International Journal of Cryptology Research 1(2): 235-243 (2009)

Improving Security Performance with Parallel Crypto Operations in SSL Bulk Data Transfer

[1]Hashem Mohammed Alaidaros, [2]Mohamed Othman and [1]Mohd Fadlee A. Rasid

[1]Department of Computer and Communication Systems Engineering, Faculty of Engineering, Universiti Putra Malaysia

[2]Department of Communication Technology and Network, Faculty of Computer Science and Information Technology, Universiti Putra Malaysia

E-mail: aidaros.dev@gmail.com; mothman@fsktm.upm.edu.my

ABSTRACT

Information security, including integrity and privacy, is an important concern among today's computer users due to increased connectivity. Despite a number of secure algorithms that have been proposed, the trade-offs made between security and performance demands further research toward improvement. For example, in bulk data transfer, especially in large messages, the secured processing time takes much longer than non-secured processes. This is due to crypto operations, which include symmetric encryption operations and hashing functions. In the current bulk data transfer phase in Secure Socket Layer (SSL), the server or the client firstly calculates the Message Authentication Code (MAC) of the data using HMAC operation, and then performs the symmetric encryption on the data together with the MAC. This paper proposes a new algorithm which provides a significant performance gain in bulk data transfer without compromising the security. The proposed algorithm performs the encryption of the data and the calculation of the MAC in parallel. The server calculates the MAC of the data at the same time as the encryption process of the data. Once the calculation of the MAC is completed, only then the MAC will be encrypted. The algorithm was simulated in two processors with one processor performing the MAC calculation and the other on encrypting the data, simultaneously. The communication between the two processors was done via Message Passing Interface (MPI). Based on the performance simulations, the new parallel algorithm gained speedup of 1.74 with 85% efficiency over the sequential (current) algorithm.

Keywords: Information security, bulk data transfer, parallel crypto operations, hashing, encryption.

INTRODUCTION

Cryptography has become an essential component of modern information systems providing the mechanisms necessary to provide integrity, accuracy and confidentiality in public communication mediums such as the Internet. Information security including integrity and privacy is an important concern due to increasing connectivity among today's computers. Despite a number of secure algorithms that have been proposed, the trade-offs made between security and performance demands further research toward improvement (Zhao et al., 2005; Elgohary et al. 2006; Dongara and Vijaykumar, 2003; Hodjat and Verbauwhede, 2004). For example, in bulk data transfer, the secured processing time takes much longer than non-secured processes. This is due to crypto operations, which include symmetric encryption operations and hashing functions.

This paper focuses on bulk data transfer stage which deals with large amounts of data. While private-key and hashing are computationally intensive in large amounts of data (although not as much as in public-key), the main reason for high consumption is lack of parallelism. Modern computers exploit parallelism to achieve performance, and the lack of parallelism obstructs achieving fast response times. This paper proposes a new algorithm which provides a significant performance gain in bulk data transfer without compromising the security. In the current bulk data transfer phase in Secure Socket Layer (SSL), the server or the client first calculates the Message Authentication Code (MAC) of the data using HMAC operation, and then performs the symmetric encryption on the data together with the MAC.

The proposed algorithm performs the encryption of the data and the calculation of the MAC in parallel, while preserving the same security level. The server calculates the MAC of the data at the same time as the encryption process of the data. Once the calculation of the MAC is completed, only then the MAC will be encrypted. The algorithm was simulated in two processors with one processor performing the MAC calculation and the other encrypting the data, simultaneously. The communication between the two processors was done via Message Passing Interface (MPI). The following section presents an overview of the SSL privacy and integrity in bulk data transfer which are provided by private-key and HMAC respectively,. The parallel crypto operations are presented in section 3. Results discussion and conclusions are given in section 4 and 5 respectively.

CRYPTOGRAPHY OVERVIEW

Private Key Encryption

The Rijndael Algorithm was chosen by the US National Institute of Standards and Technology (NIST) as the new Advanced Encryption Standard (AES). Its security is based on the secrecy of the key and is primarily used for bulk data encryption. For the block cipher encryption, one of the most popular modes is chaining-block-cipher (CBC) mode. In this mode, the plain text is XORed with the previous cipher block before it is encrypted. This ensures dependency between blocks of data within the message and removes the potential for parallelism across individual blocks of data (Stallings, 2003).

