Design and Implementation of TLS Accelerator

Recep Onur Yıldız
Computer Engineering Department
Istanbul Technical University
Istanbul, Turkey
yildizr@itu.edu.tr

Ayse Yilmazer-Metin
Computer Engineering Department
Istanbul Technical University
Istanbul, Turkey
yilmazerayse@itu.edu.tr

***Abstract*—Communication devices deploy messages to another one over either a controlled channel or a non-controlled channel. When the communication channel is non-controlled, the deployed message is available for anyone that is able to listen. But, the deployed message may comprise private information that needs to be shared securely. Therefore, to establish a secure communication over a controlled or a non-controlled channel, cryptography algorithms are used. The Transport Layer Security (TLS) is a protocol that includes several cryptography algorithms. With its algorithms, TLS provides authentication, confidentiality, and data integrity features. These features achieved with repeated calculations. Yet, these calculations increase the power consumption and degrade the performance of a sequential processor. To overcome these problems, an accelerator can be used for the calculations of TLS protocol. In this work, a TLS accelerator is designed and implemented on an FPGA. Our design aims to increase the performance and decrease the power consumption of sequential processor during the TLS calculations. The proposed accelerator is implemented with Xilinx Vivado and the implementation results show that the proposed accelerator consumes 0.86 Watt power. The accelerator is simulated with Vivado in order to measure the throughput of the proposed accelerator. The encryption throughput of the proposed accelerator at 100 MHz operating frequency is observed as 700 Mbps. Also, the proposed accelerator provides 52.4 handshake connections per second.**

I. Introduction

The secure communication is essential especially when the message includes private information. When the communication medium is a controlled channel, the secure communication can be established without precautions. Unfortunately, it is not always possible to have a controlled channel. In case of non-controlled channel, the cryptography algorithms can establish a secure communication. The devices in the communication channel can be authenticated with these algorithms. With the authentication, the devices will have the same key that is used in encryption and decryption algorithms. The encryption algorithms produce the cipher text that is encrypted message. Also, the decryption algorithms generate the plain text that is decrypted cipher text. It is impracticable to produce the same plain-cipher text tuple without the key that is used in the cryptography algorithm. Thus, the message becomes confidential. In addition to confidentiality, the integrity of the message can be verified with cryptography algorithms. Since the communication channel is non-controlled, the deployed message can be modified. When the deployed message is modified, it should be detected by the receiver. The receiver detects the interference on the deployed message by using the hash value. Thus these three main features provide secure communication.

Transport Layer Security (TLS) is a widely used protocol in Internet of Things (IoT) and Vehicle to Everything (V2X) applications [1]. TLS can be used for authentication of devices, confidentiality of communication, and data integrity of messages [2]. TLS protocol comprises several cryptography algorithms that provide these three features. Despite the benefits of these algorithms, their implementations degrade the performance and increase the power consumption of a general purpose sequential processor. That algorithms include repeated arithmetic operations and bitwise manipulations of big numbers that can be more than 128 bits wide. But, a sequential core has limited functional units and there are usually frequent memory accesses during these calculations. Therefore, implementation of TLS protocol in a sequential processoe degrades the performance as they may cause frequent pipeline stalls and cache misses. Contrarily, the sequential core provides ease of development in the IoT and V2X applications. While having the usability of a sequential processor, to provide better performance with these applications, an accelerator implementing TLS protocol can be designed and attached to the processor. In this work, we designed and implemented a TLS protocol accelerator. Our accelerator implements the Diffie-Hellman Key Excahnge (DHE), Rivest Shamir Adleman (RSA), Cipher Block Chaining (CBC) mode of Advanced Encryption Standard (AES) with Cipher based Message Authentication Code (CMAC), and CBC mode of AES with Secure Hash Algorithm (SHA).

Isobe et al. [3] also presented a TLS/SSL implementation on FPGA. They designed RSA, AES, RC4, MD5, and SHA algorithms. In this work, the presented throughput of the encryption was 10 Gbps and power consumption was 23 Watt. Comparing with our proposed system, it has higher throughout. But, the power consumption is also higher than our system and it is too much for IoT and V2X applications. To the best our knowledge, it is the only TLS protocol implementation that contains RSA, AES, and SHA algorithms in the literature.

