Design and Implementation of a Fast, Combined SHA-512 on FPGA

Seyyed Ali Emam[†] **and Sareh Sadat Emami** [††],
emam@ce.shari.edu emami@ee.kntu.ac.ir
[†]Computer Engineering Department, Sharif University of Technology, Tehran, Iran
[††]Electrical Engineering Department, Khajeh Nasiredin Tusi University of Technology, Tehran, Iran

Summary
In this paper we propose a new method for generating digital signature based on SHA-512 hash algorithm. This method uses a secret key and enjoys the benefits of the private key cryptography. Using two SHA-512 modules in parallel for generating two 512-bit temporary signatures, we permute signatures by the given secret key and generate 1024-bit signature. We have synthesized and verified the efficiency of our algorithm on a Xilinx VIRTEX4 FPGA by applying multiple scenarios.

Key words:
SHA-512, FPGA, Permutation, Digital Signature

1. Introduction

Data integrity assurance and data origin-authentication are essential security services in financial transactions, electronic commerce, electronic mail, software distribution, data storage and so on. The broadest definition of authentication within computing systems includes identity verification, message origin-authentication and message content authentication [5].
To achieve the required processing capabilities, using hardware components seems necessary. These hardware cores are usually implemented either in dedicated ASIC cores or in reconfigurable devices [1]. In this report, we use Xilinx reconfigurable devices for implementing our design.
By using private key increased security, attackers cannot generate messages offline since they don't know the secret key [3]. Thus, the method is much more secure than SHA-512 facing birthday attacks.
In this approach, we use two SHA-512 modules for generating two 512 bit signatures. Then, using the secret key we permute them and generate a 1024-bit signature.
The paper is organized as follows; Section 2 presents implementation of a SHA-512 module with FPGA. Section 3 describes the proposed design and its implementation details. In Section 4 we analyze the performance of our method and finally in section 5, we study the security of the proposed method and compare it with the SHA-512 approach.

Manuscript received May 5, 2007
Manuscript revised May 20, 2007

2. SHA-512 implementation

In 1993 the Secure Hash Standard (SHA) was first published by the NIST.
In 1995 this algorithm was reviewed in order to eliminate some of the initial weakness, and in 2001 new Hashing algorithms were proposed. This new family of hashing algorithms known as SHA-2, use larger digest messages, making them more resistant to possible attacks and allowing them to be used with larger blocks of data, up to 2128 bits, e.g. in the case of SHA-512 [6].
For implementing SHA-512, we break it to some smaller modules to ease the implementation task in Verilog description language.
These modules are:

- SHA-512 Data path
  - Message Scheduler
  - K_i memory
  - Round
- SHA-512 controller

The data path section provides the flow of data and the controller involves a state machine that controls the data flow in SHA-512 data path. Description of each part is as follows:
Message Scheduler: This unit generates 80 message-dependent words W_t. The first 16 words are simply the first 16 words of the input message block. The remaining words are computed using simple feedback function, based on rotations, shifts and XOR operations.
Message Scheduler receives a 1024-bit message block in 64-bit chunks through X_{in}. According to Fig.1 this module has a 16×64-bit shift register and one multiplexer. During the first 16 pulses of the system clock, whole message block enters into the shift register and then the multiplexer selects W_{in} to provide feedback for the shift register. Message Scheduler generates W_i for the hash algorithm in each clock pulse and $W_{out,}$ the output signal of the module, is the W_i at $Round_i$.

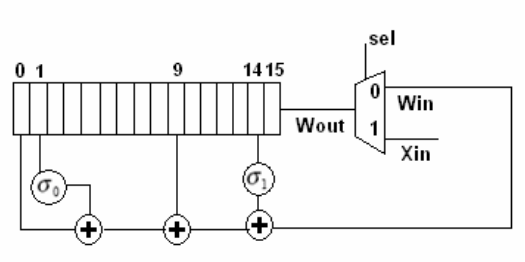

Fig.1. Message Scheduler Block Diagram

K_i Memory: According to NIST Standard, SHA-512 uses the sequence of eighty constant 64-bit words, K_0, K_1,…, K_{79}. These words represent the first sixty-four bits of the fractional parts of the cube roots of the first eighty prime numbers [6]. To store the constant values, we used an 80×64-bit Xilinx specific Block ROM.

