Further Comments on the Soviet Encryption Algorithm *

C. Charnes, L. O'Connor†
J. Pieprzyk ‡, R. Safavi-Naini, Y. Zheng§

June 1, 1994

1 Introduction

The details of the formerly Soviet (now Russian) encryption algorithm were published in GOST 28147-89 [5]. The aim of the designers was to provide an encryption algorithm with a flexible level of security. The algorithm is an example of DES-type cryptosystem with a drastically simplified key scheduling. It encrypts 64-bit messages into 64-bit cryptograms using 256-bit keys. The GOST document [5] recommends the following four modes: simple substitution mode (electronic codebook mode), stream mode called the (Γ-mode in [5]), stream mode with feedback, and authentication mode.

2 The description of the GOST algorithm

The GOST algorithm consists of 32 iterations (twice more than for the DES). A single iteration is shown in Figure 1. There are two secret components in the algorithm: the 256-bit cryptographic key K and the definition of S-boxes $S_1, \ldots, S_8$.

The cryptographic key $K = (K_0, \ldots, K_7)$ is stored in the key storage unit KSU as a sequence of eight 32-bit words $(K_0, \ldots, K_7)$. The 32-bit word K_i is called a *partial key* $(i = 0, \ldots, 7)$. To encrypt a 64-bit long message, it is first split into two 32-bit parts which are placed into 32-bit registers R_1 and R_2. The contents of register R_1 is added modulo 2^{32} to the partial key K_0 (the adder CM_1), i.e.

$$R_1 + K_0 \pmod{2^{32}}.$$

The resulting 32-bit sequence is divided into eight 4-bit blocks. The eight 4-bit blocks are the inputs to the eight corresponding S-boxes $S_1, \ldots, S_8$. Every S_i, $i = 1, \ldots, 8$, is a permutation. The eight 4-bit outputs of S-boxes are stored in the shift register R where the contents is

*the affiliation of co-authors, except Luke O'Connor, is: The Centre for Computer Security Research, Department of Computer Science, University of Wollongong, Wollongong, NSW 2500, AUSTRALIA

†Distributed Systems Technology Center and Queensland University of Technology.

‡Support for this project was provided in part by the Australian Research Council under the reference number A49131885.

§Support for this project was provided in part by the Australian Research Council under the reference number A49232172

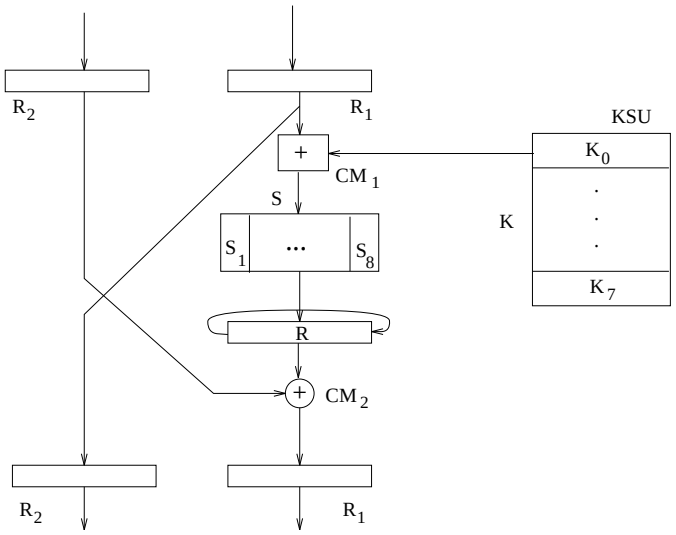

Figure 1: Information flow for a single iteration of the encryption/decryption in GOST

rotated 11 bits the left (towards the high-order bits). The contents of R is now added bitwise (Exclusive-Ored or XORed) to the contents of R_2 by the adder CM_2. The output from CM_2 is stored in R_1 and the old value of R_1 is stored in R_2. This concludes the first iteration.

