Analysis of the Paper : Low Weight Discrete Logarithm and Subset Sum in $2^{0.65n}$ with Polynomial Memory

R Sai Niranjan [1]

Abstract

Both the Discrete Logarithm and the Subset Sum Problems have seen tremendous applications in the design of cryptographical encryptions.The current study is based on the paper by Easer and May, 2020. The authors of the paper explore the inter-dependencies between the two problems before presenting novel algorithms for the same that show an improved complexity as compared to the approaches known earlier. This paper was presented in Eurocrypt 2020 . Link to the paper : https://eprint.iacr.org/2019/931.pdf The aim of the current study is to understand and explain the findings of the authors in a succinct manner

1. Preliminaries

Before we go further it's essential to be familiar with several notions that shall be used later on. One crucial assumption that is being taken is that the reader is familiar with the basic notions of group theory such as multiplicative and cyclic groups and some basic idea of Markov Chains

1.1. Knapsack Cryptography

The Subset Sum Problem or the Knapsack problem as it is popularly known among Computer Science students(presented almost always as an example of Dynamic Programming) and researchers bears a great importance in the field of cryptography. In fact one of the earliest known public key cryptosystems developed by Merklee and Hellman in 1978 was based on this . The basic idea of the problem goes like this : Given a sum V and n positive integers a_1 ....a_n
Solve the equation :

$$V = \sum_{i=1}^{i=n} a_i x_i$$

where each x_i is 0 or 1 . It is known that this problem is NP Complete which means that while it's easy to verify a solution in polynomial time it's hard to get one from scratch in polynomial time , thus it's usefulness in Cryptography. Of particular interest is the case of super-increasing integers where $a_i < a_{i+1}$ for all i .This case allows for easy decryption and thus can be used to design an appropriate 'trapdoor' allowing for an efficient encryption (note that the sequence has to be converted into a non super increasing case first using modular arithmetic to be of practical use) Issues in implementation force the size of the codes to be large (at least of 200 bits) and the code itself to be a primitive used in more sophisticated implementations . Even so , Knapsack Cryptography is still far too useful.

1.2. BCJ Algorithm

Based on an algorithm for hard instances of the subset sum problem presented in Eurocrypt 2010 , the BCJ Algorithm shows a much improved running time than previous methods used . The basic idea is as follows : Define sequences $\{y_i\}$, $\{z_i\}$ as follows :

$$V = \sum_{i=1}^{i=n} a_i x_i = \sum_{i=1}^{i=n} a_i y_i + \sum_{i=1}^{i=n} a_i z_i$$

The range of each $y_i$and z_i is in the set $\{-1, 0, 1\}$ Thus each x_i can be replaced by a tuple (y_i , z_i) whose sum is precisely x_i.Suppose there are M such possible tuples (M can be easily found using basic combinatorics). Solving the original problem for each sum modulo M is far more easier and thus accelerates the running time . In the BCJ algorithm we consider 8 such sequences at a time thus speeding up the running time tremendously.

1.3. Hamming Weight

In our case , the Hamming Weight simply refers to the number of one's in a binary sequence thus a low hamming weight sequence has a sparse number of 1's in it.Low weight in the paper refers to 'low hamming weight'

1.4. Discrete Logarithm Problem

Suppose G is a cyclic multiplicative group (such as the group consisting of integers modulo p^k , $k > 0$ where p is a prime) $\exists$ an element g (generator) such that all elements can be represented as 'powers'(repetitive performance of

the 'multiplication' operation) of g. In that case similar to the familiar logarithm function , given g and an element x in G , suppose that $g^k = x$ Our task is to find k . The idea of using low weight sequences for k can help in designing of efficient cryptosystems along with keeping p very large when the familiar group of modulo p^k is used.

1.5. Pollard's Rho Algorithm

An efficient algorithm for solving the DLP as defined as above ,it's basic idea is as follows :
Partition the group G into roughly equal sized sets S_0 , S_1 , S_2 . Let x_0 be 1_G and ensure that x_0 is not in S_1. The idea is to compute integers s, t such that $g^s = x^t$
Using this we can easily obtain the solution to the original problem by using properties of Group multiplication. Now coming back to the original algorithm , let n be the cardinality of G and define x_i recursively as :

$$x_i = \begin{cases} g x_{i-1} & x_{i-1} \in S_0 \\ {x_{i-1}}^2 & x_{i-1} \in S_1 \\ x x_{i-1} & x_{i-1} \in S_2 \end{cases}$$

Moreover let x_i be $g^{a_i} x^{b_i}$ Then $a_i = \begin{cases} a_{i-1} + 1 & x_{i-1} \in S_0 \\ 2a_{i-1} & x_{i-1} \in S_1 \\ a_{i-1} & x_{i-1} \in S_2 \end{cases}$

and $b_i = \begin{cases} b_{i-1} & x_{i-1} \in S_0 \\ 2b_{i-1} & x_{i-1} \in S_1 \\ b_{i-1} + 1 & x_{i-1} \in S_2 \end{cases}$

