ISSN(Online) : 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 5, Special Issue 9, May 2016

# An Approach to Avoid Weak RSA Key Usage

Shehanaz[1], Raju K[2]

P G Student, Department of CSE, NMAM Institute of Technology, Nitte, Udupi District, Karnataka, India[1]

Associate Professor, Department of CSE, NMAM Institute of Technology, Nitte, Udupi District, Karnataka, India[2]

**ABSTRACT:** Security of RSA algorithm mainly depends on the efforts in factorizing the huge prime numbers which are used in obtaining the RSA modulus. If, we have obtained the collection of keys which are used for encrypting the message and few of them are generated unsuitably, so that encryption keys share the same prime numbers which are known as weak RSA keys. Prime numbers can be decomposed by computing the GCD from the collection of weak RSA keys. The force to find the GCD of large numbers has evoked the research efforts. This paper discusses about how to improve GCD computation time for avoiding usage of weak RSA keys.

**KEYWORDS:** RSA algorithm, GCD, Weak RSA keys.

## I. INTRODUCTION

A Graphics processing units (GPUs) provides high performance processing power for running various applications.GPU is a logic circuit built to render the images and videos. Now GPUs are designed for the computation purpose and can compute the applications which are particularly processed by the CPUs. CPU is a single Chip processor with multiple cores may 4 to 8 cores in latest CPUs, where as in GPU there are hundreds of processing cores. Therefore GPUs have become more popular in the minds of many application developers. CPU and GPU together used in GPU computing model which is nothing but a heterogeneous computing model where serial portion of any application runs on CPU and parallel portion runs on GPU. When there is large data or huge computations which puts heavy load on CPU, using GPU this can be processed. GPUs are manufactured by Nvidia Corporation. This Nvidia Corporation along with GPU productions, it also provides parallel computing capacity to research scholars and science experts which grant them to run huge applications efficiently [1].

RSA is a public key cryptosystem used for safe transfer of data. This cryptosystem uses two types of key: *Encryption key, and a decryption key.* An encryption key has RSA modulus *m* and an exponent *e*, where *m* is the product of two separate prime numbers *pm1* and *pm2* and *e* shouldn't exceed (*pm1*-1) (*pm2*-1). Here, *e* and (*pm1*-1)(*pm2*-1) are co-prime. Now key for decrypting the encrypted data using encryption key has a pair of (*m*,*d*) where *d* is the multiplicative inverse of *e*(mod(*pm1*-1)(*pm2*-1)).This *pm1* and *pm2* can be obtained by decomposing the *m* into *pm1* and *pm2*. But the factorization process is very costly and decomposing the *m* into *pm1* and *pm2* is not possible in experimental computing time. Speed is the main disadvantage of RSA public key cryptography. Since RSA algorithm depends strongly on the factorization of large numbers which affects the performance of this algorithm and put a limit on its use in various application.

Suppose, we have a set of public encryption key which consist of RSA modulus which share or reuse the common prime numbers, such encryption keys are called *weak RSA keys.* In such keys *m* can be decomposed by calculating the GCD. Thereby, we can break the weak RSA keys by calculating the GCD of all pairs of RSA modulus.

The major contribution of this paper explains how GCD computation can be made faster for a pair of very large RSA modulus. Section 2 describes various methods for calculating GCD. Section 3 deals implementation of GCD algorithm in C and CUDA C. Section 4 compares the execution time taken by implemented algorithm in C and CUDA C, which is followed by the conclusion and future work section.

ISSN(Online) : 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

*(An ISO 3297: 2007 Certified Organization)*

**Vol. 5, Special Issue 9, May 2016**

## II. VARIOUS METHODS FOR GCD COMPUTATION

According to number system, finding the two big prime numbers is very easy, but its very difficult to factorize the product of two big integers. There are several algorithms to calculate the GCD of two numbers. Euclidean algorithm (EA) can efficiently calculate the GCD of any two numbers. Since, modulo of large number is costlier with respect to time consumption, Binary Euclidean algorithm (BEA) is used to find the GCD. Since, BEA will not make use of modulo computations and calculate the GCD by the repeatedly subtracting the two numbers and shifting arithmetically till one of the numbers becomes zero. Even though BEA will takes more iterations when compared to EA, each iteration in BEA will takes less time when compared to the iterations in EA. Hence, BEA runs faster when compared to EA and commonly used to calculate the GCD of two numbers.

In [10], Mohamed has given the new algorithm to find the GCD of two numbers. In this he has concentrated on the small and medium size numbers. He has combined both EA and BEA to form this algorithm and give the worst case complexity of this algorithm. To improve the performance of the GCD computation, Mohamed also designed an Parallel algorithm to find the GCD of two numbers based on the reduction method. Because of this reduction algorithm, values of n1 and n2 will be reduced and then GCD can be calculated based on the sequential algorithm.

In [11], David et al. developed an algorithm known as Redundant Binary Euclidean GCD algorithm, to calculate the GCD of any two numbers. They have developed this algorithm based on the SRT division procedure. This SRT division is same as that of non restoring division, but it finds its quotient from a lookup table. Here they have developed the algorithm such a way that it will normalize the integers whose GCD has to be calculated. This normalization can be done based on the three most significant signed bits.