Round: For each message block, the standard hash algorithm runs a specific round for 80 times and then generates a 512-bit output which is an initial value of the algorithm for the next message block. After processing of the all 1024-bit message blocks, the last 512-bit output is the signature of message [3] (Fig.2, Fig.3).

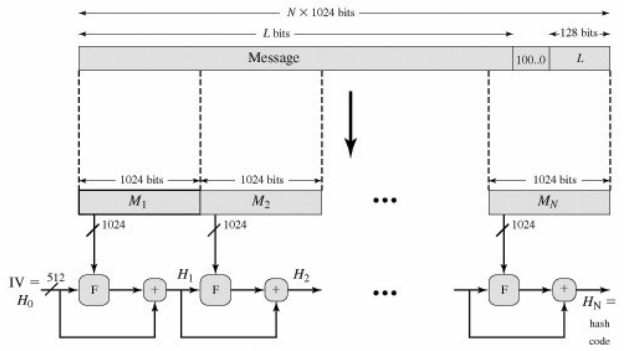

Fig.2. Message digest generation using SHA-512

There are two ways to implement rounds architecture: full loop unrolling and iterative looping [4,5].
The full loop unrolled architecture has an 80-step combinational logic which requires huge amount of area beside the complexity of generating Wi. By implementing one of the steps of the SHA-512 algorithm, the looping architecture with 80 iterations would seem to provide the most area-efficient solution. The block diagram of an iterative core is shown in Fig.4.
At the first round, Register1 is the initial value (IV) vector which is a 512-bit constant number.

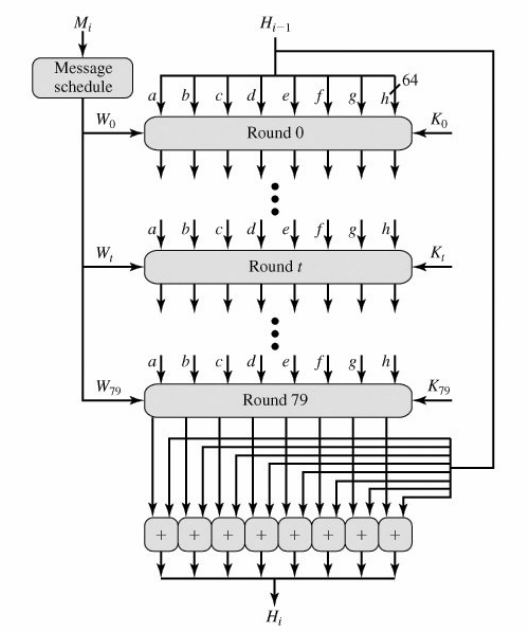

Fig.3. SHA-512 processing of a single 1024-bit block

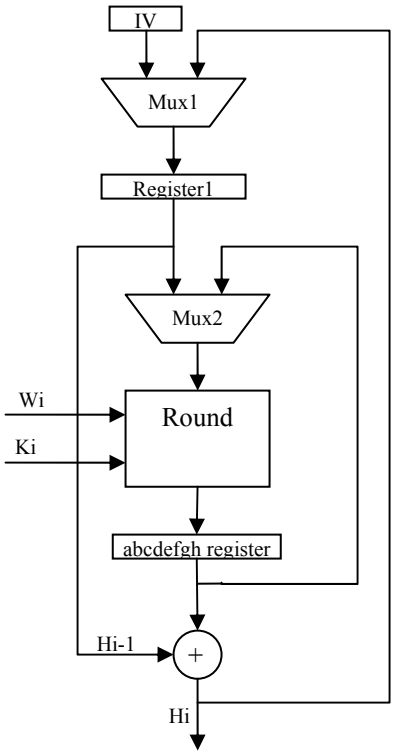

Fig.4. Iterative core block diagram

After the first round, Register1 holds the Hi of the previous round (Hi-1).
The round is repeated 80 times and this is done by the Mux2 multiplexer which returns the output feedback of each round to the Round block. The output achieved by each round is stored in ABCDEFGH register to prepare

feedback value for the next round. After 80 rounds, digest of one 1024-bit message block becomes ready at Hi.