The next idea is to compute an i such that $x_i = x_{2i}$ with $s \equiv a_{2i} - a_i \bmod n$ and $t \equiv b_{2i} - b_i \bmod n$. The tuple (s , t) so obtained are solutions for the problem . The complexity of computing this i is $O(n^{1/2})$

2. Equivalence of Subset Sum and Discrete Logarithm Problems

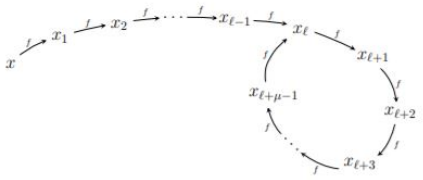
Now that the preliminaries are sorted , let's consider the relationship between the two problems discussed above . Suppose that we have the original DLP as follows $g^\alpha = x$ for the Group G , further let $2^{n-1} \leq |G| < 2^n$ for some $n \in N$. Then it is easy to compute $a_i = g^{2^i}$ for $i \leq n-1$ and $i \in N$. Now here's the crux idea computing the bit representation of α, is precisely equal to the subset sum problem !

3. Improved Low Weight DLP Algorithm

Let G be the same cyclic group as in the previous section , Assume further that the hamming weight of α, is wn , where n is the same as in the previous definition , assume further that $wn \leq \frac{n}{2}$ Borrowing from the BCJ algorithm presented earlier , our idea is to represent $\alpha = \alpha_1 + \alpha_2 \bmod |G|$ with $\alpha_1, \alpha_2 \in Z_{|G|}$

Now for some terminology , consider the Pollard Rho's Algorithm earlier the idea was to find an i such that $x_i = x_{2i}$ We refer to such an occurrence as a 'collision' as our procedure has 'collided' at the same value after some time. Suppose there exists some function f which works on an input x and leads to a collision . Let x_l be the first repeated value and x_{l+u} be it's second appearance, we define := $Rho(f, x) = (x_{l-1}, x_{l+u-1})$ in the spirit of the Rho Algorithm presented earlier and also made clear by the following diagram :

Figure 1. Visualisation of the Rho



This terminology can be extended for colliding 'functions' rather than inputs where $Rho(f_1, f_2, x) = (x_1, x_2)$ where , $f_1(x_1) = f_2(x_2)$

Moving back to our original problem , we need to determine ϕ for which weight ϕn = weight(x_1) = weight(x_2) where x_1 , $x_2 \in Z_{|G|}$ and $weight(x_1 + x_2) = wn$.

To obtain ϕ the authors use Markov Chain Analysis , a technique which has been used sparsely in earlier works. To model $x_1 + x_2$ we need a state comprising of a tuple of three variables (b_1 , b_2 , b_3) where the first element is the 'carry bit' in the operation and the others are the corresponding bits from x_1 and x_2 respectively for example if the two bits were 0 and 0 the state would be (0,0,0) and if they were 1 and 1 the state would be (1,1,1) . Considering that we are sampling uniformly from sequences with weight ϕn , the probability that any bit is 1 is precisely ϕ. Based on this idea , a state transition matrix can be constructed which is as follows :

$$M := \begin{pmatrix} (1-\phi)^2 & \phi(1-\phi) & \phi(1-\phi) & \phi^2 & 0 & 0 & 0 & 0 \\ (1-\phi)^2 & \phi(1-\phi) & \phi(1-\phi) & \phi^2 & 0 & 0 & 0 & 0 \\ (1-\phi)^2 & \phi(1-\phi) & \phi(1-\phi) & \phi^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & (1-\phi)^2 & \phi(1-\phi) & \phi(1-\phi) & \phi^2 \\ (1-\phi)^2 & \phi(1-\phi) & (\phi)(1-\phi) & \phi^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & (1-\phi)^2 & \phi(1-\phi) & \phi(1-\phi) & \phi^2 \\ 0 & 0 & 0 & 0 & (1-\phi)^2 & \phi(1-\phi) & \phi(1-\phi) & \phi^2 \\ 0 & 0 & 0 & 0 & (1-\phi)^2 & \phi(1-\phi) & \phi(1-\phi) & \phi^2 \end{pmatrix}$$

Letting π be a stationary distribution of M , and setting the constraint of summing to w we can solve for ϕ . Finally using this an algorithm can be obtained . Let f be a function $f : \{0,1\}^n \mapsto \{0,1\}^n$ and let P be a family of bijective functions with the same domain and range , now index all the functions of P . We define the 'kth flavour $f^{|k|}$' of f(x) to be P_k(f(x)). Set $\mathcal{T} := \{x \in Z_{2^n} | wt(x) = \phi n\}$ Let π be a hash function such that $\pi : G \mapsto \mathcal{T}$ Define f and f_β as follows :

