Comments on Soviet Encryption Algorithm *

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

Department of Computer Science, University of Wollongong
Wollongong, NSW 2522, AUSTRALIA
charnes/josef/rei/yuliang@cs.uow.edu.au

1 Introduction

The details of the Soviet (now Russian) encryption algorithm were published in GOST 28147-89 [2]. 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 [2] recommends the following four modes: simple substitution mode (electronic code-book mode), stream mode (in [2] called Γ-mode), 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 elements which are secret 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 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 the 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 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 rotated 11 bits left (towards high-order bits). The contents of R is now added bitwise (Exclusive-Ored or XORed) to the

* Support for this project was provided in part by the Australian Research Council grants A49131885 and A9232172

** Distributed Systems Technology Center, Queensland University of Technology, Brisbane, QLD 4001, AUSTRALIA, e-mail: oconor@fit.qut.edu.au

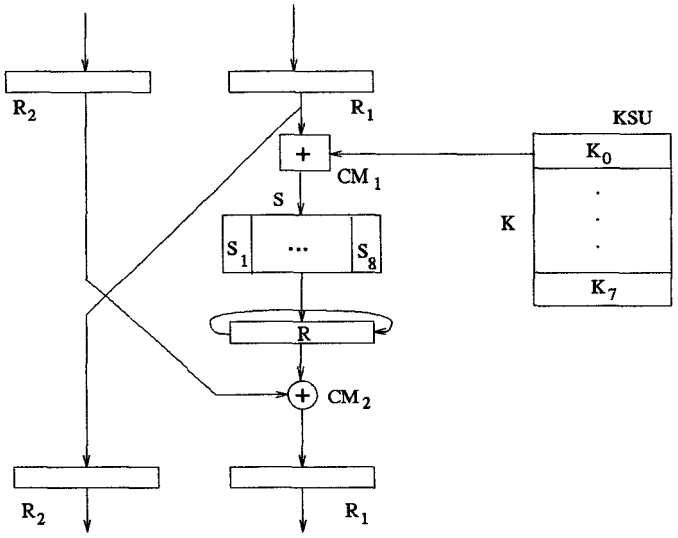

Fig. 1. Information flow for a single iteration of the encryption/decryption in GOST

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. So the order of 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 keeps its previous value. The contents of the registers R_1 and R_2 are the 64-bit ciphertext (cryptogram) for the 64-bit plaintext.

3 General properties of GOST

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

1. The complicated key schedule has been ommitted 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 in total requires the equivalent storage of 2 DES S-boxes.
4. The subley 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 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 knows how to choose "good" permutations. Therefore from users' point of view, the security is related to the secrecy of their key K. Note that the central authority may select permutations in such a way that they can break the algorithm (for instance, by selecting linear or affine permutations).

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 [1] proved that any DES-type encryption with a single iteration

$$L' = R$$
$$R' = L \oplus f(R)$$

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 [3] 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 dividing a 32-bit input string into eight (8) blocks first, each four (4) bits, and then substituting each block with four bits specified by the corresponding S-box.

One can see that output bits of the function F are affected by different collection of inputs depending on their position. This can be explained by properties of the concatenation of addition modulo 2^{32} and S boxes. Addition modulo 2^{32} generates outputs which all (except one - the less significant output bit) are nonlinear. This leads us to the following lemma.

Lemma 1. *Outputs of S_i are affected by $4i$ bits of message from the register R_1 and $4i$ bits of partial key ($i = 1, \ldots, 8$).*

The position of S_1 makes it specially vulnerable to the linear attack. S_1 permutation has to be selected very carefully so the outputs of S_1 have the maximum nonlinearity. As far as F function is concern, GOST compares favourably with the DES as there is more complex input/output relation.

There is a very interesting problem of selecting S boxes in such a way that the nonlinearity of the function F is high. In general the addition increases the nonlinearity but there are cases when you may get reduction of nonlinearity of the function F compared to the nonlinearity of the S-boxes.

4 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. We concentrate firstly 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.

4.1 Case 1: S-boxes are identity

We consider the bits effected 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, will have the same property provided $\gcd(i, 32) = 1$, or equivalently if i is odd.

4.2 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 effects all the output bits. The effect of this bit spreads over all the 32 bits of the R_1 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 effected, then an effected bit in round 4 depends on 16 input bits.

We note that as long as the cyclic shift is not a multiple of 4 spreading occurs and 8 rounds are necessary to effect all the bits of R_1. However the spreading depends on the shift. For example if the cyclic shift is rot(1), 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 effected bit occupies every position in* R_1. It can be seen that $\rho(1) = 8$ and $\rho(11) = 4$.

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 effects all 4 output bits of an S-box. Thus 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. From this remark we conclude that the minimum number of rounds such that a bit effected by a_0^1 occupies every position in R_1 at least once is as shown in Table 1. We note that the minimum value of $\rho(i)$ is 4; this

Table 1. 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

occurs for rotations: rot(9), rot(11), rot(21), rot(23). It can be seen that smallest number of rounds required for complete diffusion is at least 4 for any rotation. For any rotation rot(i), a 4-bit block diffuses into a 8-bit block after 2 rounds. After 3-rounds the 8-bit block diffuses into a 12-bit block. But no matter how these blocks are placed they cannot cover 32 bits. (At most, if they have no overalps, they cover $4 + 8 + 12 = 24$ bits.) So at least 4 rounds are needed for complete diffusion.

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 [2]). 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 2 rounds of the algorithm are:

$$R_1^2 = R_1 \oplus r_i S r_i S R_1,$$
$$R_2^2 = r_i S R_1,$$

and after 3 rounds

$$R_1^3 = r_1 S R_1 \oplus r_i S(R_1 \oplus r_i S r_i S R_1),$$
$$R_2^3 = R_1 \oplus r_i S r_i S R_1.$$

If we assume that

$$r_i S(R_1 \oplus r_i S r_i S R_1) = r_i S R_1 \oplus r_i S r_i S r_i S R_1,$$

we can say something about the diffusion of R_1 by the two halves. Using this relation, we see that after 5 rounds both R_1^5 and R_2^5 contain the term $r_1 S r_i S r_i S r_i S R_1$. But this can be rewritten using the fact that $r_i S = S' r_i$ for some S'. (The various r_i and the S-boxes form a group.) So

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

($S^{(n)}$ denotes the composition of n S-boxes, i.e., the product of the induced permutations.) Consulting Table 1, we see that 5 rounds are required for diffusion in the two arms of the algorithm for rotations: rot(9), rot(11), rot(21), rot(23).

5 Selection of S-boxes

One can note that the 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 $354 \approx 8 \cdot 44.2$ bits are required to specify 8 random S-boxes from the set of all 4-bit permutations, giving a total of $610 = 256 + 354$ 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 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 S-boxes obtained have been checked for their suitability with respect to linear and differential cryptanalysis.

References

1. S. Even and O. Goldreich. Des-like functions can generate the alternating group. *IEEE Transactions on Information Theory*, 29(6):863–865, November 1983.
2. National Soviet Bureau of Standards. Information Processing Systems. Cryptographic Protection. Cryptographic Algorithm. GOST 28147-89, 1989.
3. 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.