Controller: This module generates all necessary controlling signals such as multiplexers' selectors, registers' latch enables, ROM addresses (for K_i) and etc.
We simulated our implementation using ModelSIM XEIII 6.1e. In order to validate the simulation results, we compared them with signatures calculated by Crypto++ C library for a given message block.

3. Proposed method

To implement our method we use two SHA-512 modules in parallel. A message is passed to SHA-512 modules in 64-bit chunks, alternatively. For a given message, there will be two 512-bit temporary signatures. There is a permutation unit that permutes two temporary signatures with the given secret key which is stored in a 1024×10-bit Block ROM. Each word of ROM indicates the position of corresponding bit in output signature. After permutation of the 1024-bit temporary signatures, the final signature becomes ready (Fig. 5).
As Fig.6. shows, the structure of a round consists of multiple add operations (eq. 1 and eq. 2) in series that forms the critical path of this structure.

$$E_{t+1} = D_t + \Sigma_1(E_t) + Ch(E_t, F_t, G_t) + H_t + K_t + W_t \quad (1)$$

$$A_{t+1}=\Sigma_0(A_t)+Maj(B_t,C_t,D_t)+\Sigma_1(E_t)+Ch(E_t,F_t,G_t) + H_t + K_t + W_t \quad (2)$$

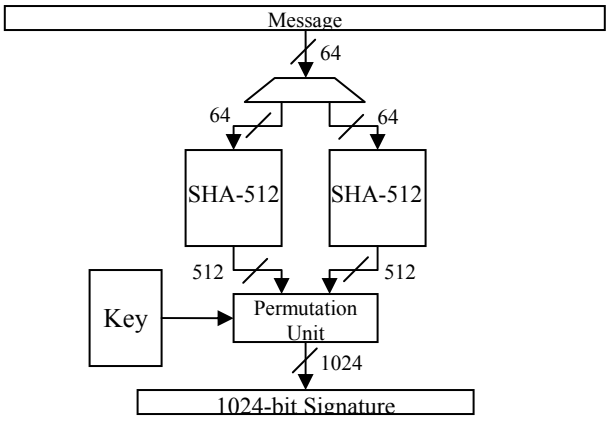

Fig.5. Combined SHA-512

In our implementation we use Carry Save Adder (CSA) [7] instead of Carry Propagate Adder (CPA) blocks for calculating A_{t+1} and E_{t+1} (Fig.7) which is known for its best-parallelizable architecture. This implementation strategy reduces the critical path delay significantly.

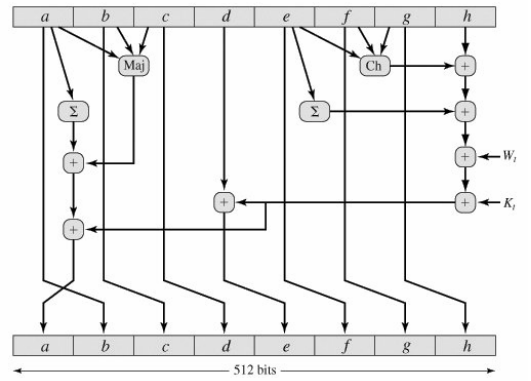

Fig.6. Single round structure

In order to reduce the design area further, we used the shift register mode of CLB [1] slices available in Xilinx Virtex4 FPGA devices in Message Scheduler implementation instead of generic shift registers.

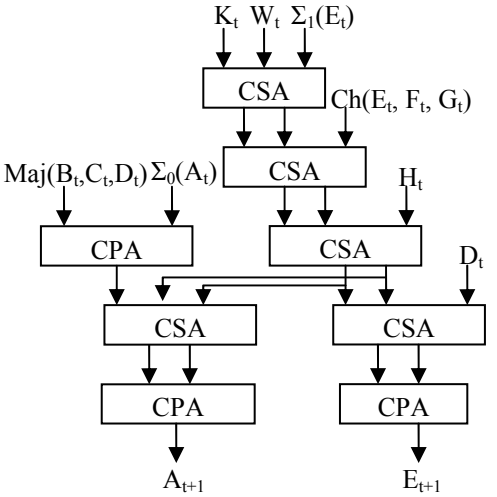

Fig.7. Calculating At+1 and Et+1 using CSA blocks

4. Performance analysis

All modules were synthesized and placed and routed on Xilinx Virtex4 xc4vsx35ff668-12 target device.
In the case of SHA-512 the utilization of slices was 1921 out of 15360 (12%) and number of consumed slice flip flops was 2240 out of 30720 (7%). The timing analysis shows 118.043 MHz as the maximum frequency of the design.
Maximum data throughput can be simply computed by the following equation:

