Symmetric-key encryption scheme based on the strong generating sets of permutation groups

Ara Alexanyan
Faculty of Informatics and Applied Mathematics
Yerevan State University
Yerevan, Armenia

Hakob Aslanyan
Computer Science Department
University of Geneva
Geneva, Switzerland
hakob.aslanyan@unige.ch

Jose Rolim
Computer Science Department
University of Geneva
Geneva, Switzerland
jose.rolim@unige.ch

Abstract—**In this article we expose a new field of application of such a classical method of computational group theory as the Sims' algorithm. We introduce a symmetric block encryption scheme based on a secret key which is a table of strong generators of a permutation group. We discuss how the proposed scheme can be adopted to be used as an authentication method.**

I. Introduction

The aim of the current article is to introduce how strong generating sets of symmetric group S_n and the Sims' algorithm[1] can be used to design a symmetric-key encryption scheme for a block cipher. We propose a scheme that uses strong generating sets, given in the form of a table, as encryption key and the $cascade$ operation of the Sims' algorithm (see Algorithm1) in the decryption process. The input plaintext is divided into blocks, each of size m, and each block is being encrypted separately into a single permutation over $n = 2m+1$ elements. The input string for decryption is a sequence of permutations over n elements and the key is the above mentioned table. We treat each permutation separately and recover a block of size m from a single permutation.

At the first sight the proposed scheme has very low computational complexity as it is based on simple permutation multiplications and modulo operations on small numbers. The need for cryptographic primitives with low computational complexity is high in systems like wireless sensor network for healthcare or military surveillances, where data security and fast data delivery are crucial for providing a quality of service. Security solutions for such systems must deal with the trade-off between security and performance keeping in mind low computational power of devices composing the system (later is implied by the low cost requirements for system components). Current cryptographic solutions, and particularly the cryptographic primitives, are unsatisfactory for such extreme resource-constrained systems, therefore there is a growing body of work on lightweight cryptography [2], [3], [4].

However the complexity analysis of the proposed scheme and its comparison with the existing lightweight cryptographic primitives is out of the scope of the present article and are left for future consideration.

In section III we discuss how an authentication method can be build based on the proposed scheme.

II. Symmetric Block Cipher

In this section we present the proposed symmetric block cipher after recalling some basic definitions from algebra that can be found in many books such as [5], [6], [7].

A. Preliminaries

Definition 1: Subset S of a group G is a generating set of G, if every element $a \in G$ can be expressed as a combination (under the group operation) of elements of subset S and their inverses

$$a = x_1^{\epsilon_1} x_2^{\epsilon_2} \dots x_k^{\epsilon_k}, x_i \in S, \epsilon_i \in \{1, -1\}, 1 \le i \le k$$

According to the above definition, it is possible to build all the elements of a group by calculating all the possible multiplications of elements of subset S. Of course this is worth doing in case of finite groups only. Algorithmically the presentation of a group as a generating set is not perfect as it is not simple to build all the possible multiplications of elements of set S, even if it is finite. Also the expression of an element of a group by the elements of a generating set is not unique and it is not possible to count the order of a group by having its generating set. Another variant of a generating set which does not have the above faults is a *strong generating set* of a group, which plays a central role in our work. Strong generating set for a group can be build by applying the Sims' algorithm to a given generating set of a group. We give a brief description of the Sims' algorithm below. According to Cayley's Theorem; *if G is a finite group with n elements, then G is isomorphic to one of n-element subgroups of symmetric group S_n.* Therefore the algorithm description below is given in terms of permutation groups.

1) Brief Description of Sims' Algorithm: For building a strong generating set of a group $G \le S_n$, Sims' algorithm [1] takes as input a generating set S of G. It stars with an empty $n \times n$ table (Table *I*), where cells below the diagonal are not used, all the diagonal cells contain the trivial permutation, which maps any i into i, $i = 1, \dots, n$ and the rest of the cells are empty. Hereafter where we refer to a cell we mean a cell over the diagonal. Sims' algorithm repeatedly performs an operation called $cascade$ (Algorithm 1), which is being applied to some permutation a when the table is partially filled in i.e. some cells may have already contain a permutation.

