Parallel Implementation of Pohlig-Hellman to Compute Discrete Logarithms

Aditya Sakre
Pimpri Chinchwad College of Engineering, Pune, India

Amit Gurbani
Pimpri Chinchwad College of Engineering, Pune, India

Pooja Dedwal
Pimpri Chinchwad College of Engineering, Pune, India

J. V. Katti
Pimpri Chinchwad College of Engineering, Pune, India

ABSTRACT
Almost half of the world is now connected to the Internet and it has become an integral part of our day to day life. The usage of Internet varies from personal communication to high level business transactions. So security of these Internet services is necessary to maintain Confidentiality, Integrity and Availability. To achieve this there were various cryptographic algorithms proposed. But security of these algorithms needs to be verified. The integer factorization problem, the finite field discrete logarithm problem and the elliptic curve discrete logarithm problem are essential mathematical problems that the practical public key cryptographic systems are based on. ElGamal is one of the cryptographic algorithm, based on discrete logarithms. Pohlig-Hellman algorithm is used for computing discrete logarithms. This paper proposes the parallel implementation of Pohlig-Hellman algorithm to observe improvement in execution time as compared to sequential execution. Paper also analyses the effect of key on execution time.

Keywords
Discrete Logarithm Problem, Pohlig-Hellman, Multithreading.

1. INTRODUCTION
In 1976, Whitfield Diffie and Martin Hellman proposed one of the first public key protocols used for securely exchanging cryptographic keys over a public channel. Taher ElGamal later proposed a public key cryptosystem based on Diffie-Hellman key exchange protocol. This is a public key cryptosystem and a signature scheme. ElGamal cryptosystem relies on discrete logarithms problem which is difficult to solve over finite fields [6]. In many computer systems, users' passwords are stored in a special file, which has the disadvantage that anyone who gets access to that file is able to freely impersonate any legitimate user. Therefore, that file has to be specially protected by the operating system. It has been known for a long time that one can eliminate the need for any secrecy by eliminating the storage of passwords themselves. Instead, one utilizes a function *f* that is hard to invert (i.e., such that given a *y* in the range of *f*, it is hard to find an *x* in the domain of *f* such that *f* (*x*) = *y*) and creates a file containing pairs (*i*, *f* (*pi)*), where *I* denotes a user's login name and *pi* the password of that user. This file can then be made public. The security of this scheme clearly relies on the function *f* being hard to invert.

Discrete Logarithm Problem is based on the concept that it is easy to compute $\beta \equiv \alpha^x \pmod p$ but difficult to find $x \equiv \log_\alpha \beta$ over GF(p) where p is a prime and α is fixed primitive element of GF(p). The security of many cryptosystems relies on the intractability of the discrete logarithm problem. The following is referred to as the DLP or even sometimes as the Generalized DLP.

Attacks on the DLP can be divided into three main categories

1. Algorithms that work in arbitrary groups, such as the exhaustive search and the Baby-Step Giant-Step algorithm,
2. Algorithms that work in arbitrary groups with special conditions present in the group, like Pollard's rho Method, and
3. Algorithms that work only in specific groups, such as the Index Calculus.

2. RELATED WORK

2.1 Index Calculus Algorithm
It is the strongest family of algorithms for finding the discrete logarithms in a cyclic group. The theme is if we can find the discrete logarithms of some small and independent elements, then we should be able to determine logarithms of almost any element in the group, as most elements we can express in terms of the small independent elements whose logs are known [2].

Let p be a prime and let g be primitive root mod p, which means that g is a generator for the cyclic group F_P^X. In other words, every $h \neq (mod\ p)$ can be written in the form $h \equiv g^k$ for some integer k that is uniquely determined mod P-1. Let $K = L(h)$ denote the discrete logarithm of h with respect to g and p, so [2]

$$g^{L(h)} \equiv h\ (mod\ p)$$

Suppose we have h_1 and h 2. Then

$$g^{L(h1h2)} \equiv h1\ h2\ (mod\ P) \equiv g^{L(h1+h2)}\ (mod\ p)$$

which implies that

$$L(h1h2) \equiv L(h1) + L(h2) mod\ (P-1)$$

Therefore L changes multiplication into addition, just like the classical logarithm.

The expected running time of the index calculus is approximately a constant times exp $\sqrt{2lnplnlnp}$, which means that it is a sub exponential algorithm [3].

