Extending Baby-step Giant-step algorithm for FACTOR problem

Martin Stanek[*]

Department of Computer Science
Comenius University
stanek@dcs.fmph.uniba.sk

Abstract. Recently, a non-abelian factorization problem together with an associated asymmetric encryption scheme were introduced in [1]. We show how a classical baby-step giant-step algorithm for discrete logarithm can be extended to this problem. This contradicts the claims regarding the complexity of the proposed problem.

1 Introduction

Baba, Kotyada, and Teja [1] introduced a factorization problem over non-abelian finite groups. Let $(G, \cdot)$ be a (non-commutative) finite group with identity e. Let $[g]$, for $g \in G$, denotes the subgroup generated by element g. Let $g, h \in G$, and we assume that $[g] \cap [h] = \{e\}$. A function $f : [g] \times [h] \to G$ is defined as follows:

$$f(g^x, h^y) = g^x h^y.$$

It is easy to verify that f is an injective mapping. The FACTOR problem is defined in the following way: given $z \in \mathrm{Range}(f)$ compute $f^{-1}(z)$, i.e. $g' \in [g]$ and $h' \in [h]$ such that $g'h' = z$.

The authors of [1] claim that FACTOR problem is more difficult than discrete logarithm problem (DLP). We show that this is not generally true, and the FACTOR problem can be solved by modification of baby-step giant-step algorithm in time $O(\sqrt{nm})$, where $n = \mathrm{ord}(g)$ and $m = \mathrm{ord}(h)$. Therefore, for groups where the best approach to DLP is a generic algorithm, the FACTOR problem is at most as difficult as DLP.

In the following section we present two generalizations of baby-step giant-step algorithm. The first one requires that the orders of g and h are relatively prime, the second algorithm is universal and works for arbitrary orders.

2 Solving FACTOR problem

2.1 Algorithm for co-prime ord(g) and ord(h)

We assume that $n = \mathrm{ord}(g)$ and $m = \mathrm{ord}(h)$ are relatively prime, i.e. $\gcd(n, m) = 1$. Let $l = \lceil \sqrt{nm}\, \rceil$. The main idea is to use baby-step giant-step algorithm for

[*] supported by VEGA 1/0266/09

discrete logarithm, but "translate" the exponent into pair of exponents (and preserve addition over both representations). Moreover, we must take into account non-commutativity of G and pay attention to the correct order of multiplications. We define a function $\varphi : \mathbb{Z}_{nm} \to \mathbb{Z}_n \times \mathbb{Z}_m$ as follows:

$$\varphi(k) = (k \bmod n, k \bmod m).$$

For brevity, we denote the first (second) coordinate of φ as φ_1 (φ_2), i.e. $\varphi_1(k) = k \bmod n$ ($\varphi_2(k) = k \bmod m$). Since $\gcd(n, m) = 1$, φ is an isomorphism between $(\mathbb{Z}_{nm}, +)$ and $(\mathbb{Z}_n, +) \times (\mathbb{Z}_m, +)$ (it follows from the Chinese remainder theorem).

Algorithm. Let $z \in \mathrm{Range}(f)$ be an input. The algorithm computes $(x, y) \in \mathbb{Z}_n \times \mathbb{Z}_m$, such that $z = g^x h^y$. The algorithm has a pre-computation phase (this phase does not depend on input), and an online phase.

Pre-computation (giant steps). We compute values $g^{\varphi_1(jl)} h^{\varphi_2(jl)}$, for all $j = 0, 1, \dots, l-1$, and store them together with value j in a hash table S. The complexity of this phase is $O(l)$, i.e. $O(\sqrt{nm})$.

Online (baby steps).

```
for $i = 0, 1, \dots, l-1$:
    if $(g^{\varphi_1(i)} \cdot z \cdot h^{\varphi_2(i)} \in S)$:
        let $j$ be the index corresponding to the found value
        return $(\varphi_1(jl - i), \varphi_2(jl - i))$
```

The correctness of the online phase follows from the fact that φ is an isomorphism, and from the following derivation:

$$\begin{aligned} g^{\varphi_1(i)} \cdot z \cdot h^{\varphi_2(i)} &= g^{\varphi_1(jl)} h^{\varphi_2(jl)} \\ &\Downarrow \\ z &= g^{\varphi_1(jl) - \varphi_1(i)} h^{\varphi_2(jl) - \varphi_2(i)} \\ &= g^{\varphi_1(jl-i)} h^{\varphi_2(jl-i)} \end{aligned}$$

Using the fact that lookup in the hash table takes a constant time, the complexity of this phase is $O(\sqrt{nm})$.

2.2 Universal algorithm

The following algorithm solves FACTOR problem for arbitrary values of $n = \mathrm{ord}(g)$ and $m = \mathrm{ord}(h)$. Without loss of generality we assume $n \geq m$. Let $l = \lceil \sqrt{nm} \rceil$. The main idea is similar to the previous algorithm but the "translation" is different.

Algorithm. Let $z \in \mathrm{Range}(f)$ be an input. The algorithm computes $(x, y) \in \mathbb{Z}_n \times \mathbb{Z}_m$, such that $z = g^x h^y$.

Pre-computation (giant steps). We compute values $g^{il}h^j$, for $0 \le i < \lceil n/l \rceil$ and $0 \le j < m$, and store them together with corresponding values i, j in a hash table S (the assumption $m \le n$ prevents storing more than $O(l)$ values). The complexity of this phase is $O(m \cdot \frac{n}{l}) = O(\sqrt{nm})$. The values of Range$(f)$ can be visualized on $n \times m$ grid where a position (a, b) corresponds to the element $g^a h^b$. Then the values stored in S correspond to grey boxes, see Fig. 1.

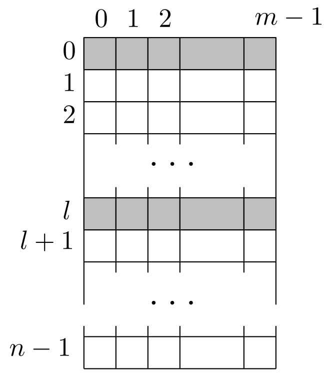

Fig. 1. Values stored in the hash table S.

Online (baby steps). In the online phase we move upward in the grid, and test whether we hit some grey box:

```
for $k = 0, 1, \ldots, l-1$:
  if $(g^{-k} \cdot z \in S)$:
    let $i, j$ be the indices corresponding to the found value
    return $((il + k) \bmod n,\ j)$
```

The correctness of the online phase follows from the fact that each $z \in$ Range(f) can be expressed as $g^{il+k}h^j$, for $0 \le i < \lceil n/l \rceil$, $0 \le j < m$, and $0 \le k < l$; and from the following derivation:

$$g^{-k} \cdot z = g^{il}h^j \qquad \Rightarrow \qquad z = g^{il+k}h^j$$

It can be easily seen that the complexity of this phase as well as the overall complexity of the algorithm is $O(\sqrt{nm})$.

References

1. S. Baba, S. Kotyada, R. Teja: *A Non-Abelian Factorization Problem and an Associated Cryptosystem*, Cryptology ePrint Archive, Report No. 2011/048, 2011. (retrieved: 31/01/2011)