$f(x) = \pi(g^x)$ and $f_\beta(x) = \pi(\beta g^{-x})$, where $g^\alpha = \beta$, is the DLP

The algorithm is now as follows :

Input: functions $f, f_\beta : \mathcal{T} \to \mathcal{T}$, generator g of G, $\beta \in G$
Output: $\alpha = \text{dlog}_g \beta$ satisfying $g^\alpha = \beta$
1: **repeat**
2: choose random flavour k
3: choose random starting point $s \in \mathcal{T}$
4: $(x, y) \leftarrow \text{Rho}(f^{[k]}, f_\beta^{[k]}, s)$
5: $\alpha' \leftarrow x + y$
6: **until** $g^{\alpha'} = \beta$
7: **return** α'

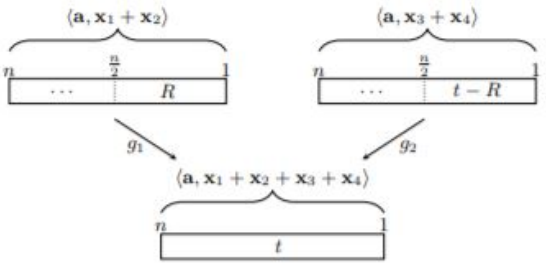
4. Subset Sum in $2^{0.65n}$ with Polynomial Memory

Let's now tackle the second problem we have discussed so far . Worst case scenario for low weight is when the weight is precisely $\frac{n}{2}$. Approaches used for this problem typically involve considering two sequences x_1 and x_2 such that they concatenate to the binary sequence.

In the approach proposed by the authors , the main idea is to use a 'nested collision search' to find solutions to match on the first $\frac{n}{2}$ bits and then use another search for the next $\frac{n}{2}$ bits . Taking the BCJ approach given a solution sequence x we split it into four parts x_1 , x_2 , x_3 , x_4 . Now , the problem is equivalent to solving :
$\sum_{i=1}^{i=\frac{n}{2}} a_i(x_{1,i} + x_{2,i}) = V - \sum_{i=1}^{i=\frac{n}{2}} a_i(x_{3,i} + x_{4,i}) \bmod 2^{\frac{n}{2}}$

The idea then is to fix some $R \in Z_{|2^{\frac{n}{2}}|}$ and then solve for the first two sequences and then solve for the other two with the value V-R.

The figure below illustrates this idea with t used here in the place of V



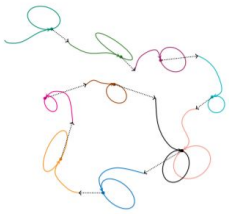
Based on these ideas the algorithm is presented as follows :

Input: subset sum instance $(\mathbf{a}, t) = (a_1, \ldots, a_n, t) \in \mathbb{Z}_{2^n}^{n+1}$
Output: solution $\mathbf{e} \in \{0,1\}^n$ satisfying $\langle \mathbf{a}, \mathbf{e} \rangle = t \bmod 2^n$
1: **repeat**
2: Randomly permute the a_i.
3: Choose $R, z \in \mathbb{Z}_{2^{n/2}}$ randomly.
4: $(s_1, s_2) \leftarrow \text{Rho}(g_1, g_2, z)$
5: Compute $(\mathbf{x}_1, \mathbf{x}_2) = \text{Rho}(f_1, f_{2,R}, s_1)$.
6: Compute $(\mathbf{x}_3, \mathbf{x}_4) = \text{Rho}(f_3, f_{4,t-R}, s_2)$.
7: Set $\mathbf{e} = \mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3 + \mathbf{x}_4$.
8: **until** $\mathbf{e} \in \{0,1\}^n$

Based on an analysis conducted by the authors the algorithm shows a much better time complexity along with Polynomial Memory which makes it pretty advantageous .

Finally a diagram showing the nested rho function :

Figure 2. Different colours represent different flavours in this visualization of the Nested Rho



To conclude, the authors have presented innovative approaches to two standard frameworks that can have wide implications.

References

[1] Esser Andre, May Alexander, **Low Weight Discrete Logarithm and Subset Sum in $2^{0.65n}$ with Polynomial Memory**, *Eurocrypt 2020*

[2] Sussilo Willy , Mu Yi , Seberry Jennifer (eds) , **Information Security and Privacy: 17th Australasian Conference Proceedings** , *ACISP 2012* (Consulted for the BCJ algorithm)

[3] Peikert Chris , Crockket Eris , **Lattices in Cryptography : Lecture 5 Cryptanlysis of Knapsack Cryptography** , https://web.eecs.umich.edu/~cpeikert/lic13/lec05.pdf , *Georgia Tech*

[4] Vaikuntanathan Vinod , Goldwasser Shafi , **Cryptography 6.875 course Video lectures** (available on youtube) , *MIT*