The rest of this paper is organized as follows: Section II explains the basics of TLS protocol and its comprised algorithms. In Section III, we explain the implementation of the accelerator. The simulation result of sequential core performing modular exponentiation is represented in Section IV. Also, the implementation and simulation results of the proposed accelerator are given in the Section IV. The paper is concluded and future works are given in Section V.

II. TLS Protocol

TLS protocol comprises several cryptography algorithms to provide secure communication. It utilizes RSA and DHE algorithms for sharing secret key. The shared secret key is used in encryption and decryption. There are several options for the encryption and decryption in TLS protocol. The CBC mode of AES is one of these options. Moreover, the shared secret key is used to validate the data integrity of the received message using hash values. In TLS protocol, the hash value can be calculated using CMAC or SHA algorithms.

The DHE and RSA algorithms allow to share a secret key securely. To utilize DHE algorithm, the devices in the communication have their public and private keys. The first step of the protocol is to encrypt a public variable with the private key. Both server and client share the encrypted public variable with the other one. The next step is to encrypt the received data. The server encrypts the client's shared value and the client encrypts the server's value. They both use their own private key for encryption. The results of these encryption are the same because of the modular exponentiation property. Therefore, the server and the client have the same shared secret key. Since they use modular exponentiation in encryption, it is hard to recover their private key from the shared results. After the secret key is shared, RSA algorithm is used to ensure that both devices have the same shared secret key. In RSA, the server has a public key and a private key. The private key is the modular inverse of the public key. The client encrypts the shared secret key with server's public key and then, sends the result to server. Since the private key of server is modular inverse of its public key, server decrypts the received data. If decrypted shared secret key matches with server's calculated one, the devices are authenticated.

In TLS protocol, the hash value can be calculated with CMAC or SHA algorithms. The CMAC uses the AES algorithm to calculate the hash value. The data is divided into fixed length bit blocks and these blocks are encrypted with AES. Except the first and last blocks, a plain text block is XOR-ed with cipher of previous block. The first block is directly encrypted and the last block is XOR-ed with subkey in addition to cipher of previous block. The CMAC algorithm utilizes 2 subkeys and one of them is selected for hash generation depends on the length of the plain text. CMAC algorithm is represented in Fig. 1.

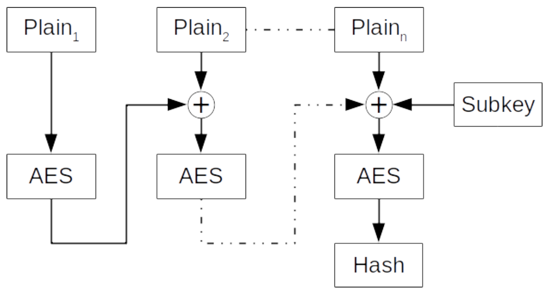

Fig. 1. Block diagram of CMAC

SHA algorithm is the another way to calculate hash value in TLS. SHA algorithm uses the Keccak function to calculate the hash value. The input of the Keccak function is plain text concatenated with two bits. Also, the Keccak function defines the padding that appends a string to produce required length of string. That padded plain text is divided into fixed length blocks and the Keccak permutation function is applied to each of them. The output of Keccak permutation function is XOR-ed with plain text block and the result is the input of next function. The Keccak permutation function consists of several operations and the function is repeated for 24 times in TLS protocol [4]. SHA algorithm is represented in Fig. 2.

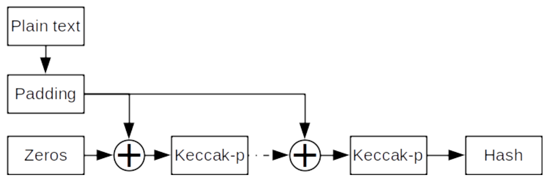

Fig. 2. Block diagram of SHA

The CBC mode of AES algorithm can be used to provide confidentiality in TLS protocol. The plain text of encryption is the message concatenated with its hash value. The plain text is divided into fixed length blocks and AES algorithm is used for encryption. The first block is XOR-ed with the initialization vector and the result of XOR is encrypted. The following blocks are XOR-ed with the cipher of previous block and then, they are encrypted. The CBC mode of AES is illustrated in Fig. 3.