Padding

AES is a block cipher; it works on a unit of fixed size, meaning that it divides the data into blocks of 16 bytes. In figure 1 (a) before encryption, if the message length n is not a multiple of 16 Bytes, then padding is needed. It will append zeros to the end of data array until the data length is a multiple of 16 bytes (Stallings, 2003).

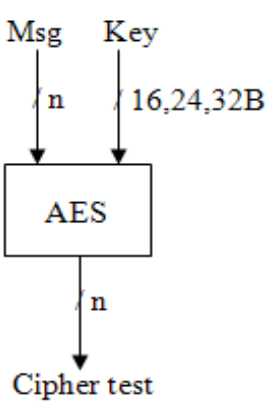

Figure 1(a): AES

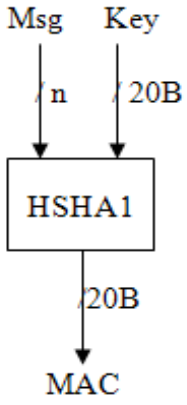

Figure 1(b): HMAC_SHA1

HMAC

Hash functions map an arbitrary long message to a fixed-size hash value. In 2002 the NIST Secure Hash Standard proposed the use of Secure Hash Algorithm (SHA1). Message authentication is achieved via the construction of a message authentication code (MAC). MACs based on cryptographic

hash functions, such as SHA1, are known as Keyed-Hash Message Authentication Code (HMAC), proposed by FIPS (FIPS 2002). The message authentication algorithm is called HMAC, while the result of applying HMAC is called the MAC. The purpose of a MAC is to authenticate both the source of a message and its integrity.

HMAC with its underlying SHA1 are called HMAC-SHA1 or HSHA1. As in Figure 1(b) HSHA1 has two functionally distinct parameters, (1) a message input of length n and (2) a secret key of length 20 bytes known only to the message originator and intended receiver(s), it produces 20 bytes as output. HMAC-SHA1 is faster than AES.

Secure Sockets Layer (SSL) Protocol and Bulk Data Transfer Load

SSL is one of the most widely used security protocols on the Internet. It is implemented at the transport layer of the protocol stack. SSL offers the basic security services of encryption, source authentication, and integrity protection for data exchanged over underlying unprotected networks. Details on SSL protocol are available elsewhere (Rescola, 2000).

The two phases of an SSL connection, handshake and data transfer are worth considering separately. Handshake happens only once per connection, but is comparatively expensive. Each individual data record is comparatively cheap, but for connections involving large amounts of data the cost of the data transfer phase eventually dominates the cost of the handshake. Nevertheless, the cost of data transfer is mostly due to cryptography (Rescola, 2000).

The portion of the private key encryption and hashing are increasing as we increase the request file size. However, as this part is the main contributor to the bulk data transfer phase, it can become significant at very large file size, over 32KB, since the private key and hashing are ones of the main operations in the SSL bulk data transfer.

PARALLEL MODEL IN CRYPTO OPERATIONS

Existing Sequential Model

In current bulk data transfer phase in SSL, the message is broken into fragments F up to 16384 bytes, each fragment will therefore be treated separately. The server or the client firstly calculates the Message

Authentication Code (MAC) of the fragment using HMAC operation followed by padding if needed, and then performs the symmetric encryption E on the fragment together with the MAC to produce the cipher text as shown in Figure 2(a). The sequential algorithm is simulated on one processor P0. Compression operation is not considered since it is optional.

Proposed Parallel Model

Our model proposes a new parallel security algorithm that provides integrity by HMAC-SHA1 and privacy by AES, without compromising security. As shown in Figure 2(b) the proposed parallel algorithm performs the encryption of the fragment and the calculation of the MAC in parallel, preserving the same security level. The algorithm was simulated in two processors, P0 and P1, with one processor performing fragment encryption and the other a MAC calculation, simultaneously. The server calculates the MAC of the fragment at the same time as the encryption process of the fragment. Once the calculation of the MAC and padding are completed, it will be sent to P0. Once P0 receives it from P1, only then the padded MAC will be encrypted.