Algorithm 1 Cascade

```
1: procedure CASCADE($a, T^{n\times n}$)        ▷ Permutaion $a$ and the table $T$ of strong generators
2:     $B^n$        ▷ Empty vector of permutations
3:     $b \leftarrow a$        ▷ Assign $a$ to the current permutation $b$
4:     for $i = 1 \rightarrow n$ do
5:         if $b(i) \neq i$ then        ▷ Check if $b$ maps $i$ into $i$
6:             if $T[i][b(i)] = null$ then        ▷ Check if the $b(i)$-th cell of the $i$-th row of the table $T$ is empty
7:                 $T[i][b(i)] \leftarrow b$        ▷ Update the empty cell
8:                 return $T$        ▷ Return the table
9:             else
10:                $B[i] \leftarrow T[i][b(i)]^{-1}$        ▷ Update the vector containing the representation of $a$
11:                $b \leftarrow T[i][b(i)]^{-1} \cdot b$        ▷ Update the current permutation
12:            end if
13:        end if
14:    end for
15:    return $B$        ▷ Return the representation of $a$
16: end procedure
```

TABLE I
THE EMPTY TABLE THE SIMS' ALGORITHM STARTS WITH

	1	2	3	...	n
1	e				
2	■	e			
3	■	■	e		
⋮	■	■	■	⋱	
n	■	■	■	■	e

$Cascade$ either fills an empty cell of a table with the "correct" permutation, i.e. the permutation in the cell with row number i and column number j maps i into j, or goes through all the rows of the table and gets a representation of a by the permutations which are already in the table, by taking exactly one permutation from each row (it can be a trivial permutation for some rows). We give a detailed description of $Cascade$ operation later in Algorithm 1. The Sims' algorithm has two stages, during the first stage $cascade$ is being applied to all permutations from S. As a result some of the cells are assigned a "correct" permutation. In the second stage $cascade$ is applied to all the pairs of permutations which have been filled in a table during the first stage. After the second stage the algorithm is done and the table contains a *strong generating set* of G. Each element of G can be expressed as a multiplication of elements of the resulting table by taking exactly one permutation from each row starting from the first one. After building the table of strong generators of G, representation of any element $a \in G$ can be achieved by applying $cascade$ to a.

Definition 2: Generating set given by the output table of the Sims' algorithm is a **strong generating set**.

B. Encryption Key

Our encryption scheme is based on a secret key, which is a $n \times n$ table of strong generators of symmetric group S_n (as per Sims' Algorithm). Each permutation in a table is over $n = 2m + 1$ elements, where m is the size of a input block to encryption algorithm. All diagonal cells of the table contain the trivial permutation e (i.e. $e(i) = i, 1 \leq i \leq n$, where $p(i)$ stands for the image of i under the permutation p).

Originally the cells of the table are filled by randomly chosen permutations from the sets

$$T_{ij} = \{p \in S_n | p(k) = k, 1 \leq k \leq i-1, p(i) = j\}$$

where T_{ij} contains all the permutations that can be assigned to the cell with row and column numbers i and j respectively (please note that we do not use the Sims' Algorithm for key generation). Obviously, this table is a strong generating set for the symmetric group S_n, and any permutation over n elements can be generated by the elements of this table. The cardinality of a set T_{ij} is $(n-i)!$, thus the number of possible tables (keys) is

$$\prod_{i=1}^{n-1} ((n-i)!)^{(n-i)} = \prod_{i=1}^{n-1} (i!)^i \qquad (1)$$

C. Encryption Algorithm

The input for the proposed encryption algorithm is a binary string divided into consecutive blocks consisting of m bytes[1] and each block is being encrypted separately. We do not use the last row of the table (key), as it contains the trivial permutation only. Encryption of a block is done as follows. For each byte of a block we fix two permutations from consecutive rows of the table, one from each row. Formally we read the first byte in a block and treat it as an unsigned integer, i.e. an integer x_1 such that $0 \leq x_1 \leq 255$. We compute

$$s_1 = x_1 \mod (n-1) + 2$$

where $x_1 \mod (n-1)$ stands for the remainder when x_1 is divided by $n-1$. We have that $2 \leq s_1 \leq n$ and we pick the

[1]Here we use a byte as a unit for simplicity, one can fix any number of bits N, however this will change few parameters of the algorithm

permutation that occupies the $s_1 - th$ cell in the first row of the table. Then we compute

$$s_2 = x_1 \mod (n-2) + 3$$

and pick the permutation that occupies the $s_2 - nd$ cell in the second row of the table. Thus we fixed two permutations from the first two rows of the table using the first byte of the input block. We continue this way choosing two permutations for each byte of the input block. For the k^{th} byte x_k we compute

$$s_{2k-1} = x_k \mod (n-2k+1) + 2k$$

to pick a permutation from the $(2k-1)^{th}$ row and

$$s_{2k} = x_k \mod (n-2k) + 2k + 1$$

to pick a permutation from the $2k^{th}$ row. As a result we fix $2m = n-1$ permutations by picking exactly one permutation from each row of the table. We calculate the multiplication of the fixed permutations starting from the first one and obtain a single permutation, which is the output code for the block. Thus the output code for the entire input string is a sequence of permutations over n elements.