The other iterations are similar to the first one. In the second iteration, we use the partial key K_1 from the KSU. The iterations 3,4,5,6,7,8 apply partial keys $K_2, K_3, K_4, K_5, K_6, K_7$, respectively. Iterations from 9 to 16 and from 17 to 24 use the same partial keys. The iterations from 25 to 32 apply the reverse order of partial keys, so the 25-th iteration uses the key K_7, the 26-th iteration - the key K_6 and so on. The last iteration uses the key K_0. Thus the order of the partial keys in the 32 iterations is as follows:

$$K_0, \ldots, K_7, K_0, \ldots, K_7, K_0, \ldots, K_7, K_7, \ldots, K_0.$$

After 32 iterations the output from the adder CM_2 is in R_2 and R_1 stores its previous value. The contents of registers R_1 and R_2 is the 64-bit ciphertext (cryptogram) for a 64-bit plaintext.

3 General properties of GOST

The GOST algorithm replicates the general structure of the DES. It is obvious that the designers of the algorithm tried to achieve a balance between the efficiency and the security of the algorithm. They used simple and regular building blocks. In particular, GOST deviates from DES in the following ways:

1. The complicated key schedule has been omitted and replaced by a regular sequence of partial keys.

2. The cryptographic key has been lengthened to 256 bits as compared to 56 bits for DES. Moreover, the actual amount of secret information in the system, including the S-boxes, comprises approximately 610 bits of information.

3. The 8 S-boxes $S_1, S_2 \cdots, S_8$ are permutations $S_i : GF(2^4) \to GF(2^4)$, which require in total the equivalent storage of 2 DES S-boxes.

4. The subkey for each round is combined using 32-bit addition with carry rather than 48-bit XOR as in DES.

5. The irregular permutation block P in DES has been replaced by a simple shift register R which rotates the contents 11 bits to the left after each round.

6. The number of rounds has been increased from 16 to 32.

As the security of the algorithm relies on the secrecy of both the cryptographic key and the eight permutations S_i, $i = 1, \ldots, 8$, users have to know how to select these two secret elements. The cryptographic key can be selected at random but the selection of S_i permutations is left to the central authority who know how to choose "good" permutations. Therefore from the users' point of view, the security is related to the secrecy of their key K. Note that the central authority can select weak permutations (for instance linear or affine), so that they can break the algorithm.

Looking at the structure of the GOST algorithm, one can ask whether the use of permutations instead of much bigger class of all functions, will compromise the security of the system. Even and Goldreich [2] proved that any DES-type encryption with a single iteration of the type

$$\begin{aligned} L' &= R, \\ R' &= L, \oplus f(R) \end{aligned}$$

generates the alternating group, where $f : GF(2^{32}) \to GF(2^{32})$ is a Boolean function, the input is (L, R) and the output is (L', R'). Later Pieprzyk and Zhang [6] showed that if f is a permutation then the DES-type encryption still generates the alternating group. Thus the use of permutations instead of functions does not deteriorate the security of the algorithm when a large number of rounds are considered.

The concatenation of CM_1, S-boxes S, and the cyclic shift R can be seen as a round function F. The F function maps a 32-bit input string into an output string of the same length, subject to the control of a 32-bit subkey. The central part of the F function is eight 4×4 S-boxes. The F function operates by firstly dividing a 32-bit input string into 8 blocks, each of 4-bits, and then substituting each block with four bits specified by the corresponding S-box.

Lemma 3.1 *For a fixed key the F function in GOST is a permutation.*

Proof: Since in GOST the S-boxes are permutations and the cyclic shift is a permutation we only have to consider the adder CM_1. If $x + k \equiv y + k \pmod{2^{32}}$, then $x - y \equiv 0 \pmod{2^{32}}$. This is not possible as $x \leq 2^{32}$ and $y \leq 2^{32}$. □

This result is in contrast to the F function of DES, the latter is discussed by Desmedt, Quisqauter and Davio [1].

4 Avalanche Characteristic of CM_1

For a fixed key we obtain a complete description of the *avalanche characterics* of the GOST adder CM_1, i.e., of the probability distribution of the number of output bits changed if changes are made to one input bit.

We have to take into account the fact that addition by CM_1 is done modulo 2^{32}, and this entails the carry of bits the left.

Let $a_0, a_1, \ldots, a_{31}$ be the the bits of R_1. Changing a_0 (the highest order bit of R_1), changes only one output bit, as this adds 2^{32} to R_1. More generally, the following holds.

Lemma 4.1 *Changing any bit a_i of R_1, changes exactly one output bit with probability* $1/2$.

