cryptography

Brief Report

Cryptanalysis of a Proposal Based on the Discrete Logarithm Problem Inside S_n

María Isabel González Vasco [1,*,†], Angela Robinson [2,†] and Rainer Steinwandt [2,†]

1 MACIMTE, Universidad Rey Juan Carlos, 28933 Móstoles, Madrid, Spain
2 Department of Mathematical Sciences, Florida Atlantic University, Boca Raton, FL 33431, USA; arobin65@my.fau.edu (A.R.); rsteinwa@fau.edu (R.S.)
* Correspondence: mariaisabel.vasco@urjc.es; Tel.: +34-91-488-7605
† These authors contributed equally to this work.

Received: 21 May 2018; Accepted: 16 July 201; Published: 19 July 2018

Abstract: In 2008, Doliskani et al. proposed an ElGamal-style encryption scheme using the symmetric group S_n as mathematical platform. In 2012, an improvement of the cryptosystem's memory requirements was suggested by Othman. The proposal by Doliskani et al. in particular requires the discrete logarithm problem in S_n, using its natural representation, to be hard. Making use of the Chinese Remainder Theorem, we describe an efficient method to solve this discrete logarithm problem, yielding a polynomial time secret key recovery attack against Doliskani et al.'s proposal.

Keywords: cryptanalysis; symmetric group; public key encryption

1. Introduction

Discrete logarithm problems in certain representations of cyclic groups, such as subgroups of elliptic curves over prime fields, are a popular resource in the construction of cryptographic primitives. Widely deployed solutions for digital signatures and key establishment rely on the computational hardness of such discrete logarithm problems. Doliskani et al. proposed a cryptosystem in [1] which relies on the discrete logarithm problem inside the symmetric group S_n, using its standard representation, to be hard. The encryption scheme proposed in [1] is essentially an instantiation of the classic ElGamal [2] encryption scheme, but using a cyclic subgroup of S_n in standard representation as platform instead of a more traditional platform choice.

We show that this particular discrete logarithm problem is problematic for cryptographic purposes by showing how to find such discrete logarithms in polynomial time. Consequently, in the proposal from [1], secret keys can be recovered from public data in polynomial time. Our algorithm exploits the permutation representation of a cyclic group that is used in [1]. Even though any finite cyclic group is isomorphic to a subgroup of a suitably large symmetric group, our algorithm does not imply an efficient solution for discrete logarithm problems in established cryptographic platform groups.

2. The Scheme of Doliskani et al.

In this section, we briefly recall the cryptosystem proposed by Doliskani et al. following the description given in [1] (Section 4). The scheme has the same basic structure as ElGamal encryption:

Key Generation. The key generation algorithm, executed by the receiver, selects an appropriate index n and a suitable permutation $g \in S_n$. The cyclic group generated by g will be denoted by $\langle g \rangle$, and we represent its order by $|g|$. Further, an integer α is selected uniformly at random from $\{1, \ldots, |g| - 1\}$. The public key is the pair (g, g^α), while the private key is the secret "exponent" α. (Even though these points are not clarified by the authors, as is customary, we assume n is chosen from an input security parameter ℓ, and is polynomial in ℓ.)

Encryption. On input of a plaintext m, which we may assume belongs to S_n (we omit the encoding described in [1] (Section 3), which is irrelevant for our purposes), an integer k is chosen uniformly at random from $\{1, \ldots, n\}$. The ciphertext is computed as the pair of group elements $(g_1, g_2) := (g^k, mg^{\alpha k})$.

Decryption. The group element g_1 is raised to the secret exponent α and further inverted to compute $m := g_2(g_1^{\alpha})^{-1}$.

As made clear by the authors, this scheme essentially instantiates ElGamal encryption in the symmetric group S_n. As such, some of the security concerns of the original ElGamal over finite fields carry over. Given an encryption of m, one can trivially derive an encryption of hm for any $h \in S_n$, so we observe that malleability is one such concern. Very limited plaintext leakage is another concern. It is known that in a straightforward ElGamal implementation over $\mathbb{Z}^*_{2q+1}$ for a Sophie Germain prime q, one bit of the message leaks. Indeed, if the cyclic group $\langle g \rangle$ has order q, it is possible to determine from the ciphertext whether the underlying plaintext m is a quadratic residue mod $2q + 1$ or not, as the ciphertext leaks the Legendre symbol $\left(\frac{m}{2q+1}\right)$ of m.