D. Decryption Algorithm

The input string for the decryption algorithm is a sequence of permutations over n elements and the secret key is the table described above. We take the permutations one by one and perform the following action to recover the original bytes. One block of m bytes is being recovered from one permutation. We take the first permutation and apply *cascade* operation of the Sims' algorithm to obtain the representation of the permutation by elements of the table (the key). Because the presentation of an element by a strong generating set is unique, we are guaranteed to get the same permutations that were used during encryption. Thus, we find the cells that contain the permutations returned by *cascade*, one permutation per row. If s_1 is the number of a cell in the first row and s_2 is the number of a cell in the second row then we recover the first byte x_1 using the Chinese reminders theorem; x_1 is the solution of the system

$$\begin{cases} x_1 = (s_1 - 2) \mod (n-1) \\ x_1 = (s_2 - 3) \mod (n-2) \end{cases}$$

As $n-1$ and $n-2$ are co-prime the Chinese reminder theorem gives us the x_1. All the solutions of the system differ by an integer, which is a multiple of $(n-1)(n-2)$ and we are guaranteed that there is a solution in $[0, 255]$ as per encryption scheme. The next byte x_2 is being obtained by the same way from the third and fourth permutations. This time the system is as follows

$$\begin{cases} x_2 = (s_3 - 4) \mod (n-3) \\ x_2 = (s_4 - 5) \mod (n-4) \end{cases}$$

the system for the k^{th} byte is

$$\begin{cases} x_k = (s_{2k-1} - 2k) \mod (n-2k+1) \\ x_k = (s_{2k} - 2k - 1) \mod (n-2k) \end{cases}$$

and by Chinese reminder theorem[2]

$$\begin{aligned} x_k =& ((s_{2k-1} - 2k)(n-2k)[(n-2k)^{-1}]_{n-2k+1} + \\ & (s_{2k} - 2k - 1)(n-2k-1)[(n-2k-1)^{-1}]_{n-2k}) \\ & \mod (n-2k+1)(n-2k) \end{aligned}$$

Thus, we recover initial m bytes one by one.

E. Analysis

1) Notes On The Implementation: The implementation of the above scheme requires some accuracy and attention. One should choose m in such a way as to reduce the redundancy as much as possible and ensure the complete randomness of the encryption table. One can use a table that does not generate all the permutations over n elements, but a subgroup of a symmetric group. On practice it would be better to use only a part of the table, excluding some number of the lower rows of the table, which have very few cells. However, these questions are out of the scope of the present article, which has to expose a new field of application of the Sims' algorithm.
One can implement the scheme by choosing $l > n = 2m + 1$ and using a $l \times l$ table that generates the symmetric group S_l as encryption key. For encryption of a block of m bytes $2m$ rows can be chosen from the table randomly in such a way that the indexes of two consecutive rows that are used to encrypt the same byte of the block are co-prime to ensure the correctness of decryption procedure. From the rows that are not used for the encryption of the current block the trivial permutation e will be chosen. This way the ciphertexts for the same block of m byes encrypted with the same key in different sessions may differ from each other, meanwhile the *cascade* operation of decryption process will still point out all the permutations that were used during encryption process and the trivial permutation for unused rows.

2) Statistics: The fact that the output of the scheme is a sequence of permutations leads to a very statistically uniform string. Our tests showed that standard compression software such as *zip* or *rar* fails to compress the output generated by the scheme. Thus non-random patterns are missing in the ciphertext and statistical analysis on the output of the scheme will fail. It is also a good idea to compress the input string first and then encrypt.