Proof: If bit a_i is changed, carry only affects the bits a_h, where $h \leq i$. The total number of combinations of these bits is 2^i. The i-th coordinate of exactly 2^{i-1} combinations is zero. And changing the i-th coordinate produces a change in one output bit only for these combinations. Hence the probability of changing only one output bit is $2^{i-1}/2^i = 1/2$. □

Using this lemma, and a more elaborate counting argument we obtain the probability distribution for bits a_i, $0 \leq i < 7$, given in Table 1. This analysis extends to all 32-bits of R_1.

Theorem 4.1 *The probability distribution of changes to output bits, given in* Table 1 *extends to all the bits a_i, $7 \leq i < 31$. That is, changing the a_m-th bit changes k output bits of R_1 with probability:*

$$2^{m-k+1}/2^{m+1}$$

if $1 \leq k < m$, and m-bits are changed with probability $2/2^{m+1}$.

Table 1: Avalanche characteristic.

Bit \Bits changed	1	2	3	4	5	6	7	8
a_0	1							
a_1	$\frac{2}{4}$	$\frac{2}{4}$						
a_2	$\frac{4}{8}$	$\frac{2}{8}$	$\frac{2}{8}$					
a_3	$\frac{8}{16}$	$\frac{4}{16}$	$\frac{2}{16}$	$\frac{2}{16}$				
a_4	$\frac{16}{32}$	$\frac{8}{32}$	$\frac{4}{32}$	$\frac{2}{32}$	$\frac{2}{32}$			
a_5	$\frac{32}{64}$	$\frac{16}{64}$	$\frac{8}{64}$	$\frac{4}{64}$	$\frac{2}{64}$	$\frac{2}{64}$		
a_6	$\frac{64}{128}$	$\frac{32}{128}$	$\frac{16}{128}$	$\frac{8}{128}$	$\frac{4}{128}$	$\frac{2}{128}$	$\frac{2}{128}$	
a_7	$\frac{128}{256}$	$\frac{64}{256}$	$\frac{32}{256}$	$\frac{16}{256}$	$\frac{8}{256}$	$\frac{4}{256}$	$\frac{2}{256}$	$\frac{2}{256}$

5 Cyclic Shifts R in GOST

The primary effect of the cyclic shift is to provide diffusion. To study this we assume that $KSU = 0$ and disregard the effect of the key. Firstly we concentrate only on the mixing effect of the cyclic shift within one arm of the algorithm. We consider two cases of the algorithm in the simple substitution mode:

1. S-boxes are the identity transformation,
2. S-boxes are *complete spread functions*, i.e. every input bit effects every output bit of the S-box, or equivalently every output bit depends on every input bit.

Let R_1^i denote the input to the right-half of the algorithm at round i, and $a_0^i, \cdots a_{31}^i$ the individual input bits to this round.

5.1 Case 1: S-boxes are the identity

We consider the bits affected by a_0^1 – the first input to round 1. Then we have

$$\begin{aligned} a_0^1 \rightarrow\ & a_{11}^2 \ \rightarrow\ a_{22}^3 \ \rightarrow\ a_1^4 \\ \rightarrow\ & a_{12}^5 \ \rightarrow\ a_{23}^6 \ \rightarrow\ a_2^7 \ \Rightarrow \\ a_3^{10} \ \Rightarrow\ & a_4^{13} \ \Rightarrow\ a_5^{16} \ \Rightarrow\ a_6^{19} \\ \Rightarrow\ a_7^{22} \ \Rightarrow\ & a_8^{25} \ \Rightarrow\ a_9^{28} \ \Rightarrow\ a_{10}^{31}. \end{aligned}$$

where $\Rightarrow$ stands for a three round transformation. Hence after 32 rounds a_0^1 occupies every other bit of the R_1-half exactly once. It is easy to see that any other cyclic shift rot(i)–which rotates R_1 by i places, has the same property provided that $\gcd(i, 32) = 1$, or equivalently if i is odd.

5.2 Case 2: S-boxes are complete functions

We consider now the spread of the bit a_0^1 in the case that an input to an S-box affects all the output bits. It can be seen from Figure 2 that the avalanche effect of this bit influences all 32 bits of the R_1-half after 8 rounds. We also note that

$$a_0^1 \Rightarrow a_0^4 \ \rightarrow a_0^5 \ \Rightarrow a_0^7 \rightarrow a_0^8,$$

where $\rightarrow$ denotes a single round and $\Rightarrow$ a multiple round.