2.2 Baby step Giant step Algorithm
In group theory, the baby-step giant step is a meet-in-the-middle algorithm computing the discrete logarithm. The discrete log problem is of fundamental importance to the area of public key cryptography. Most commonly used cryptographic systems are based on the difficulty of discrete logarithm problems; the more difficult it is, the more security

it provides a data transfer. One way to increase the difficulty of the discrete log problem is to base the cryptosystem on a larger group. This method was developed by D. Shanks requires √ N steps and √ N storage. The algorithm is based on a space time trade off. This algorithm is a simple modification of trial multiplication method (a naive method of finding discrete logarithms). This attack uses a combination of computational power and memory storage to solve the DLP. Consider G as a cyclic group with generator α. Suppose that α has order n and set $m = [\sqrt{n}]$.

Observe that if $\beta = a^x$, then using the Euclidean algorithm we can write x as follows: x = im + j, where 0 ≤ i, j < m. Thus we have that $\beta = ax = aim + j = aim\, aj$, which implies that $(\alpha - m)i = \alpha j$. For computing the discrete logarithm, begin by computing and storing the values $(j, \alpha j)$ for $0 \leq j \leq m$. Then compute $\beta(\alpha - m)$ and raise that to the exponent i for $0 \leq I \leq m - 1$ and check these values against the stored values of αj to find a match. When a match is found, we have solved the DLP and we have $x = im + j$ as required.

The drawbacks of this algorithm lie in the computation and formulation of the table of pairs $(j, \alpha j)$. At each stage we are required to compute a power of α and look in the table to see if it returns a match. If this is successful then the DLP has been solved. Unfortunately, one has to store around $O(\sqrt{n})$ group elements, perform around $O(\sqrt{n})$ multiplications to find the correct power of α, and in turn perform $O(\sqrt{n})$ table look-ups [1]. As a consequence this algorithm has an expected running time of $O(\sqrt{n})$, which makes it impractical for cryptographic purposes. Procedure:-

1. Fix an integer m> $\sqrt{N}$ and compute mP
2. Make and store a list of iP for 0 ≤i<m
3. Compute the points Q −jmP for j = 0,1…..m-1 until one matches an element from the stored list

4. If iP=Q-jmp, we have$Q = kP$ with $k = i + m\ (mod\ N)$

2.3 Pollards Rho Algorithm

The running time of this algorithm is similar to the Baby-Step Giant-Step method and also requires less memory, which is an advantage. Let G be a cyclic group of order n, where n is prime. G is then partitioned into three subsets of roughly equal size, call these sets S_1, S_2 and S_3. We then define a sequence of group elements, x_i, as follows: $x_0 = 1$ and

$$x_{i+1} = f(x_i) = \begin{cases} x_{i\beta} & if\ x_i \in S_1 \\ x_i^2 & if\ x_i \in S_2 \\ x_i\, a & if\ x_i \in S_3 \end{cases}$$

This in turn defines two sequences of integers a_i and b_i. The sequences a_i and bi are defined as follows: set $a_0 = 0 = b_0$ and for i > 0;

$$a_{i+1} = \begin{cases} a_i & if\ x_i \in S_1 \\ 2a_i\ mod\ n & if\ x_i \in S_2 \\ a_i + 1\ mod\ n & if\ x_i \in S_3 \end{cases}$$

$$b_{i+1} = \begin{cases} b_i + 1\ mod\ n & if\ x_i \in S_1 \\ 2b_i\ mod\ n & if\ x_i \in S_2 \\ b_i & if\ x_i \in S_3 \end{cases}$$

We then begin with a pair (x_1, x_2) and iteratively compute pairs (x_i, x_{2i}) until a pair of group elements such that xi = x_{2i} for some i. [4] When such a pair is found we then have the following relation:

$$a^{a_i}\beta^{b_i} = a^{a_{2i}}\beta^{b_{2i}}$$

$$\beta^{b_i - b_{2i}} = a^{a_i - a_{2i}}$$

$$(b_i - b_{2i}) \log_a \beta = (a_i - a_{2i})\ \ mod\ n$$

$$x = \log_a \beta = (b_i - b_{2i})^{-1}\ (a_i - a_{2i})\ mod\ n$$

2.4 Pohlig Hellman Algorithm

The algorithm was first discovered by Roland Silver, but first published by Stephen Pohlig and Martin Hellman (Independent of Silver). Thus it is sometimes called as Silver – Pohlig Hellman Algorithm. This algorithm is used for computing discrete logarithms in a multiplicative group whose order is a smooth integer.

The Pohlig Hellman method:

P, Q are elements in a group G and we want to find an integer k with Q=kP. We also know the order N of P and we know the prime factorization [5]

N=$\Pi_i q_i^{ei}$

The idea of Pohlig Hellman is to find k (mod q_i^{ei}) for each i, then use the Chinese Remainder theorem to combine these and obtain $k\ (mod\ N)$.