Simulation Platform

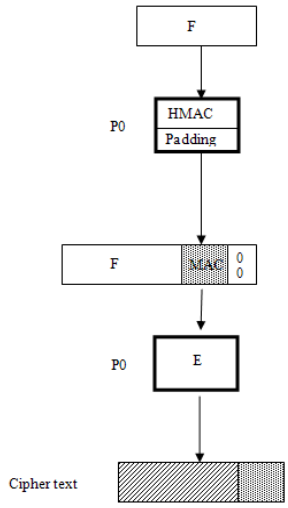

Figure 2 (a): Sequential algorithm

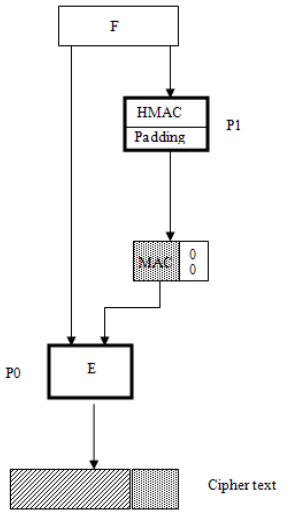

Figure 2(b): Proposed Parallel algorithm

Figure 2 (a) and (b) has been simulated and produced the same output without interference. However, relative comparison of these benchmarks is more meaningful: each implementation is in the same OS platform, programming language, API (Application Programming Interface), coding style, and compiler. The slave machine we use is under Linux OS with two Intel® Xeon processors running at 2.3GHz and with 774 MB of shared memory. The slave machine is clustered with the master machine. The master machine specifications (performance, speed etc) are the same as the slave machine, but uses only a single processor.

We use an AES with 128 of key size in CBC mode as an encryption algorithm and SHA1 as the underlying hash algorithm in HMAC, so the overall operation can be written as AES128-CBC-SHA1. Note that AES is always slower than HMAC-SHA1, although the latter uses hashing twice. The machine is installed with a Personal Version of CryptoSys API as cryptography library in C language (CryptoSys, 2006).

Implementation in MPI

MPI stands for "Message Passing Interface". It is a library of functions (in C) that are inserted into source code to perform data communication between processors (Quinn, 2003). Our parallel version has been developed starting from a sequential version in C that has been extended through a set of procedures written in C plus MPI primitives. MPICH version 1.2.7p1 is installed in our machine as MPI library and compiler (MPICH, 2005). The development from the sequential to parallel version has shown reduction in the processing time.

PERFORMANCE RESULTS

Speedup Over Sequential

We will first present that our parallel approach achieves good speedup over sequential approach (uniprocessor). In Fig 3, we show the execution time in milliseconds spent by parallel and sequential processes varying the size of messages from 16KB up to 1MB. Since AES and HSHA1 increase as message size increases, we observed that the sequential and parallel execution times increased as message size increased in the same ratio.

As is evident from the chart (Figure 3), we achieved good speedup by parallelizing the cipher operations because the parallel execution time is less than the sequential time, by a significant ratio.

For example, in 256K bytes file size, when we use the sequential process it costs 13 ms, but when we use our proposed process it costs only 7.4 ms. Speedup is the ratio between sequential execution time and parallel execution time (Quinn, 2003). The efficiency of a parallel program is a measure of processor utilization. We define efficiency to be speedup divided by the number of processors used (two processors in our case) (Quinn,2003).

$$\text{Efficiency} = \frac{sequential\ \ execution\ \ time}{processors\ \ used \times parallel\ \ execution\ \ time}$$

Since the AES128-CBC and HMAC-SHA1 increase at the same ratio (linearly) when message size increases, the message size does not affect the speed up. We achieved speedup of about 1.74 and efficiency of 87%. Parallel overheads obstruct efficiency in order to reach the maximum as we will see in the next section.