The level of spread at each round determines functional dependencies; e.g. if in round 1, 16 bits are affected, then an affected bit in round 4 depends on 16 input bits.

Figures 3 to 6 show the functional dependencies of $a_0^i, i \in \{4, 5, 7, 8\}$ on $a_i^1,\ 0 \leq i \leq 31$. It can be seen that a_0^8 depends on $a_0^1, \cdots a_{31}^1$.

We note that as long as the cyclic shift is not a multiple of 4 spreading occurs and 8 rounds are necessary to affect all the bits of R_1. However the spreading depends on the shift. For example, if the cyclic shift is rot(1) (see Figure 7), the effect of a_0^1 does not reach a_{31}^i for $i < 8$. We can compare rotations by introducing a measure $\rho(i)$–*the minimum number of rounds required so that an affected bit occupies every position in* R_1. It can be seen that $\rho(1) = 8$ and $\rho(11) = 4$.

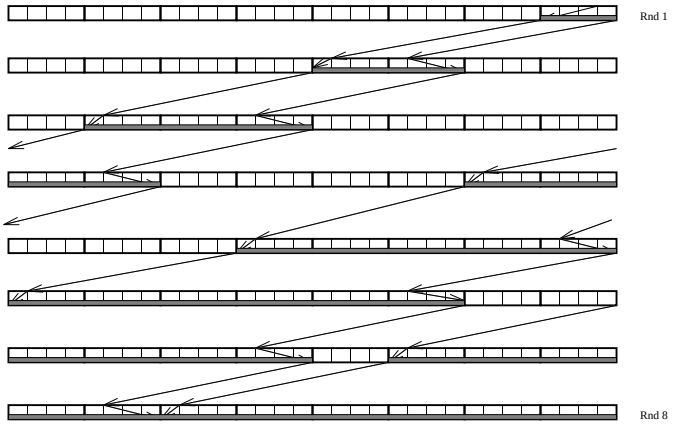

Figure 2: Diffusion of dependencies for a single bit a_0^1 with complete S-boxes

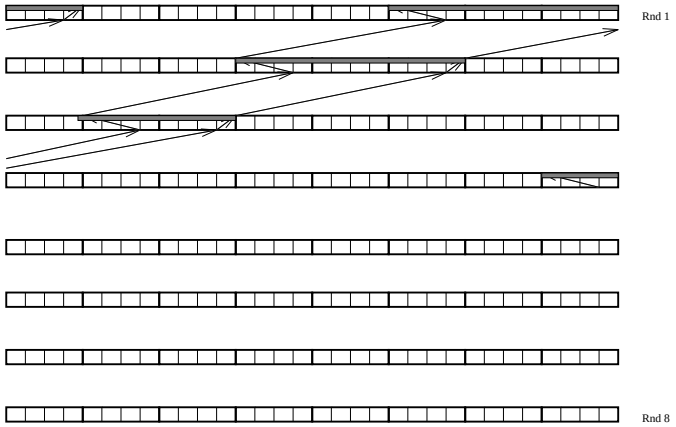

Figure 3: Illustration of dependencies of a_0^4 on a_i^1 $(0 \leq i \leq 31)$

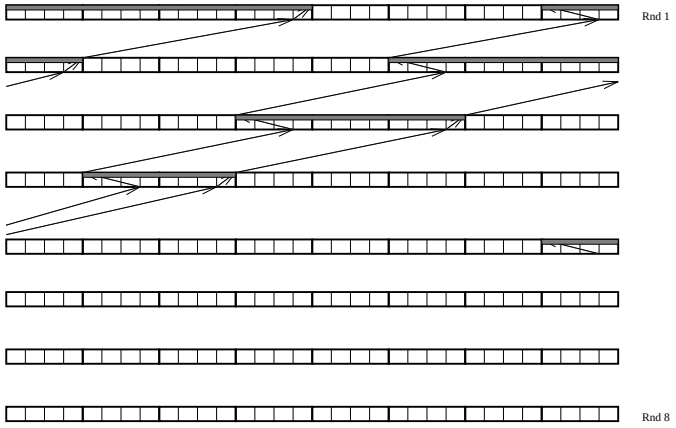