3) Ciphertext only attack: In a ciphertext only attack an adversary has access to a set of ciphertexts and tries to deduce the corresponding plaintexts or the encryption key [8]. In a ciphertext only attack against proposed scheme an adversary knows the size n of the encryption table and possess a set of permutations $P = \{p_1, ..., p_k\}$. The attack is considered successful if he finds the encryption key or the cell numbers that were used for calculating permutations in P (this will allow the attacker to perform the decryption process of the scheme). All the tables/keys of the scheme are strong generating sets of the symmetric group S_n which means that all the tables can produce the set of permutations P with equal probabilities and therefore they are indistinguishable, and there

[2]In the formula $[a^{-1}]_b$ stands for multiplicative inverse of a modulo b.

Algorithm 2 Encryption

```
1: procedure ENCRYPT($X^m, T^{n\times n}$)                        ▷ Plaintext $X$ and encryption table/key $T$
2:     $P \leftarrow e$                                           ▷ Assign a trivial permutation to the current permutation $P$
3:     for $i = 1 \to m$ do
4:         $s_{2i-1} \leftarrow x_i \mod (n-2i+1) + 2i$           ▷ Calculate the index of the first permutation for the $i$-th byte
5:         $s_{2i} \leftarrow x_i \mod (n-2i) + 2i + 1$           ▷ Calculate the index of the second permutation for the $i$-th byte
6:         $P \leftarrow P \cdot T[2i-1][s_{2i-1}] \cdot T[2i][s_{2i}]$   ▷ Multiply the current permutation with the fixed permutations
7:     end for
8:     return $P$                                                 ▷ Return the ciphertext
9: end procedure
```

Algorithm 3 Decryption

```
1: procedure DECRYPT($P, T^{n\times n}$)                          ▷ Ciphertext $P$ and encryption table/key $T$
2:     $X^m$                                                      ▷ Empty vector of bytes
3:     $Q^{2m}$                                                   ▷ Empty vector of permutations
4:     $Q \leftarrow Cascade(P, T)$                               ▷ Perform $cascade$ to get the representation of $P$
5:     for $i = 1 \to m$ do
6:         $s_{2i-1} \leftarrow Lookup(Q[2i-1])$ ▷ Find the index of the cell on the $2i-1$-th row containing the permutation $Q[2i-1]$
7:         $s_{2i} \leftarrow Lookup(Q[2i])$     ▷ Find the index of the cell on the $2i$-th row containing the permutation $Q[2i]$
8:         $X[i] \leftarrow ChineseTheorem(s_{2i-1}, s_{2i}, i, n)$   ▷ Calculate the $i$-th byte of the plaintext
9:     end for
10:    return $X$                                                 ▷ Return the plaintext
11: end procedure
```

is no way for an adversary to find the correct table among all the possible tables (the number of tables is given by (1)).

4) Chosen plaintext attack: In a chosen plaintext attack it is assumed that an adversary has a capability to choose arbitrary plaintexts and get corresponding ciphertexts. The goal of an adversary is to gain some information which reduces the security of the scheme [8]. We discuss a scenario where an adversary tries to deduce the encryptin table (or a part of it) by choosing plaintexts and obtaining corresponding ciphertexts. Instead of picking a random plaintext an adversary can fix one cell on each row of the encryption table and choose a plaintext X in way such that during the encryption of X permutations located in the chosen cells will be used to calculate the ciphertext P of X, i.e. an adversary can obtain the product of permutations that are located in the cells of his choice. Assume an adversary wants to obtain a product of permutations located in the cells $s_1, ..., s_i, ..., s_{2m}$ where s_i is the index of a cell on the i-th row of the table. By picking the k-th byte x_k of the plaintext $X = x_1 x_2 ... x_m$ in a way such that

$$\begin{cases} s_{2k-1} = x_k \mod (n-2k+1) + 2k \\ s_{2k} = x_k \mod (n-2k) + 2k + 1 \end{cases} \tag{2}$$

and obtaining the ciphertext of X an adversary will get a permutation $P = y_{s_1} ... y_{s_{2m}}$ which is a product of permutations located in the cells $s_1, ..., s_{2m}$ and adversary tries to find $y_{s_i}, i = 1, ...2m$. The bytes of X can be calculated from (2) with the help of Chinese reminder theorem.

Assume the adversary repeated the procedure of choosing a plaintext described above l times and obtained the equations

$$\begin{aligned} P_1 &= y_{11} \cdot y_{12} \cdot \dots \cdot y_{12m} \\ P_2 &= y_{21} \cdot y_{22} \cdot \dots \cdot y_{22m} \\ \dots \\ P_l &= y_{l1} \cdot y_{l2} \cdot \dots \cdot y_{l2m} \end{aligned} \tag{3}$$

where y_{ij} is a variable that represents a permutation from the j-th row of the table (for all $i = 1, ..., l$). For some i_1, j and i_2, j the corresponding variables can be identical, i.e. $y_{i_1 j} = y_{i_2 j}$ if adversary wants so.