In [12], Chor and Goldreich have developed the parallel algorithm based on the Brent-Kung GCD algorithm. this Brent-Kung GCD(BKGCD) algorithm unlike Euclidean algorithm it performs operation on LSB and check whether the given number is even or odd based on the parity test. This Chor et al. chosen this algorithm as basis for implementing their parallel algorithm, because using this algorithm GCD can be calculated parallel as there is no need to wait for previous operations to be executed, soon we get the LSB of previous iteration, and next iterations can be executed. Chor et al. developed their algorithm by packing the consecutive number of transformation done in BKGCD and executing that transformation in parallel.

In [7], they have implemented the algorithm to find the GCD of RSA moduli to verify that some of the RSA moduli are sharing the same prime key. Here Joseph et al. used the BEA for their work and parallelized the operations of right shift, subtraction and comparison of greater than or equal, so that performance can be improved by performing parallel computation of GCD. But they have memory issues with the GPU, which they have used while implementing their algorithm.

In [8], Miller et al. have worked to parallelize then GCD computation to speed up the performance. They have implemented their algorithm by packing the number of iterations in the Euclidean algorithm. Their final answer as GCD will be very close to the actual GCD but not the exact GCD. They have used the CRCW model so that cost of each bit operation is unit cost. This algorithm uses the pigeon-hole principle which helps in finding the combination of the two integer n1 and n2 which has the lesser bits compared to n. here arithmetic operations performed in parallel in each iteration only once. This algorithm works well only for numbers which takes less time for GCD computations.

All the above methods have one or the other issues. Majority of the methods are proposed for smaller numbers, hence not suitable for large numbers and some are not parallelized efficiently.

Considering all these issues, to address some of them, a new algorithm using which GCD of large number can be calculated has been implemented and compared with the parallelized implementation of the same.

## III. PROPOSED METHOD

All Proposed method of GCD computation uses the approximation algorithm given in [13] to avoid the costlier division operation which is involved in the basic Euclidean algorithm when used to compute GCD of pair of large numbers. The approximation algorithm makes use of only few most significant bits of very large number to perform division operations. For example if we have a number with 1024bits which has to be factorized, here we divide the number of 1024bits to 32words of 32bits and only 2 words of 32bits is considered performing division operation using the proposed algorithm which is an intermediate steps involved in proposed method of GCD computation of pair of

ISSN(Online) : 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 5, Special Issue 9, May 2016

large numbers. The approximation algorithm is as shown below:

```
Approximate (M, N) {
        If (lm≤2)
                return (M div N, 0);
        If (ln=1)
        {
                If (m1≥n1)
                        return (m1 div n1, lm-1);
                Else
                        return (m1m2 div n1, lm-2);
        }
        If (ln=2)
        {
                If (m1m2≥n1n2)
                        return (m1m2 div n1n2,lm-2);
        }
        If (m1m2>n1n2)
                return (m1m2 div (n1n2+1),lm-ln);
        If (lm>ln)
                return (m1m2 div (n1+1),lm-ln-1);
        return (1,0);
    }
```

Here M and N are the two numbers of 1024bits, lm and ln is the number of 32bit words in the M and N, m1, n1, m1m2 and n1n2 are the most significant 32bit word and 64bit word of M and N. This approximation algorithm returns two values, one as quotient and the other as remainder. Using this approximation algorithm, GCD algorithm is designed such a way that it will calculate the GCD of very large numbers of 1024bits.Designed GCD algorithm is as shown below:

```
GCD (M, N) {
        Do {
                (x,y)=approximate (M, N);
                If(y==0) {
                        If(x is even)
                                x=x-1;
                                M=M-(N*x);
                        } M=M-(N*x*pow(d,y))+N;//d is the length of 1 word of input of s-bit
        If (M is even) {
                Do {
                        Right-shift (M);
                    } while (m is even)
                     If (M<N) Swap (M, N);
                    }
                } while (N!=0); Return M;
    }
```

To improve the performance of the above GCD algorithm, it is implemented using CUDA C. CUDA C implementation is done as shown below:

```
Initialize i=0
If(threadIdx.x<num_thread)
{
        If ((blockIdx.x*blockDim.x+threadIdx.x)<(no. of RSA modulus – 1))
```

ISSN(Online) : 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 5, Special Issue 9, May 2016

```
{
        Do
        {
                If((i>(blockIdx.x*blockDim.x+threadIdx.x))&& (i<no. of RSA modulus))
                {
                        M=key [blockIdx.x*blockDim.x+threadIdx.x];
                        N=key [i];
                        M1=GCD (M, N);
                }
                Final_GCD[(blockIdx.x*blockDim.x+threadIdx.x)*no. of RSA modulus + i]= M1;
                i++;
        }while (i<gridDim.x*blockDim.x);
}
}
```

## IV. RESULTS AND DISCUSSIONS

Designed GCD algorithm has been implemented using C and also using CUDA C and basic Euclidean algorithm is implemented in C for the comparison with designed GCD algorithm in C and CUDA C. Figure 1 shows the graphical representation of the time taken for execution of both versions of GCD computation and Basic Euclidean algorithm. We can expect huge speed up using this method compared to basic Euclidean algorithm for computing GCD when used for very large numbers. An advantage of this method is that unlike the basic algorithms it can perform the operation for 1024bits of number.