Similarly, the construction in [1] leaks one bit, corresponding to the *sign* of the plaintext permutation m. Recall that the sign of a permutation can be seen as a group homomorphism

$$\varepsilon : S_n \longrightarrow \{-1, 1\},$$

defined as $\varepsilon(\sigma) = 1$ if and only if σ can be written as the product of an even number of transpositions. Otherwise, if σ is odd (and can thus only be decomposed as a product of an odd number of transpositions), $\varepsilon(\sigma) = -1$. It is easy to see that ε is a group homomorphism, hence $\varepsilon(mg^{\alpha k}) = \varepsilon(m)\varepsilon(g^{\alpha k})$. If any of the (public) g, g^{α}, or g^k are in the kernel of ε, then necessarily the sign of the "mask" $g^{\alpha k}$ is one, too, and the sign of the plaintext leaks. Information on the elements in $\{1, \ldots, n\}$ not stabilized by the permutation m, known as the support of plaintext m, may leak if the support of g is *small*. This follows from the fact that the elements in $\{1, \ldots, n\}$ not stabilized by any permutation from $\langle g \rangle$ is always a subset of the support of g.

We include these remarks to emphasize that, when considering a concrete ElGamal instantiation, a thorough analysis is essential. One must consider the specific group representation and parameters in use.

In [1], the order of the public group element g is identified by the authors as the main relevant parameter determining the security of the above scheme. Indeed, when approaching a *generic* instance of the discrete logarithm problem in an arbitrary cyclic group, the order of the generator gives us an idea of how successful standard methods such as those mentioned in [1] (Section 2) might be when it comes to solving the associated discrete logarithm problem. This, however, does not rule out the existence of more efficient algorithms for computing discrete logarithms exploiting a concrete representation of the underlying group. As we show in the next section, this is the case for the symmetric group.

3. Finding Discrete Logarithms in Cyclic Subgroups of S_n

Let S_n be the symmetric group on n points, with elements $f \in S_n$ represented as a list of images $[f(1), \ldots, f(n)]$ (or in standard cycle notation). Moreover, let $g \in S_n$, and $h = g^{\alpha}$ some element in the cyclic group $\langle g \rangle$ generated by g. For the encryption scheme put forward in [1], the pair (g, h) represents a public key, and being able to recover α from the public key yields a successful recovery of a user's secret key. When applied to the input (g, h), the following procedure returns $\alpha \pmod{|g|}$, thereby solving the discrete logarithm problem in $\langle g \rangle$.

Step 1. Decompose g and h into disjoint cycles

$$\begin{aligned} g &= \pi_1 \circ \ldots \circ \pi_r \\ h &= \sigma_1 \circ \ldots \circ \sigma_s. \end{aligned}$$

Here, we include length-one cycles if needed, so that each $i \in \{1, \ldots, n\}$ occurs in exactly one cycle.

Step 2. Compute arrays `G` and `H`, such that the ith entry `G[i]` stores:

- the index j of the cycle π_j containing i; and
- the position of i within this cycle ($1 \leq i \leq n$).

That is, $\texttt{G}[i] = (j, \mathrm{pos}(i))$ would indicate that element i appears in cycle π_j at position $\mathrm{pos}(i)$. Similarly, in `H[`i`]`, we store:

- the index k of the cycle σ_k containing i; and
- the position of i within this cycle ($1 \leq i \leq n$).

Thus, $\texttt{H}[i] = (k, \mathrm{pos}(i))$ would indicate that element i appears in cycle σ_k at position $\mathrm{pos}(i)$.

Step 3. Store the first element of each cycle σ_j of h as `First[`j`]` in an array. Analogously, store the second element of σ_j as entry `Second[`j`]` in an array. (For a length-one cycle, we set `Second[`j`]=First[`j`]`.) Note that `First[`j`]` and `Second[`j`]` belong to the same cycle $\pi_{j'}$ of g.