Let q be a prime, and let q^e be the exact power of q dividing N. Write k in its base q expansion as

k =$k_0 + k_1 q + k_2 q^2 + \ldots..$

with $0 \leq ki < q$. We will evaluate $k(mod\ q^e)$ by successively determining k_0 , k_1, k_2 , ... , k_{e-1}.

The procedure is as follows [2]:

1. Compute T = {j ($\frac{N}{q}$.P) 0 ≤ j ≤ q-1

2. Compute $\frac{N}{q}$.Q . This will be an element of k_0 ($\frac{N}{q}$. P) of T.

3. If e = 1, stop. Otherwise, continue.

4. Let $Q_1 = Q - k_0 P$.

5. Compute $\frac{N}{q2}$.Q . This will be an element of $k_1\frac{N}{q}$. P of T

6. If e=2 stop otherwise continue. Suppose we have computed k_0, k_1, ... , k_{r-1}, and Q_1, ... , Q_{r-1}.

7. Let $Q_r = Q_{r-1} - k_{r-1} q^{r-1} P$.

8. Determine k_r such that $\frac{N}{qr+1}.Q_r = K_r$ ($\frac{N}{q}$ P)

9. If r = e -1 , stop. Otherwise, return to step (7).

Then$K \equiv k_0 + k_1 q + k_{e-1} q^{e-1}$ (mod q^e). Therefore we find $k1$. similarly, the method produces k_2 , k_3 ,….. We have to $stop\ after\ r = e - 1$

3. PROPOSED METHOD

Discrete logarithm problem is solved here using Pohlig-Hellman Algorithm. Discrete logarithm problem is to calculate X in :

$$\beta \equiv \alpha^X \text{ (mod P)}.$$

Inputs to the algorithm are α, β and P, where P is a large prime number and X is unknown. Multithreading concept is used for parallel execution of the Pohlig-Hellman algorithm.

Step 1: Inputs for alpha (α), beta (β) and prime number P are accepted.

Step 2: Prime Factors of P-1 are calculated.

Step 3: A thread is created for each prime factor of P-1.

Step 4: Factors are used to compute equations for X

Step 5: Using these equations Pohlig-Hellman algorithm generates modulus equations.

Step 6: Output of Pohlig-Hellman is given to Chinese remainder theorem

Step 7: Chinese remainder theorem gives the value of X, which is the final result.