[1] Configuration Logic Block

$$Throughput = S_{mb} \times F_{max} / R \qquad (3)$$

where:

S_{mb}: *message block size*
F_{max}: *Maximum clock frequency*
R: *Number of rounds*

Using the (eq. 3) maximum expected throughput is: 1.51Gbps (1024b × 118.043MHz / 80).

For the combined SHA-512 method, without using CSA, the utilization of slices is 3541 out of 15360 (23%) and number of slice flip flops is 4490 out of 30720 (14%) with the maximum clock frequency of 117.845 MHz and maximum throughput of 2.51Gbps.

In this model, it takes 80 + 16 clock pulses for two 1024-bit message blocks. Extra 16 clock pulses are needed for second SHA-512 message scheduler. This throughput is valid for one message because of the permutation delay (about 8.7us). Permutation can be overlapped by the processing of the next message if its length is at least 13 message blocks.

After using carry save adders in combined SHA-512 method, because of the CSA adder overhead, the slice utilization increased to 3826 out of 15360 (24%) and number of used slice flip flops increased to 4528 out of 30720 (14%). However, the maximum clock frequency rose to 135.466 MHz which results in improved throughput of 2.89Gbps with 7.5us permutation delay for each message (2.89Gbps = 2×1024b×135.466MHz /(80+16)) .

5. Security analysis

The security of hash functions is determined by the size of their outputs, referred to as hash values, n. The best known attack against these functions, the "birthday attack", can find a pair of messages having the same hash value with a work factor of approximately $2^{n/2}$ [3].

Therefore, for a SHA-512 module, complexity of the best attack is 2^{256}. For our method each SHA-512 has the same attack complexity, but our permutation process increases the attack complexity in proportion to the size of the secret key.

If we use a 1024-bit secret key, permutation complexity becomes $2^{10}!$ which yields the total complexity of $2^{10}! \times 2^{256}$ and it's much more secure than the SHA-512 algorithm.

6. Conclusion

In this paper, we propose a new idea to generate a signature for messages based on the well-known SHA-512 algorithm. The implementation and simulation results show our method superiority over the basic algorithm in throughput and security features. Because of permutation delay, the new method is suitable for long messages (longer than 1.5kB).

Also, the numerical results prove the suitability of the FPGAs as the implementation platform for cryptographic accelerators.

References

[1] R.Chaves, G.K.Kuzmanov, L.Sousa, S.Vassiliadis, "*Improving SHA-2 Hardware implementations*", Workshop on Cryptographic Hardware and Embedded Systems (CHES), October, 2006.

[2] T. Grembowski, R. Lien, K. Gaj, N. Nguyen, P. Bellows, J. Flidr, T. Lehman, and B. Schott, "*Comparative analysis of the hardware implementations of hash functions SHA-1 and SHA-512*", in ISC (A. H. Chan and V. D. Gligor, eds.), vol. 2433 of Lecture Notes in Computer Science, pp. 75{89, Springer, 2002.

[3] William Stallings, "*Cryptography and Network Security Principles and Practices, Fourth Edition*", Prentice Hall, 2005.

[4] R.P. McEvoy, F.M. Crowe, C.C. Murphy, and W.P. Marnane, "*Optimisation of the SHA-2 family of hash functions on FPGAs*", IEEE Computer Society An-nual Symposium on Emerging VLSI Technologies and Architectures ISVLSI'06), 2006.

[5] Janaka Deepakumara, Howard M. Heys, and R. Venkatesan. "*FPGA Implementation of MD5 Hash Algorithm*". In Canadian Conference on Electrical and Computer Engineering (CCECE), May 2001.

[6] NIST, *"Announcing the Standard for Secure Hash Standard, 180-2*", tech. rep., National institute of Standard, August 2002.

[7] Behrooz Parhami, "*Computer Arithmetic, Algorithms and Hardware Design*", Oxford University Press, 2000.