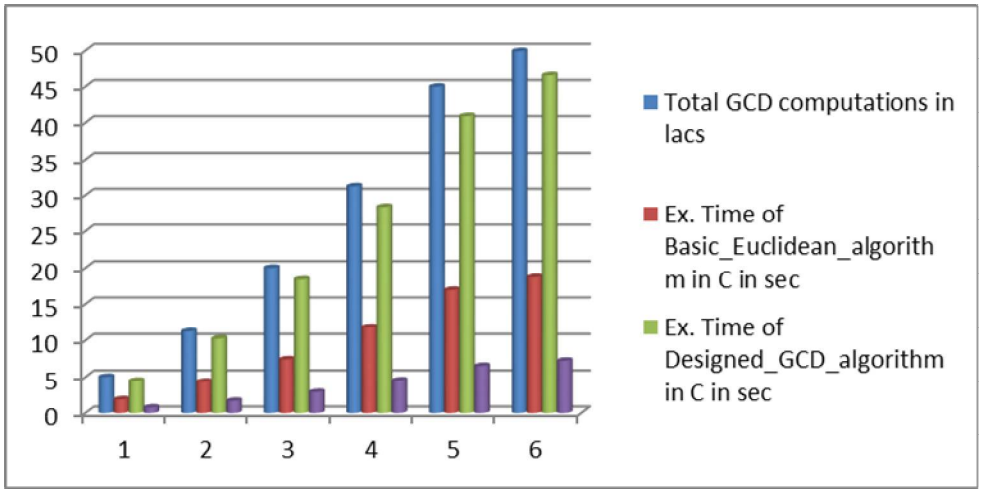


**Figure 1: Graphical Representation: Execution time of implemented algorithms.**

## V. CONCLUSION AND FUTURE WORK

RSA algorithms face the difficulty in factorizing the RSA modulus in the Encryption key. Factorizing of RSA modulus can be done using GCD computation. Hence to speed up this GCD Computation and to avoid the costlier Division operation, approximation algorithm has been implemented. A new GCD algorithm has been implemented in C and to parallelize this GCD computation to speed up the performance CUDA C has been used, which is 6 times faster than the C implementation and 2.5 times faster than the basic Euclidean algorithm. As a part of future work, identification of weak RSA keys based on the GCD output has to be implemented.

ISSN(Online) : 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

*(An ISO 3297: 2007 Certified Organization)*

**Vol. 5, Special Issue 9, May 2016**

## REFERENCES

[1] https://en.wikipedia.org/wiki/Nvidia
[2] https://en.wikipedia.org/wiki/CUDA
[3] JayshreeGhorpade, JitendraParande, MadhuraKulkarni,andAmitBawaskar "GPGPU processing in CUDA Architecture,"Advanced Computing: An International Journal ( ACIJ ), Vol.3, No.1, January 2012.
[4] NVIDIA Corporation, "NVIDIA CUDA C programming guide version 5.0," 2012.
[5] Ambedkar, B. R., and S. S. Bedi. "A New Factorization Method to Factorize RSA Public Key Encryption." IJCSI International Journal of Computer Science Issues 8.6 (2011).
[6] N.Fujimoto,"High Throughput Multiple-Precision GCD on the CUDA Architecture", IEEE International Symposium on Signal Processing and Information Technology (ISSPIT), IEEE, 507512, 2009.
[7] K. Scharfglass, D. Weng, J. White, and C. Lupo, "Breaking weak 1024bit RSA keys with CUDA," in Proc. of Internatinal Conference of Breaking weak 1024-bit RSA keys with CUDA, Dec. 2012, pp. 207 – 212.
[8] Kannan, Ravindran, Gary Miller, and Larry Rudolph. "Sublinear parallel algorithm for computing the greatest common divisor of two integers." SIAM Journal on Computing 16.1 (1987): 7-16.
[9] https://en.wikipedia.org/wiki/Binary_GCD_algorithm
[10] Sedjelmaci, Sidi Mohamed. "The mixed binary euclid algorithm." Electronic Notes in Discrete Mathematics 35 (2009): 169-176.
[11] Parikh, Shrikant N., and David W. Matula. "A redundant binary Euclidean GCD algorithm." Computer Arithmetic, 1991. Proceedings, 10th IEEE Symposium on. IEEE, 1991.
[12] Chor, Benny, and OdedGoldreich. "An improved parallel algorithm for integer GCD." Algorithmica 5.1-4 (1990): 1-10.
[13] Fujita, Toru, Koji Nakano, and Yasuaki Ito. "Bulk GCD Computation Using a GPU to Break Weak RSA Keys." Parallel and Distributed Processing Symposium Workshop (IPDPSW), 2015 IEEE International. IEEE, 2015.