Figure 4: Illustration of dependencies of a_0^5 on a_i^1 $(0 \leq i \leq 31)$

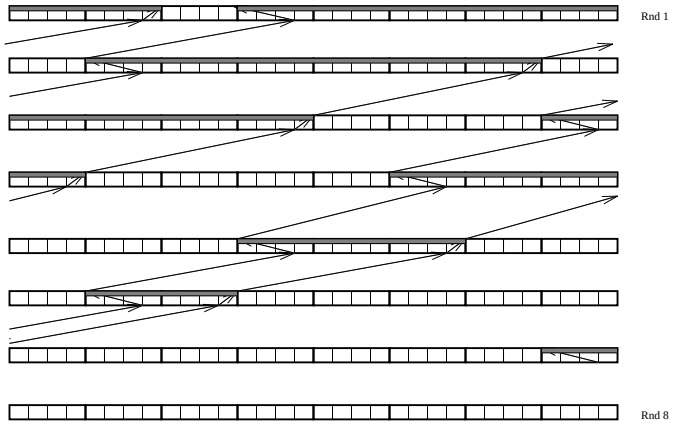

Figure 5: Illustration of dependencies of a_0^7 on a_i^1 $(0 \leq i \leq 31)$

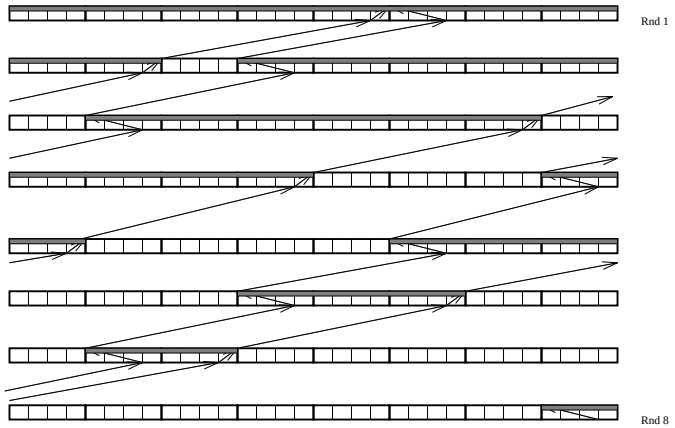

Figure 6: Illustration of dependencies of a_0^8 on a_i^1 $(0 \leq i \leq 31)$

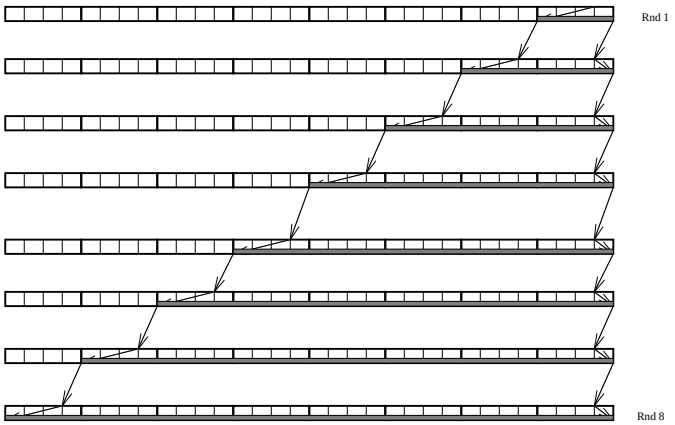

Figure 7: Diffusion of dependences if the cyclic shift is rot(1)

Lemma 5.1 *To compare the effects of rotations we need only consider rotations* rot(i) *for either* $i = 1, 5, \ldots, 29$, *or* $i = 3, 7, \ldots, 31$. $\rho(i)$ *is completely determined by the multiplicity of* 4 *in* i.

Proof: Since $\gcd(i, 32) = 1$, then either $\text{rot}(i) \equiv 1 \pmod 4$, or $\text{rot}(i) \equiv 3 \pmod 4$. Now, changing either 1 bit or 3 input bits affects all 4 output bits of an S-box. □

Using this lemma we observe that the minimum number of rounds such that a bit affected by a_0^1 occupies every position in R_1 at least once is:

Table 2: Spreading induced by rotations.

rot(i)	1/3	5/ 7	9/ 11	13/ 15	17/ 19	21/ 23	25/ 27	29/ 31
$\rho(i)$	8	5	4	5	5	4	5	8