Step 4. Use the array `G` to find for each $i \in \{1, \ldots, n\}$ the cycle of g containing `First[`i`]` and `Second[`i`]`, and store the difference `D[`i`]` between their positions in an array `D`. Then, $\texttt{D}[i] = \mathrm{pos}(\texttt{Second}[\mathrm{i}]) - \mathrm{pos}(\texttt{First}[\mathrm{i}])$, for each $i \in \{1, \ldots, n\}$. Further, compute the length of the cycle containing element i and store it in an array `L`.

Step 5. The solution α is congruent to each residue $\texttt{D}[i]$ modulo $\texttt{L}[i]$ for $1 \leq i \leq |\texttt{D}|$. Compute α with the Chinese Remainder Theorem.

It may be worth noting that the last step of the above procedure uses a slightly more general version of the Chinese Remainder Theorem than is commonly discussed in introductory computer algebra courses. Instead of exploiting the availability of an efficiently computable isomorphism between $\mathbb{Z}/(m_1 \cdot \cdots \cdot m_r)$ and $\mathbb{Z}/(m_1) \times \cdots \times \mathbb{Z}/(m_r)$, with $m_1, \ldots, m_r$ being pairwise coprime natural numbers, we face the more general situation of a linear system of congruences of the form

$$\begin{aligned} x &\equiv x_1 \pmod{m_1} \\ &\vdots \\ x &\equiv x_2 \pmod{m_r} \end{aligned}$$

where $m_1, \ldots, m_r$ may have common factors. This situation is covered, e.,g., in [3] (Theorem 3.12) and in [4], which show that a solution is unique modulo the least common multiple of $m_1, \ldots, m_r$, and for executing Step 5 we basically follow the proof given in [4]. Putting everything together, it turns out that the running time of the above procedure is polynomial. (As is common, we use the (bit) length of the group size as cost parameter. With the natural representation of S_n used, the running time is also polynomial in the input length.)

Theorem 1. *Let $g \in S_n$. Then, the discrete logarithm problem in the group generated by g can be solved in time $\mathcal{O}(\log^4 |g|) = \mathcal{O}(n^2 \log^2 n)$.*

Proof. Let $g \in S_n$. It is easy to see that Step 1 from the above description can be completed in time $\mathcal{O}(n)$. Indeed, to express g in cycle notation, we assume (without loss of generality) it acts on $\{1, \ldots, n\}$. Thus, we start from $i = 1$, perform a look-up and find the image of i under g. If the image is equal to i, close the cycle and increment the index i moving ahead to $i + 1$. Otherwise, append $g(i)$ at the end of the cycle and repeat the process for this index. There will be at most n look-ups and n stored integers between 1 and n. The arrays `G`, `H` each contain $2n$ integers.

Further, Step 2 can also be completed in time $\mathcal{O}(n)$. As there are at most n cycles in g^α, the arrays `First`, `Second` are at most n integers long. Thus, the construction of these two arrays requires storing at most $2n$ integers.

Let us now move ahead to Steps 3 and 4. For each $1 \leq i \leq |\texttt{First}|$, perform a look-up in array G to determine to which cycle of g the value First[i] belongs. This requires at most n look-ups. Look up the position numbers of Second[i] and First[i] and subtract. This requires at most $\mathcal{O}(n)$ computations plus $\mathcal{O}(n)$ look-ups.

The final step requires that we solve a system with at most $|\texttt{D}|$ modular arithmetic equations, where the moduli are not necessarily coprime. We have $|\texttt{D}| \leq n/2$, so let $k = \lceil n/2 \rceil$, and let

$$\begin{aligned}
\alpha &\equiv \texttt{D}[1] \mod \texttt{L}[1] \\
\alpha &\equiv \texttt{D}[2] \mod \texttt{L}[2] \\
&\vdots \\
\alpha &\equiv \texttt{D}[k] \mod \texttt{L}[k]
\end{aligned}$$

denote the system of congruences found in Step 5, where each $\texttt{L}[i]$ is the length of a cycle of g. As in [4], let $m = \operatorname{lcm}(\texttt{L}[1], \texttt{L}[2], \ldots, \texttt{L}[k])$. Now, we can closely follow the the proof of [4, Theorem 2]:

Compute the solution to the first two congruences

$$\begin{aligned}
\alpha &\equiv \texttt{D}[1] \mod \texttt{L}[1] \\
\alpha &\equiv \texttt{D}[2] \mod \texttt{L}[2],
\end{aligned}$$

and call this solution α_1. There are $t, s \in \mathbb{Z}$ with $\gcd(\texttt{L}[1], \texttt{L}[2]) = t \cdot \texttt{L}[1] + s \cdot \texttt{L}[2]$. By [4], we know the solution is $\alpha_1 = \texttt{D}[1] + t \cdot \texttt{L}[1]$, which is unique $\mod \operatorname{lcm}(\texttt{L}[1], \texttt{L}[2])$. According to [5], this application of the Extended Euclidean Algorithm has a cost of $\mathcal{O}(\log \texttt{L}[1] \cdot \log \texttt{L}[2])$. We upper-bound this by $\mathcal{O}(\log^2 n)$.

Next, consider the two equivalences

$$\begin{aligned}
\alpha &\equiv \alpha_1 \mod \operatorname{lcm}(\texttt{L}[1], \texttt{L}[2]) \\
\alpha &\equiv \texttt{D}[3] \mod \texttt{L}[3].
\end{aligned}$$

Compute the solution to this pair of congruences as above, and call this solution α_2. The cost of computing α_2 is

$$\begin{aligned}
\mathcal{O}(\log(\operatorname{lcm}(\texttt{L}[1], \texttt{L}[[2])) \cdot \log \texttt{L}[3]) &= \mathcal{O}(2 \log n \log n) \\
&= \mathcal{O}(\log^2 n).
\end{aligned}$$

Iterate this step until the k equivalences are reduced to 2. The solution to the last pair of equivalences is the solution, α.

There will be at most $n - 1$ applications of the Extended Euclidean Algorithm, with total complexity in $\mathcal{O}(\sum_{k=1}^{n-1} k \cdot \log^2 n) = \mathcal{O}(n^2 \log^2 n)$. From [6–8], we know that, for any $g \in S_n$, it holds that $\log |g| = \mathcal{O}(\sqrt{n \log n})$, and the claim follows. □

Correctness of the above procedure is not hard to verify:

Proposition 1. *For any $g \in S_n$ and $h \in \langle g \rangle$ such that $h = g^\alpha$, the above procedure computes $\alpha \pmod{|g|}$, given g, h, and n.*

Proof. Let $g = \pi_1 \circ \cdots \circ \pi_r$ and suppose the algorithm returns $\bar{\alpha} \equiv \texttt{D}[i] \mod \texttt{L}[i]$ for all i as in the proof of Theorem 1. We proceed by showing that $g^{\bar{\alpha}} = h$. Since the π_i are disjoint,

$$\begin{aligned} g^{\bar{\alpha}} &= (\pi_1 \circ \cdots \circ \pi_r)^{\bar{\alpha}} \\ &= \pi_1^{\bar{\alpha}} \circ \cdots \circ \pi_r^{\bar{\alpha}}. \end{aligned} \tag{1}$$

There exist $k_i \in \mathbb{Z}$ such that $\bar{\alpha} = k_i \cdot \texttt{L}[i] + \texttt{D}[i]$ for all i, so (1) is equal to

$$\pi_1^{k_1 \texttt{L}[1]+\texttt{D}[1]} \circ \cdots \circ \pi_r^{k_r \texttt{L}[r]+\texttt{D}[r]}. \tag{2}$$

The order of π_i is $\texttt{L}[i]$ for each i, so Equation (2) simplifies to

$$\pi_1^{\texttt{D}[1]} \circ \cdots \circ \pi_r^{\texttt{D}[r]}.$$

To show that $g^{\bar{\alpha}} = h$, we evaluate $g^{\bar{\alpha}}(\texttt{First}[i])$ and show that the result is $\texttt{Second}[i]$ for all i. Let $\texttt{G}[\texttt{First}[i]] = (l, \mathrm{pos}(\texttt{First}[i]))$. As $\texttt{First}[i]$ and $\texttt{Second}[i]$ belong to the same cycle of g, then $\texttt{G}[\texttt{Second}[i]] = (l, \mathrm{pos}(\texttt{Second}[i]))$. It follows that

$$\pi_1^{\texttt{D}[i]} \circ \cdots \circ \pi_r^{\texttt{D}[r]}(\texttt{First}[i]) = \pi_l^{\texttt{D}[i]}(\texttt{First}[i]).$$

