International Journal of Applied Engineering Research
ISSN 0973-4562 Volume 10, Number 3 (2015) pp. 7757-7764

http://www.ripublication.com

Parallel RSA-Based Digital Signature Algorithm For Digital Signing Applications

Sapna Saxena
Chitkara University, Himachal Pradesh, India

Bhanu Kapoor
Chitkara University, Himachal Pradesh, India

Abstract

A Digital Signature is a mean of providing security, authenticity and integrity to the electronic information shared or transferred during the crucial applications running on Internet such as Banking, Trading, and Money Transactions in e-commerce, etc. Popularly two algorithms are used for generating the digital signatures for electronic documents – RSA and DSA. The Digital Signature algorithms are the combination of two important techniques – Signing and Verification of Signature. In the RSA Digital signature algorithm, the signature generation and signature verification is solely based on the modular exponentiation and modular reduction. The larger key sizes are taken in RSA Digital signature, typically 2048 bits, to ensure the security of the algorithm. Because of the involvement of larger key sizes the algorithm becomes compute-intensive and takes lot of time and energy to execute. Moreover in the era of mobile devices with limited power backup there has to be some technique that can increase the pace of Signing / Verification. One of the most popular techniques is Parallel Programming that can be used to redesign the existing digital signature algorithm at the same time maintaining its security intact. In this paper, we are proposing the redesigned Parallel RSA Digital Signature algorithm. The parallel algorithm is tested using OpenMP API on GCC infrastructure and results shows the speedup of approximately 5X as compared to its sequential counterpart.

Keywords Digital Signature, RSA Algorithm, SHA-1, OpenMP, GCC Infrastructure, Serial Implementation, Parallel Implementation.

Introduction

In the modern era, Internet is becoming the integral part of the business at very fast pace. All type of e-commerce applications such as Banking, Trading, Stock Marketing,

online sale-purchase etc need to share electronic documents among different parties involved in business. Involvement of significant amount of online money transaction is even more complicating the scenario. Traditionally, the physical paper documents are authenticated by the means of signature. It is difficult to tamper, forge or alter the signature once it is done on the paper document. In addition to this after doing the signature the signer cannot deny the signing later. Similarly, the electronic documents can also be authenticated by using digital signatures where tampering with or forging the digital signature is difficult.

A Digital signature [1] is based on mathematical calculations used for the representation of the integrity and authenticity of a message meant for the transmission on network. Once the message is digitally signed, the sender cannot deny that he has sent the message to the recipient. In other words, a valid digital signature gives a recipient reason to believe that the message was created by a known sender. Digital signatures are used at many places such as in the money transactions through the Internet, in distributing software, and to detect tampering forgery. Digital Signature helps to ensure following security features –

- **Authentication:** The message is the one that was originally sent by the sender.
- **Non-Repudiation:** If the sender has sent a digitally signed message then he cannot deny that he or she is the original sender of the message.
- **Integrity:** The message is the original one and nobody has tampered with or altered it during the transmission.

Digital Signature Algorithms

There are number of algorithms based on public key cryptography and hashing which are used in combination to generate digital signatures. The most popular public-key algorithms are the RSA and the Digital Signature Algorithm (DSA) algorithms.

Rivest Shamir Adleman Algorithm

The RSA digital signature algorithm [2, 3] was invented by the three cryptographers at MIT in 1977 namely - Ron Rivest, Adi Shamir, and Leonard Adleman. RSA is a cryptographic algorithm based on the modular exponentiation and modular reduction. RSA does not define any specific hashing algorithm to generate the hash code (MAC) for the message, and therefore the security of the digital signature and the encryption also depends upon the hash function being used to generate the digital signature. Usually SHA-1 [5, 6] is used to generate the hash code with RSA.

Digital Signature Algorithm

The DSA (Digital Signature Algorithm) [4] was developed by the National Institute of Standards and Technology (NIST) in 1991 and it is defined by the Digital Security Standard (DSS). The algorithm uses SHA-1 hashing algorithm to generate the hash code (MAC) to compute its digital signature. The DSA algorithm does not sign the hash code after computing the digital signature rather it merely generates a signature that can be used to prove the authenticity and integrity of the message. DSA signatures can be generated as quickly as RSA signatures, but their verification can take much longer because of the complexity of the algorithm.

Parallel Rsa Based Digital Signature Algorithm

The parallel RSA digital signature algorithm is the combination of two algorithms, first is SHA-1 which is used to generate the MAC for the message and the second is RSA algorithm which is used to generate the pair of private and public key, signature generation and signature verification. The method that is used to parallelize the proposed RSA digital signature algorithm is the repeated square-and-multiply method [7].