Theorem 5.1 *The absolutely least number of rounds of GOST so that* a_0^1 *occupies every position in* R_1 *at least once is four.*

Proof: We argue that at least four rounds are required for complete diffusion for any rotation. For any rotation rot(i), a 4-bit block is diffused into a 8-bit block after two rounds. After three rounds the 8-bit block is diffused into a 12-bit block. But no matter how these blocks are arranged they cannot cover 32 bits. (At most, if they do not overalp, they cover 4+8+12 = 24 bits.) So at least four rounds are needed for complete diffusion. Table 2 shows that this minimum value is attained by rotations: rot(9), rot(11), rot(21), rot(23), □

Note also that 11 and 23 do not divide $2^{32} - 1$, the modulus of the adder CM_4 (the adder CM_4 is used in the stream mode - see [5]). This could have influenced the choice of rotation by 11 bits for the cyclic shift register.

In the above analysis we have not taken account of swapping of the two halves. To study this we can start the algorithm with $R_2 = 0$. Let R_2^i denote the left-hand input to the algorithm at round i, we also denote rot(i) by r_i and by S the permutation induced by the S-box. ($S : GF(2^{32}) \to GF(2^{32})$). With these conventions the symbolic equations for two rounds of the algorithm are:

$$\begin{aligned} R_1^2 &= R_1 \oplus r_i S r_i S R_1, \\ R_2^2 &= r_i S R_1, \end{aligned}$$

and after three rounds

$$\begin{aligned} R_1^3 &= r_1 S R_1 \oplus r_i S(R_1 \oplus r_i S r_i S R_1), \\ R_2^3 &= R_1 \oplus r_i S r_i S R_1. \end{aligned}$$

Theorem 5.2 *For input to GOST which satisfies the relation* $r_i S(R_1 \oplus r_i S r_i S R_1) = r_i S R_1 \oplus r_i S r_i S r_i S R_1$, *five rounds are required for the diffusion of* R_1 *by the two halves.*

Proof: Using this relation, we see that R_1^5 and R_2^5 contain the expression $r_1 S r_i S r_i S r_i S R_1$. This expression can be rewritten using the relation $r_i S = S' r_i$ for some S'. (Which is a consequence of the fact that the r_i and the S-boxes generate a group.) Using this, we can write

$$r_1 S r_i S r_i S r_i S R_1 = S^{(4)} r_i^4 R_1.$$

($S^{(n)}$ denotes the composition of n S-boxes, i.e., the product of the induced permutations.) Now, Table 2 shows that five rounds are required for diffusion in the two arms of the algorithm, i.e, for rotations: rot(9), rot(11), rot(21), rot(23). □

6 Selection of S-boxes

We note that GOST has an effective key length of approximately 610 bits, where 256 bits are used to represent the key and the remaining bits encode the S-boxes. Each of the 8 S-boxes is a permutation of the integers $[0, 1, \cdots, 14, 15]$, and there are $16! \approx 2^{44.2}$ such permutations. It follows that $8 \cdot 44.2 \approx 354$ bits are required to specify 8 random S-boxes from the set of all 4-bit permutations, giving a total of $256 + 354 = 610$ key bits. To reduce the size of the key required, the designers could alternately generate a collection of S-boxes (a pool) of a relatively small size, say 10,000, and use the key to specify a S-box from this fixed pool.

As noted the set of all possible S-boxes is quite large, and does not permit an exhaustive search to find S-boxes optimal to a set of criteria. Experiments with randomly selected S-boxes have been done and the resulting S-boxes were checked for their resistance to linear and differential cryptanalysis.

6.1 Linear Cryptanalysis

In the case of linear cryptanalysis we are looking for S-boxes $S = [f_1, f_2, f_3, f_4]$, $f_i : GF(2^4) \to GF(2)$, for which each f_i has a high nonlinearity. It is known that the maximum nonlinearity (distance to the set of affine functions) for 4-bit functions is 6, and since the f_i are balanced, the only possible values of nonlinearity are $\{0, 2, 4\}$. ¿From a sample of 10 million S-boxes it was found that the distribution of nonlinearity of the f_i was approximately: 14% at 2, 81% at 4 and less than 5% at 6. Gordon and Retkin [3] have enumerated the class of n-bit permutations for which all f_i are nonlinear, and for $n = 4$, a direct calculation shows that 99% of all mappings are nonlinear in all 4 output functions. Together with the experimental data, these results suggest that randomly selected 4-bit permutations are highly likely to be nonlinear in all output bits, and there is an 86% chance that each output function will have a nonlinearity of 4.