Parallel Overhead

The next result is the discussion on the execution time breakdown in parallel overheads which obstruct the maximum achievable speedup. Figure 4 shows the parallel overhead analysis in our parallel code simulation. All overheads increase linearly as message size increases since they are called as many times as the number of fragments. Figure 4 shows the parallel overheads execution time in microseconds. It can be seen that the barrier synchronization time takes a very large portion of about 65% of the parallel overheads. This is obvious since the HMAC and padding are faster than AES, so it delays the P1 until P0 processes the fragment. It also shows that the portion of send and receive overheads take almost the same amount of time about, of 15% each. Send overhead does not depend on crypto operation speeds, and receive overhead is not obstructed since the HMAC is faster than AES.

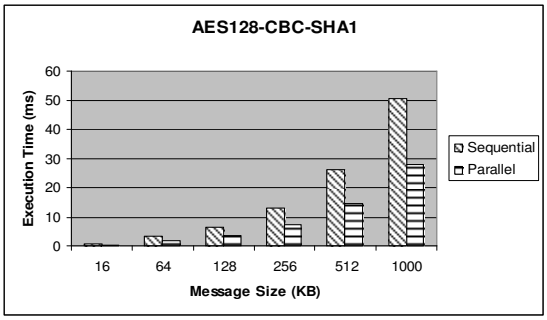

Figure 3: Parallel & Sequential Execution Time

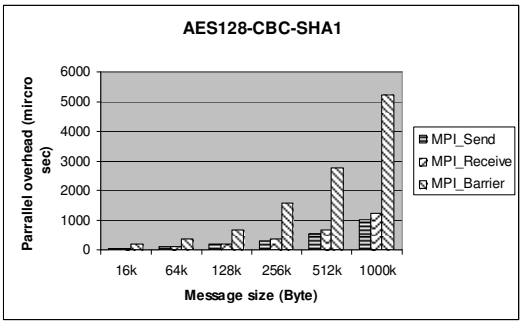

Figure 4: Parallel Overheads Execution Time

CONCLUSIONS

This paper proposed a new algorithm which provides a significant performance gain in bulk data transfer without compromising the security. The proposed algorithm performs the encryption of the data and the calculation of the MAC in parallel. The algorithm was simulated in two processors with one processor performing the MAC calculation and the other encrypting the data, simultaneously. The communication between the two processors was done via Message Passing Interface. Based on the performance simulations, the new parallel algorithm gained speedup of 1.74 with 85% efficiency over the sequential (current) algorithm. With respect to parallel overhead, we conclude that the barrier synchronization time takes a very large portion of about 65% compared with the communications time. This is obvious since the HMAC and padding are faster than AES.

REFERENCES

CryptoSys. 2006. *Crypto System API*. Accessed on 1 May 2006. www.cryptosys.net.

Dongara, P. and Vijaykumar, T. 2003. Accelerating Private-key cryptography via Multithreading on Symmetric Multiprocessors. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software.*

Elgohary, A, Sobh, T. and Zaki, M. 2006. *Design of an enhancement for SSL/TLS protocols*, Computers & Security.

Federal Information Processing Standards Publication. 2002. *The Keyed-Hash Message Authentication Code (HMAC),* US

Hodjat A. and Verbauwhede I. 2004. *A 21.54 Gbits/s Fully Pipelined AES Processor on FPGA*. IEEE Symposium on Field- Programmable Custom Computing.

MPICH. 2005. *MPICH-A Portable Implementation of MPI,* Accessed on 1 May 2006. http://www-unix.mcs.anl.gov/mpi/mpich1

Rescorla, E. 2000. *SSL and TLS Designing and Building Secure System* , 1st ed., New Jersey : Pearson Education Corporate Sales Division.

Quinn, M. 2003. *Parallel Programming in C with MPI and OpenMP,* 1st ed., New Jersey : McGraw-Hill higher education.

Stalling, W. 2003. *Cryptography and Network Security*, 3rd ed., New Jersey: Pearson Education International.

Zhao, L., Makineni, S., and Bhuyan L. 2005. Anatomy and Performance of SSL processing, In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software.*