The proposed RSA digital signature algorithm is based on the repeated square-and-multiply modular exponentiation [7, 8] algorithm is based on the mathematical equation that for an even exp,

$base^{exp}$ mod modulus = ($base^{exp/2}$ * $base^{exp/2}$) mod modulus

The parallel distribution of data among the multiple cores is performed by the data distribution technique and static mapping is used to map the tasks among the available cores. The model which is followed in redesigning the parallel algorithm is Shared Multiprocessor Architecture. The private thread dependent variables are allocated to local cache memory of the cores. Each thread is assigned with their personal copies of the private variables. The global variables are allocated to shared memory, which are equally shared among the threads.

The proposed RSA digital signature algorithm is scalable thus it is capable of adapting itself according to any multi-core machine either quad-core or dual quad-core and so on. The proposed algorithm is given in Figure 4.

Table 1: Parallel RSA Based Digital Signature Algorithm

Step 1. Declare Vector Str
Step 2. Read Message from File to Vector Str
Step 3. Apply SHA-1 to Vector Str and Generate Message Digest HashCode
Step 4. Declare Result=1
Step 5. Read the Number_Of_Cores available on Target_Machine
Step 6. Declare N = Power / Number_Of_Cores
Step 7. Declare Chunk_Size = N / Number_Of_Cores
Step 8. Decompose the whole modular exponentiation and reduction into 'N' Parallel Sections
Step 9. Assign No_Of_Iterations equal to Chunk_Size to each Thread of the each Parallel Section
Step 10. For each Section execute "Step 11" PARALLELY
Step 11. IF Power mod Number_of_Cores == 0
FOR I = 1 to N
Result = Result * HashCode
END FOR
ELSE
FOR I = 1 to N
Result = Result * HashCode
END FOR
Result = Result * HashCode

```
END IF
        Final Signature = Result MOD Modulus
```

The Parallel RSA Based Digital Signature algorithm is divided into two parts:

Digital Signing

In order to sign a message the sender performs the following steps:

1. Generates the hash code (MAC) of the message using SHA-1 algorithm.
2. Uses his/her private key (d, n) to compute the signature:
   $S = M^d \bmod n$
   where S represents signature, M is the message and (d, n) is the private key.
3. Appends the Signature to the message.
4. Sends the message along with the signature to recipient

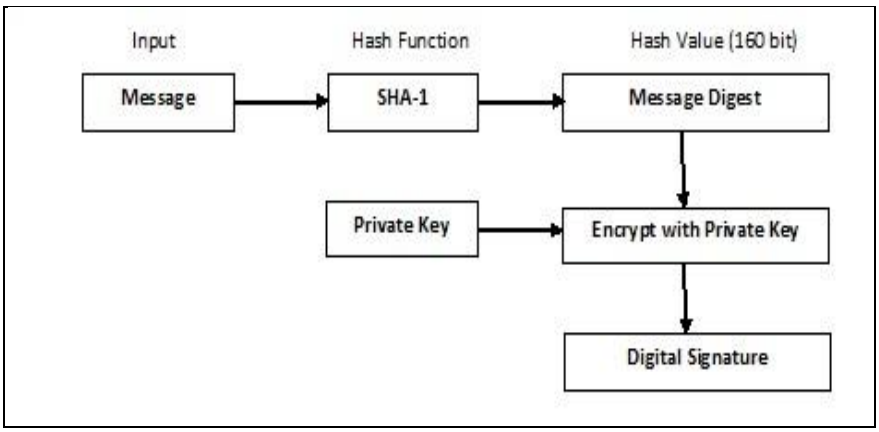

Figure 1: Digital Signing Process

Digital Signature Verification

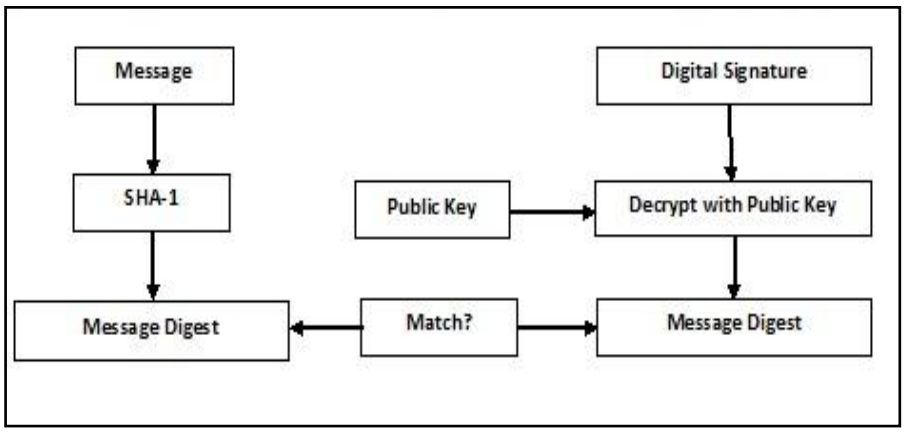

Figure 2: Digital Signature Verification Process

The recipient does the following in order to verify the message:

1. Removes the Signature from the message.
2. Uses the sender's public key (n, e) to compute the hash value $V = S^e \bmod n$
3. Generates the hash code from the message once again.
4. If both the hash codes are identical then the signature is valid otherwise it is discarded due to its invalidity.

The proposed parallel RSA Digital signature algorithm is implemented with the OpenMP API. OpenMP API [9] is a portable parallel programming model based on shared memory architecture. OpenMP API is comprised of compiler directives and library routines specially designed to achieve massive parallel capabilities.

For the experiments, we have used various data sets to test the algorithm. The algorithm is tested for various key sizes varying from the 512 bits to 2048 bits. The same set of message with size 5 MB characters is used for every key size to test the strength of the algorithm.

The performance gained during the execution of RSA digital signature algorithm is measured in terms of time. To measure the time conserved during the execution of algorithm "time" utility of Linux is used. The time utility measures the elapsed time in milliseconds. To record the time taken during the various segments of the algorithm, it is divided into two parts: the signature generation comprised of key generation and the signing part and the signature verification part. The results for each data set are displayed in the Table 1. To record the time during the execution, the algorithm is executed for exactly 25 times for each data set and the average of the results is taken as the final time.

In order to demonstrate the speeding up of the algorithm, the comparative analysis is done with respect to the time taken during the serial and parallel implementation of Parallel RSA digital signature algorithm.

Performance Analysis

The Parallel RSA-Digital signature is developed to improve the performance of RSA based digital signature on multi-core machines. After executing the parallel algorithm on an 8-core machine, we see promising results in terms of time saved during execution. After getting the final results, it is obvious that fast implementation of RSA Signing/Verification is possible on multi-core machines if implemented in a parallel fashion.

We have succeeded to obtain the improvements in the time taken during the execution of Parallel RSA Digital signature algorithm, as presented in the Table 1. The Table 1 shows the execution time and performance comparison between the serial and parallel implementation of Proposed RSA Digital signature algorithm. The results shown in the table are the comparative results with respect to the fixed message size of 50,000 characters but varied key sizes i.e. from 512 bit to larger key size 2048 bit. It is apparent in the results shown that as and when we increase the key size of the algorithm it gives the better result as compared to serial algorithm. The faster implementation can be used for power savings by running the algorithms at slower speeds and at lower voltages.

Fig 3 shows the graphical representation of the results shown in Table 1. It can be seen in the results shown in Table 1 and Figure 4 that when the proposed RSA digital signature algorithm is executed on dual quad core, it gives approximately 5x or more speedup with respect to its execution on single core.

Table 2: Time taken during the serial/parallel implementation of various program segments

Key Size (in Bits)	**Program Segment type**	**Time taken w.r.t. the serial and parallel execution**		**Speedup obtained**
		Sequential Version running on Single core	**Parallel Version running on Eight Cores**	
512	Signature Generation	0.0131	0.00422	3.1
	Signature Verification	0.00079	0.00051	1.55
1024	Signature Generation	0.07755	0.0205	3.78
	Signature Verification	0.00171	0.00071	2.41
1280	Signature Generation	0.13766	0.0325	4.24
	Signature Verification	0.00228	0.00083	2.75
1536	Signature Generation	0.27458	0.05707	4.81
	Signature Verification	0.00361	0.00105	3.44
1792	Signature Generation	0.43585	0.08813	4.95
	Signature Verification	0.00522	0.0013	4.02
2048	Signature Generation	0.67535	0.13167	5.13
	Signature Verification	0.00606	0.00146	4.15