Having the family of equations (3) adversary can form a system of this equations and try to find a single solution which will give him the permutations used in the table. The difficulties he faces is that the variables y_{ij} and the constants P_k are from the symmetric group S_n which is a non-abelian group and due to [9] the problem of finding a solution to a system of equations consisting of such variables is NP-hard problem. Note that a solution to a system is not enough as the adversary needs to have such a set of equations so that a single solution to the system exists.

It worth mentioning that if one uses an implementation of the scheme with a bigger than $n \times n$ table (as discussed in II-E1), then an adversary can not fix the cells of a table and pick a corresponding plaintext that will be encrypted by the use of permutations contained in the fixed cells as the rows for an encryption session are picked randomly.

III. AUTHENTICATION SCHEME

Below we discuss how the proposed scheme can be used to design an authentication method. Consider the parties

$A_1, ..., A_m$ and B, where users $A_i, 1 \leq i \leq m$ want to authenticate themselves at B. Initially B is given a table (or a set of tables) that generate a subgroup of a symmetric group S_n and users A_i are supplied by permutations of this subset. When A_i wants to authenticate himself, he sends a permutation to B that belongs to subset of B. The later performs $cascade$ to this permutation and verifies the decency of A_i. B may ask for more than one permutation from A_i to prove its decency. The attacker C that does not know the subset of B and does not possess a permutation that belongs to the subset of B faces the problem of guessing the subgroup assigned to B. Assuming C knows the n his only strategy is to send a random permutation (assuming B will not accept the identity element) to B in a hope that it will belong to the subgroup of B. The probability of success is $\sim m/n!$ where m is the order of B's subgroup. B may be given more than one subgroup and/or can ask for more than one permutation to prove user's decency (assuming that B will not accept any permutation that is a combination of already accepted permutations). Of course the parameters m, n and subgroups of B should be chosen carefully to minimize the probability of successful attacks.

IV. Conclusion and Future Work

The current work aims to expose a new field of application of such a classical method as Sims' algorithm. We presented a symmetric block encryption scheme that is based on strong generating sets of permutation groups and we discussed how the proposed scheme can be used for authentication purposes. At the first sight our scheme has a low computational complexity and we point out its complexity analysis and comparison with the existing lightweight encryption schemes as a future work. The formers are required in the systems with low computational resources like RFIDs and wireless sensor networks[10], [11].

Acknowledgment

This work has been partially supported by the IST Programme of the European Union under contract number ICT/FIRE/-STREP257466 (HOBNET).

References

[1] C. C. Sims, "Computational methods in the study of permutation groups," *Computational Problems in Abstract Algebra, Pergamon, Oxford*, pp. 169–183, 1970.

[2] A. Bogdanov, L. R. Knudsen, G. Le, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe, "Present: An ultra-lightweight block cipher," in *the proceedings of CHES 2007*. Springer, 2007.

[3] T. A. S. M. T. Shirai, K. Shibutani and T. Iwata, "The 128-bit blockcipher clefia." no. 4593, 2007, pp. 181–195.

[4] M. Feldhofer, J. Wolkerstorfer, and V. Rijmen, "Aes implementation on a grain of sand," *Information Security, IEE Proceedings*, vol. 152, no. 1, pp. 13 – 20, oct. 2005.

[5] A. Alexanyan, *Algebra (Groups, Rings, Fields)*. Yerevan University Publisher, 2006.

[6] S. Lang, *Algebra*. Springer Science+Business Media, Inc., 2002.

[7] B. V. der Varden, *Algebra*. Springer Verlag, 1968.

[8] P. C. v. O. Alfred J. Menezes and S. A. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 1996.

[9] M. Goldmann and A. Russell, "The complexity of solving equations over finite groups," *Information and Computation*, vol. 178, no. 1, pp. 253–262, 2002.

[10] M. Brown, D. Cheung, D. Hankerson, J. L. Hernandez, M. Kirkup, and A. Menezes, "Pgp in constrained wireless devices," *9th USENIX Security Symposium*, 2000.

[11] D. W. Carman, P. S. Kruus, and B. J. Matt., "Constraints and approaches for distributed sensor networks security," NAI Lab, Tech. Rep. 00-010, September 2000.