In TLS protocol, a device decrypts the cipher text that is received from the other device. The decrypted cipher text contains the message and its hash value. To check the data integrity of message, the device calculates hash value. If the calculated and the received hash values match, the data integrity is ensured.

III. Implementation

In this work, TLS protocol is implemented in the proposed accelerator. The architecture of the proposed accelerator is represented in Fig. 4. The modular exponentiations of DFE and

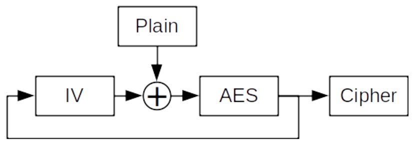

Fig. 3. Block diagram of AES in CBC mode

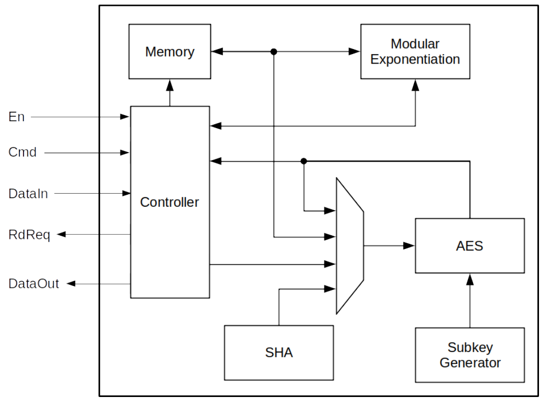

Fig. 4. Architecture of proposed accelerator

RSA is calculated by the *ModExp*. The *AES* is used during the encryption, decryption, calculation of hash value with CMAC. The hash value can be also calculated with *SHA*. The subkeys of CMAC algorithm are calculated with the *SubKey*. In our design, the *Controller* is responsible for routing the intermediate results and managing the data transfer with the outer system. The inputs of the proposed accelerator are *clock*, *reset*, *enable*, and *data input*, and the outputs are *read request* and *data output*. The *data input* and *data output* are 128 bits wide.

A. Controller

The controller is responsible for performing the selected algorithm of TLS protocol or storing the values. It routs the inter-calculation results, stores the final result, and communicates with the outside of the proposed system. The controller has 9 operational modes. The 4 of them are storing the private key, public key, public value, shared secret key. If the DHE algorithm and RSA are not used for secret key sharing, the shared secret key can be loaded into the accelerator. Otherwise, the DHE and RSA algorithms are handled by the accelerator. The remaining modes are setting the accelerator for CBC mode of AES with CMAC and CBC mode of AES with SHA. With these modes, the proposed accelerator performs the encryption or decryption.

B. ModExp

The modular exponentiation is calculated with Montgomery Multiplication algorithm [5]. The algorithm propose a residue class to prevent the redundant division operation in each iteration of loop. The implementation of Montgomery Multiplication includes shifting and addition. The shifting can be implemented easily within the hardware. Yet, the addition may increase the path delay because of the long carry chain. Therefore, to prevent the long chain, carry-save adder is used in the implementation of Montgomery Multiplication. The carry-save adder gets three inputs and generate two outputs as carry and sum. The *carry* and *sum* represents the bitwise carry and sum output, respectively. Therefore, the dependency between the rightmost and leftmost bits is eliminated. The modular exponentiation is calculated as in Algorithm 1. Since there is no data dependency between Step 5 and Step 7, these steps can be executed in parallel to improve the throughput of modular exponentiation. Therefore, there are 2 Montgomery multiplier in the proposed system.

