), 1-10.
5. McKay, K., Bassham, L., Sönmez Turan, M., & Mouha, N. (2016). *Report on lightweight cryptography* (No. NIST Internal or Interagency Report (NISTIR) 8114 (Draft)). National Institute of Standards and Technology.
6. Biryukov, A., & Perrin, L. P. (2017). State of the art in lightweight symmetric cryptography. *Esch-sur-Alzette, Luxembourg*. http://hdl.handle.net/10993/31319.
7. De Canni`ere, C. (2005). GOST article. *In: Encyclopedia of Cryptography and Security.* pp. 242-243.
8. Schneier, B. (1996). Section 14.1 Gost, in applied cryptography, second edition. In John Wiley and Sons.

9. Leander, G., Paar, C., Poschmann, A., & Schramm, K. (2007). New lightweight DES variants. In *International Workshop on Fast Software Encryption* (pp. 196-210). Springer, Berlin, Heidelberg.
10. Saarinen, M. (2012). Cryptographic Analysis of All 4 × 4 -Bit SBoxes. *Cryptography*, Springer.

11. Matsui, M. (1993). Linear cryptanalysis method for DES cipher. In *Workshop on the Theory and Application of of Cryptographic Techniques* , Springer, Berlin, Heidelberg, pp. 386-397.

12. Courtois, N. T., & Misztal, M. (2011). Differential Cryptanalysis of GOST. *IACR Cryptol. ePrint Arch.*, *2011*, 312.

13. Seberry, J., Zhang, X. M., & Zheng, Y. (1993, December). Systematic generation of cryptographically robust S-boxes. In *Proceedings of the 1st ACM Conference on Computer and Communications Security* (pp. 171-182).

14. Ullrich, M., De Canniere, C., Indesteege, S., Küçük, Ö., Mouha, N., & Preneel, B. (2011, February). Finding optimal bitsliced implementations of 4× 4-bit S-boxes. In *SKEW 2011 Symmetric Key Encryption Workshop, Copenhagen, Denmark* (pp. 16-17).

15. Tiago, B. S., & Diego, A. (2017). Julio OpezPRESENT runs fast: efficient and secure implementation in software International Association for Cryptologic Research. CHES, pp. 644-664.

16. Poschmann, A., Ling, S., & Wang, H. (2010, August). 256 bit standardized crypto for 650 GE–GOST revisited. In *International Workshop on Cryptographic Hardware and Embedded Systems* (pp. 219-233). Springer, Berlin, Heidelberg.