In linear cryptanalysis a linear approximation to a S-box $S = [f_1, f_2, f_3, f_4]$, $f_i = f_i(x_1 x_2 x_3 x_4)$ is considered as follows. Let $i = i_1 i_2 i_3 i_4, j = j_1 j_2 j_3 j_4$ be the 4-bit binary representation of two integers $0 < i, j < 16$. Consider the approximation $L(i, j)$ defined as

$$L(i,j) \;=\; \#\left\{ \sum_X \sum_{k=1}^{4} i_k x_k \;=\; \sum_X \sum_{k=1}^{4} j_k f_k(X) \right\}, \qquad (1)$$

where $X = x_1x_2x_3x_4$ varies over all 16 possible values. It follows that $L(i,j)$ gives the distance of the function $\sum_{k=1}^{4} j_k f(X)$ to the linear function $\sum_{k=1}^{4} i_k x_k$. In linear cryptanalysis we are interested in the values of i,j which maximize $|L(i,j) - 16|$; this is directly related to the nonlinearity of the functions $\sum_{k=1}^{4} j_k f(X)$. Our experiments have shown that most of the $\sum_{k=1}^{4} j_k f(X)$ are likely to have a nonlinearity of 4, so that $|L(i,j) - 16| \leq 12$ most of the time.

If the subkeys are combined using the XOR operation so that adding (XORing) the various round approximations enables the key bits to be isolated, and hence approximated. However this analysis breaks down when the key is added modulo 2^{32} with carry (as in the GOST F function), since the key bits from several round approximations cannot be combined directly (that is, Matsui's Piling-Up lemma does not apply).

6.2 Differential Cryptanalysis

In differential cryptanalysis we are looking for mappings whose XOR tables exhibit a relatively flat distribution (all values are small). It is known [4] that the largest entry in the XOR table of an n-bit permutation is tending to be at most $2n$ as n increases. Applying this result directly to 4-bit mappings indicates that the probability of the most likely input/output difference pair $\Delta X, \Delta Y$ in a 4-bit permutation is approximately $\frac{8}{16} = \frac{1}{2}$. ¿From a sample of 10 million random permutations it was found that 90% had a maximum XOR table value of at most 8, as predicted, with over 60% having a maximum value of 6. We note that no permutations with a maximum value of 2, the so-called differentially 2-uniform mappings, were found in this sample. However, we did find permutations for which a nonzero input difference ΔX leads to a nonzero output ΔY with probability 1. For example, the permutation $\pi = (15,13,9,11,7,14,10,3,2,12,8,6,0,5,1,4)$, where $\pi(0) = 15, \pi(1) = 13$ and so on, all inputs of difference $\Delta X = 3 = 0011$ lead to the output difference $\Delta Y = 4 = 0100$. In fact, 1096 permutations with this property were found.

References

[1] Y. Desmedt, J-J. Quisquater, and M. Davio. Dependence of output in DES: Small avalanche characteristics. In *Advances in Cryptology, Proceedings of CRYPTO'84, Ed G R Blakley, D Chaum, Vol.196*, pages 359–376. Springer-Verlag, 1985.

[2] S. Even and O. Goldreich. Des-like functions can generate the alternating group. *IEEE Transactions on Information Theory*, 29(6):863–865, November 1983.

[3] J. Gordon and H. Retkin. Are big S-boxes best ? In T. Beth, editor, *Cryptography, Proceedings, Burg Feuerstein 1982*, pages 257–262. Springer Verlag, 1983.

[4] L.J. O'Connor. On the distribution of characteristics in bijective mappings. In *Extended Abstracts - Eurocrypt'93*, May 1993.

[5] National Soviet Bureau of Standards. Information Processing Systems. Cryptographic Protection. Cryptographic Algorithm. GOST 28147-89, 1989.

[6] J.P. Pieprzyk and Xian-Mo Zhang. Permutation generators of alternating groups. In *Advances in Cryptology - AUSCRYPT'90, J. Seberry, J. Pieprzyk (Eds), Lecture Notes in Computer Science, Vol.453*, pages 237–244. Springer Verlag, 1990.