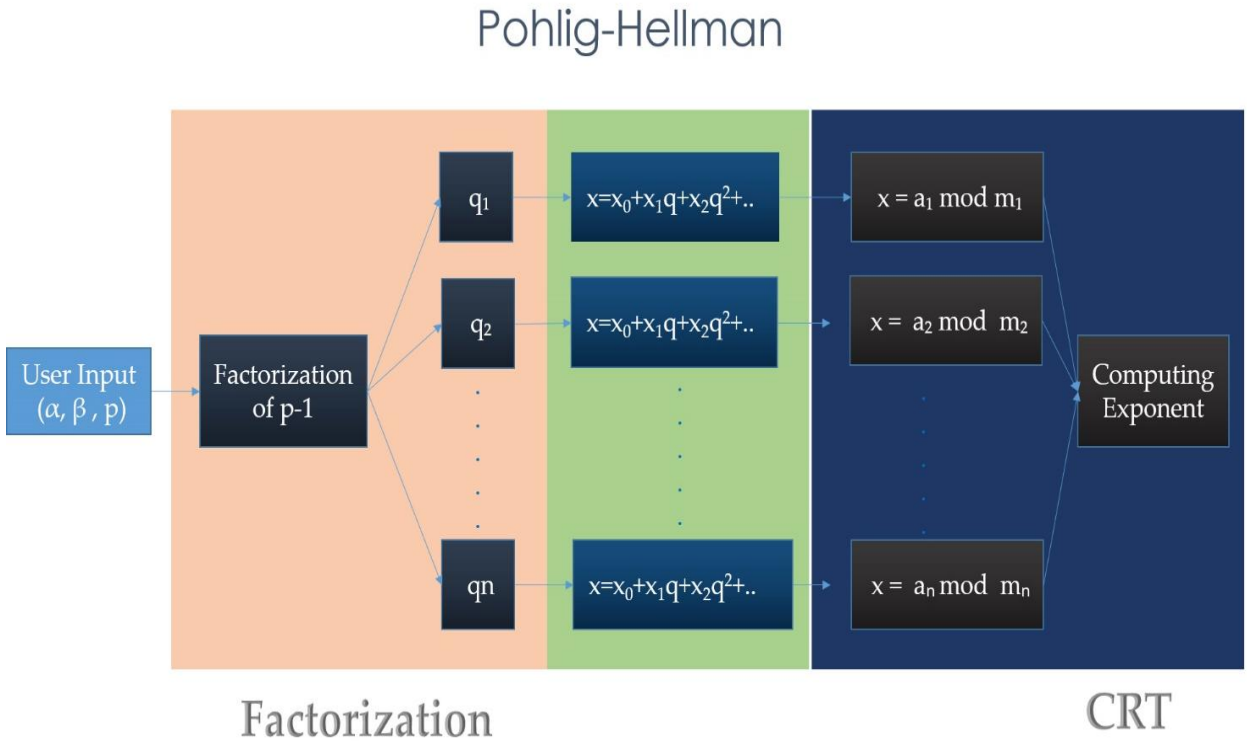

Fig 1: Proposed system and its components

4. METHODOLOGY

The proposed parallel method is implemented using Java programing language. Developed program is executed on x64-based system with 4GB DDR3 RAM and processor - Intel(R) Core(TM) i5-3210M CPU @ 2.50GHz, 2501 MHz, 2 Core(s), 4 Logical Processor(s).

5. RESULTS

For random cases of prime numbers with digits ranging from 10 to 33, the serial and parallel execution time is computed. And the results are depicted in Table 2. The graph of the same is shown in Fig 2. The relationship between execution time of Parallel and Serial and Number of digits in P is also depicted in graph.

Table 1: Output

Prime Number	No of Digits	Alpha	Beta	No. of Prime factors	Serial Exec. Time(sec.)	Parallel Exec. Time(sec.)
2860486313	10	805134798	11796346756	4	0.35542385	0.311159617
501096024853	12	34423486	32170773092	5	49.494820247	47.3468896404
2814112013697373 1333	20	95868702899	2579922701753118 2788	5	47.794943773	44.0839887546
6189700196426901 37449562111	27	8549034111234	1811974120718768 71279130704	11	0.74736394	0.72421763
1622592768292133 6339157801028812	33	509183241453194 1243	3376684179794957 7708534830485778	7	60.626447344	31.456063906

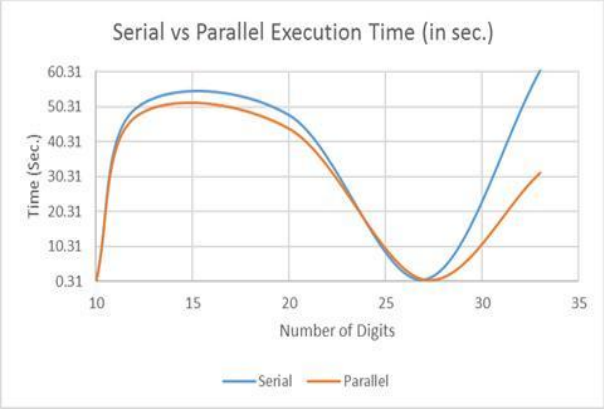

Fig 2: Execution Time graph

6. CONCLUSION

Computing discrete logarithms is basis for many cryptographic protocols. The discrete logarithm problem is considered to be computationally intractable. That is, no efficient classical algorithm is known for computing discrete logarithms in general. More sophisticated algorithms exist, but none of them run in polynomial time. Pohlig-Hellman is one of the algorithm used for computing discrete logarithms. In this paper proposed the parallel implementation of Pohlig-Hellman. After factorization, thread wise computation is carried for each of the factor of one less than prime number (p-1). It is observed that as the number of digits in prime number increases, the execution time of parallel implementation decreases as compared to serial implementation. Expected speedup is n times the serial execution of Pohlig-Hellman, where n is the number of factors of (p-1).

7. REFERENCES

[1] Mrs. Santoshi Pote and Mrs. Jayashree Katti 2015 "Attacks on Elliptic Curve Cryptography Discrete Logarithm Problem (EC-DLP)".

[2] Lawrence C. Washington university of Maryland "Elliptic Curves Number Theory and Cryptography".

[3] Jason S. Howell, "The index calculus algorithm for discrete logarithm", March 31, 1998.

[4] Joppe W.BOS, Alina Dudeanu, Dimitar Jetchev, "Collision bounds for the additive pollard rho algorithm for solving discrete logarithms".

[5] EDLYN TESKE, "The Pohlig-Hellman Method Generalized for Group Structure Computation".

[6] ElGamal T. A public key cryptosystem and a signature scheme based on discret logarithms [J].IEEE Transactions on information Theory,1985,31(4):469-472.