The image of $\texttt{First}[i]$ under $\pi_l^{\texttt{D}[i]}$ is found by moving (cyclically) right by $\texttt{D}[i]$ positions inside π_l. Thus, $\texttt{First}[i]$ ends up being mapped to the cycle entry at position $\mathrm{pos}(\texttt{First}[i]) + (\mathrm{pos}(\texttt{Second}[i]) - \mathrm{pos}(\texttt{First}[i])]) = \mathrm{pos}(\texttt{Second}[i])$. Consequently, $\pi_l^{\texttt{D}[i]}(\texttt{First}[i]) = \texttt{Second}[i]$. As this holds for all i, the resulting permutation satisfies $g^{\bar{\alpha}} = h$. $\square$

4. Experimental Validation

The proposed attack was implemented in Magma V2.21 on a personal computer. An example of the attack in S_{100} is as follows. Let

$$\begin{aligned} g \;=\; & (1,12,90,19,7,30,44,72,57,55,34,81,82,17,54,21,80,94,35,11,85,100) \\ & (2,9,83,87,45,13,67,24,78,4,16,32,65,51,29,33,22,59,50,69,56,58,43, \\ & 31,47,96,91,92,15,75,86,49,68,88,95,36,63,23,71,98,42,28,64,8,38,40) \\ & (3,10,97,48,74,39,46,60,89,5)(6,26,79,25,20,76)(14,84,37,53,61,70,73) \\ & (18,99,93,66,62,27,77,41). \end{aligned}$$

The order of g is 212,520. Given $(g, g^{178{,}705})$, let us try to recover the secret exponent α =178,705. Following the procedure presented above, we store

$$\begin{aligned} \texttt{D} &= [21,41,5,-5,1,5,2,1,5,-5,0] \text{ and} \\ \texttt{L} &= [22,46,10,10,6,10,7,8,10,10,1]. \end{aligned}$$

Further, we know that α is congruent to $\texttt{D}[i]$ modulo $\texttt{L}[i]$ for each i. Applying the Chinese Remainder Theorem yields the solution α = 178,705, as expected.

5. Conclusions

The above discussion provides a polynomial time solution for the discrete logarithm problem inside the symmetric group S_n, using its standard presentation. On suitable elliptic curves, efficient implementations of ElGamal are available, where (in the absence of quantum computers) no polynomial time attacks on the secret key are known. With the availability of a polynomial-time secret key recovery, it seems fair to consider the security assumption underlying Doliskani et al.'s proposal as problematic.

Author Contributions: Conceptualization, Formal Analysis, and Writing: M.I.G.V, A.R., and R.S.; Software: A.R.

Funding: This research was funded by Spanish MINECO grant number MTM2016-77213-R.

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

References

1. Doliskani, J.N.; Malekian, E.; Zakerolhosseini, A. A Cryptosystem Based on the Symmetric Group S_n. *IJCSNS Int. J. Comput. Sci. Netw. Secur.* **2008**, *8*, 226–234.
2. Gamal, T.E. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Inf. Theory* **1985**, *31*, 469–472.
3. Jones, G.A.; Jones, J.M. *Elementary Number Theory*; Springer Undergraduate Mathematics Series; Springer: Berlin, Germany, 1998.
4. Bogomolny, A. Chinese Remainder Theorem from Interactive Mathematics Miscellany and Puzzles. 2012. Available online: http://www.cut-the-knot.org/blue/chinese.shtml (accessed on 1 May 2018).
5. von zur Gathen, J.; Gerhard, J. Chapter The Euclidean Algorithm. In *Modern Computer Algebra*; The Press Syndicate of the University of Cambridge: Cambridge, UK, 1999; pp. 50–55.
6. Landau, E. Über die Maximalordnung der Permutationen gegebenen Grades. *Arch. Math. Phys.* **1903**, *5*, 92–103.
7. Massias, J.P. Majoration explicite de l'ordre Maximum d'un Élément du groupe symétrique. *Ann. Fac. Sci. Toulouse Math.* **1984**, *6*, 269–280. [CrossRef]
8. Massias, J.P.; Nicolas, J.L.; Robin, G. Effective Bounds for the Maximal Order of an Element in the Symmetric Group. *Math. Comput.* **1989**, *53*, 665–678. [CrossRef]

© 2018 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).