Algorithm 1 Modular Exponentiation algorithm

```
1: $M^* \leftarrow Montgomery(M, R^2, N)$
2: $S \leftarrow R \ mod \ N$
3: for $i = 0 \ to \ k_e - 1$ do
4:     if ($e_i = 1$) then
5:         $S \leftarrow Montgomery(M^*, S, N)$
6:     end if
7:     $M^* \leftarrow Montgomery(M^*, M^*, N)$
8: end for
9: $C \leftarrow Montgomery(S, 1, N)$
10: return $C$
```

C. AES

AES algorithm in the proposed system is implemented using Rijndael algorithm [6]. This algorithm encrypts the 128 bits width block of data with key. The encryption comprises rounds of non-linear byte substitution, cyclic shifting, bit mixing, and XOR-ing with the round key of loop. The round keys in Rijndael algorithm are generated from the initial key using key expansion function. Our implementation of Rijndael algorithm is based on the implementation that is shared by National Institute of Standards and Technology (NIST). The pipelined implementation of Rijndael is selected to achieve higher throughput.

D. SHA

SHA algorithm processes the plain text with functions defined in *Keccak* function. *Keccak* function appends the necessary bits to end of the plain text and the extended plain text is divided into fixed length of blocks. *Keccak* function comprises *Keccak-p* permutation function that includes rounds of 5 functions. A round of *Keccak-p* function is defined as in Eq. 1.

$$round = \iota(\chi(\pi(\rho(\theta(A)))), i_r) \tag{1}$$

The round number of *Keccak-p* is defined as 24 for SHA algorithm. ι, χ, π, ρ, and θ functions are implemented as defined in [4]. The input of ρ is the output of θ, the input of π is the output of ρ and so on. Therefore, these functions can not be implemented in parallel architecture. Moreover, the round index is calculated in a loop. To improve the throughput, that loop can be flattened or round indexes can be stored in memory since they are constant. In the proposed accelerator, round indexes are stored in the memory.

IV. Results

The proposed accelerator is intended to increase the usability of a sequential core in applications that utilizes TLS protocol. To observe the overheads of TLS protocol, ibex core is simulated with Spike simulator. Spike is a simulator

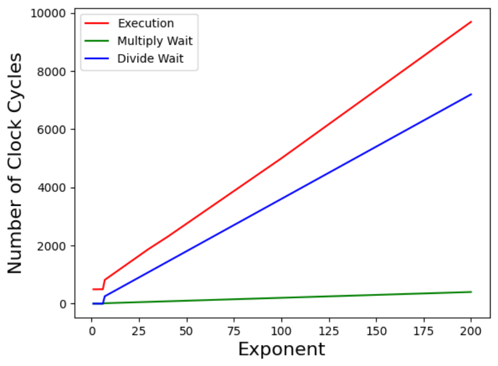

Fig. 5. Modular exponentiation simulation result on RISC-V core

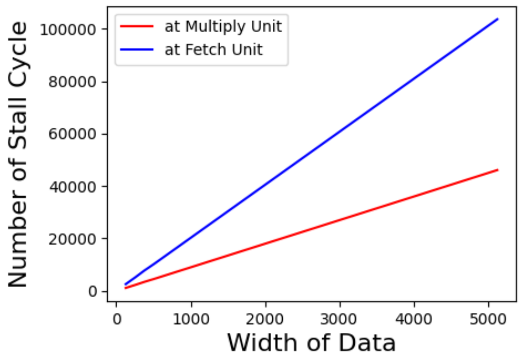

Fig. 6. AES block cipher simulation result on RISC-V core

that compiles the given program and simulates it on RISC-V core [7]. Ibex is an open source 32 bit RISC-V core that has 2 stages pipeline [8]. The TLS protocol uses the modular exponentiation as encryption in DHE and RSA. We simulate a simple program that calculates the modular exponentiation for various exponent values. The simulation result is illustrated in Fig. 5. As the exponent increases, the clock cycle required for the calculation is increases. Also, the calculation causes stalls since the functional units are busy. Ibex core is also simulated with a program which encrypts varying length of plain text with AES block cipher. The simulation result is illustrated in Fig. 6. It is shown that number of stall cycle increases with the increase in width of plain text. Yet, the designed accelerator perform AES encryption without causing stall as it is pipelined.