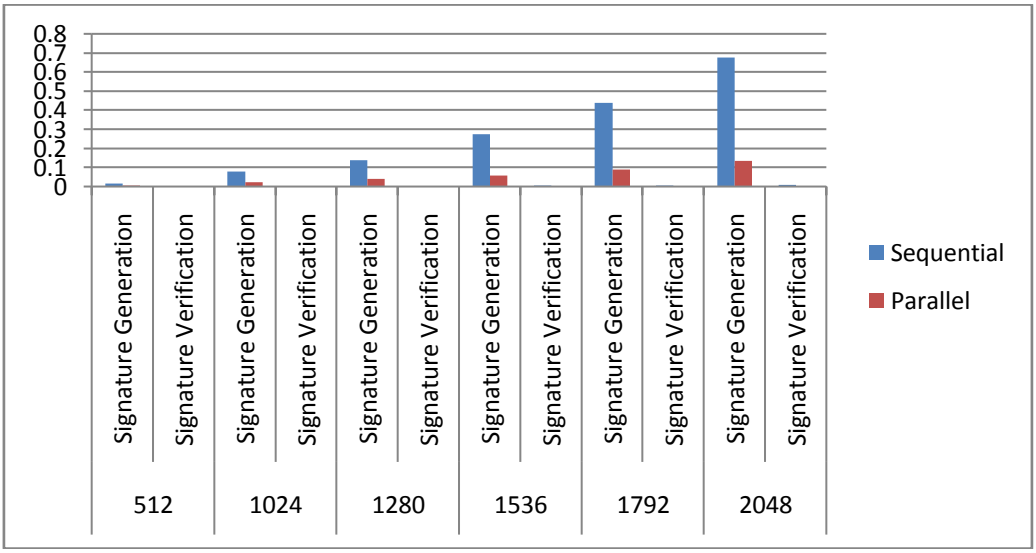

Fig 3: Graphical Representation of Time taken during the serial/parallel implementation of various program segments

Thus, a faster version of the RSA Digital Signature algorithm can be implemented on multi-core machines if implemented parallel. OpenMP in combination with GCC infrastructure allows implementing Parallel RSA Digital Signature algorithm that decreases the execution time and improves the performance in terms of time and energy.

Conclusion and Futurescope

The experimental results show that the parallel RSA based Digital Signature algorithm gives the promising results if implemented using the OpenMP API in the GCC environment. The parallel RSA based digital signature algorithm is more time efficient as well as energy efficient than its sequential counterpart.

The experiments are performed under dual quad core environment and are based on repeated square-and-multiply method. These experiments could be further performed using other modular exponentiation methods. They could be further improved upon the synchronization issues to improve the runtime. Moreover, the experiments performed to obtain the parallel RSA bases Digital Signature algorithm is based on the modular exponentiation and modular reduction part of the RSA. The further experiments of the research will be focused on the factorization part of the RSA key generation algorithm.

References

[1] D. Poulakis. A variant of Digital Signature Algorithm. Designs, Codes and Cryptography, 51(1):99-104, 2009.

[2] Menezes, A. J., Vanstone, S. A., & Oorschot, P. C. V. 1996. Handbook of Applied Cryptography. CRC Press, Inc., Boca Raton, FL, USA.

[3] Fu, C. & Zhu, Z.-L. Oct. 2008. An efficient implementation of RSA digital signature algorithm. In Wireless Communications, Networking and Mobile Computing, 2008. WiCOM '08. 4th International Conference on, 1–4.

[4] Diffie, W. & Hellman, M. Nov 1976. New directions in cryptography. Information Theory, IEEE Transactions on, 22(6), 644–654.

[5] Rivest, R. L., Shamir, A., & Adleman, L. 1978. A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM, 26(1), 96–99.

[6] R.L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM, 21(2):120-126, 1978.

[7] Bewick, G. 1994. Fast multiplication algorithms and implementation.

[8] Igor L. Markov, Mehdi Saeedi, "Constant-Optimized Quantum Circuits for Modular Multiplication and Exponentiation", Quantum Information and Computation, Vol. 12, No. 5&6, pp. 0361-0394, 2012

[9] R. Chandra, R. Menon, L. Dagum, D. Kohr, D. Maydan, J. McDonald, Parallel Programming in OpenMP. Morgan Kaufmann, 2000.

[10] Josef Pieprzyk1 and David Pointcheval, Parallel Authentication and Public key encryption, Springer-Verlag 2003

[11] Barrett, P. 1986. Implementing the rivest, shamir and adleman public-key encryption algorithm on standard digital signal processor. Proceedings of CRYPTO'86, Lecture Notes in Computer Science, 311–323.

[12] Diego Viot, Rodolfo Aurelio, Helano Castro and Jardel Silveria, Modular Multiplication Algorithm for PKC, Universiadade Federal do Ceard, LESC

[13] Cohen, H., Frey, G. (editors): Handbook of elliptic and hyperelliptic curve cryptography. Discrete Math.Appl., Chapman & Hall/CRC (2006)