The proposed system is implemented and simulated with Xilinx Vivado v2019.2 and the FPGA is selected as XC7A200T. The implementation results shows that the proposed system can operate at 100 MHz clock frequency. Also, the power consumption of the system is 0.86 Watt. The resource usage of the proposed system is given in the Table I.

The number of handshake per second and throughput of the encoding is observed from the simulation results of proposed system. The DHE and RSA algorithms are performed within

Utilization	**Used**	**Utilization%**
Slices LUTs	18169	13.58
Slice Registers	11132	4.16

TABLE I
RESOURCE USAGE OF PROPOSED SYSTEM

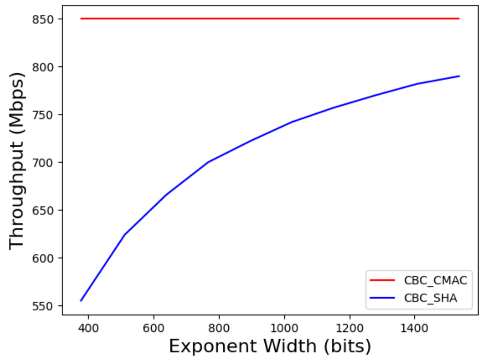

Fig. 7. Throughput of CBC with CMAC and CBC with SHA

19 ms. In the simulation, the widths of base, exponent, and modulo are selected as 2048 bits. The sequential core perfroms modular exponentiation with 20 bit exponent within the same duration that is 19 ms. Therefore, the proposed accelerator reduces the power consumption because of the reduced clock cycle. The throughput is simulated with various length plain texts and the result is illustrated in Fig. 7. Since the proposed accelerator gets the message 128 bits at a time, the proposed accelerator has to wait for the whole plain text to calculate hash value with SHA But, CMAC can calculate the hash value block by block. Therefore, while CBC with CMAC has constant throughput, throughout of CBC with SHA increases as the width of plain text increases. But, it saturates since the AES implementation takes fixed time for calculation.

V. Conclusion and Future Works

In this work, an accelerator for TLS protocol is proposed. The proposed accelerator is capable of performing DHE, RSA, CBC mode of AES, CMAC, and SHA. To observe the overhead of TLS protocol, calculation of modular exponentiation is simulated with Spike on ibex core. Moreover, the accelerator is implemented and simulated with Xilinx Vivado. The resource usage and power consumption of the accelerator is observed from the implementation results. Also, the throughput of the proposed accelerator is simulated.

As the future work, the accelerator will be attached to a RISC-V core that contains modulation and demodulation accelerator. Therefore, the system will be used in the application that requires encryption and decryption for the communication, such as IoT and V2X applications. Also, the Elliptic Curve Cryptography algorithms will be implemented on the designed accelerator.

References

[1] "Ieee standard for wireless access in vehicular environments (wave)–certificate management interfaces for end entities," *IEEE Std 1609.2.1-2020*, pp. 1–287, 2020.

[2] K. McKay and D. Cooper, "Guidelines for the selection, configuration, and use of transport layer security (tls) implementations," 2019-08-29 2019.

[3] T. Isobe, S. Tsutsumi, K. Seto, K. Aoshima, and K. Kariya, "10 gbps implementation of tls/ssl accelerator on fpga," in *2010 IEEE 18th International Workshop on Quality of Service (IWQoS)*, 2010, pp. 1–6.

[4] M. Dworkin, "Sha-3 standard: Permutation-based hash and extendable-output functions," 2015-08-04 2015.

[5] P. L. Montgomery, "Modular multiplication without trial division," *Mathematics of Computation*, vol. 44, pp. 519–521, 1985.

[6] M. Dworkin, E. Barker, J. Nechvatal, J. Foti, L. Bassham, E. Roback, and J. Dray, "Advanced encryption standard (aes)," 2001-11-26 2001.

[7] [Online]. Available: https://github.com/riscv-software-src/riscv-isa-sim

[8] [Online]. Available: https://github.com/lowRISC/ibex