Proceedings of the 5th WSEAS Int. Conf. on MULTIMEDIA, INTERNET AND VIDEO TECHNOLOGIES, Corfu, Greece, August 17-19, 2005 (pp130-135)

High-Speed and Low-Power Implementation of Hash Message Authentication Code through Partially Unrolled Techniques

H.E.MICHAIL, A.P.KAKAROUNTAS,E.FOTOPOULOU, C.E.GOUTIS
Electrical & Computer Engineering Department
University of Patras
GR-26500 Patra
GREECE

Abstract: - In this paper an efficient implementation, in terms of performance, of the keyed-hash message authentication code (HMAC) using the SHA-256 hash function is presented. This mechanism is used for message authentication in combination with a shared secret key. The proposed hardware implementation, invokes a partially unrolled implementation for the underlying hash function leading to a high-throughput and low-power implementation for the whole HMAC construction. Special care has been taken so that the proposed implementation doesn't introduce extra design complexity; while in parallel functionality was kept to the required levels.

Key-Words: - Security, HMAC, Hash functions, SHA-256, Partial- Unrolling, High-Throughput, Hardware Implementation

1 Introduction

The scope of an HMAC implementation is to authenticate both the source of a message and its integrity without the use of any additional mechanisms. This is achieved by attaching a digital signature to the message [1]. The digital signature is generated by the HMAC itself after supplying it with the message. HMAC's have two functionally distinct parameters, a message input and a secret key known only to the message originator and intended receiver(s). HMAC is used not as a cipher, but rather as a mechanism for signing a packet with a key at one end of the connection, and then verifying the signature at the other end using the same key. Without the key it is infeasible to generate a packet with the correct signature.

Due to an essential need for security, in mobile services as specified in the WTLS security level of WAP , in Public Key Infrastructure (PKI), in SSH protocol and in many other applications an efficient and small-sized HMAC implementation is very important. Especially, nowadays that security is a major demand, due to the rapid evolution of plethora of communications standards. In addition to the demanded high security level, the need for high performance is a significant factor for the selection of a security implementation. Thus, hardware implementation is far more suitable, for security issues, compared to the corresponding software implementations. The proposed implementation is structured in such way so that it can be used in a variety of applications, maintaining the flexibility of similar software constructions.

The security level of a HMAC implementation is based on the underlying cryptographic hash function. Up to now MD-5 and SHA-1 were used but lately the security problems have been discovered in both SHA-1 [2] and MD-5 [3]. It is clear enough that in the next time new implementations of HMAC mechanism are going to be demanded by the market incorporating hash functions with higher levels of security which in turns results to a higher level of security for the whole HMAC construction. For this reason in this paper the proposed HMAC implementation uses the SHA-256 hash function which up to now is considered as safe.

But not only that ,having in mind all new applications that need a HMAC mechanism the proposed architecture leads to a high-throughput and low-power implementation. Authentication to Virtual Private Networks (VPN's) that companies are establishing in order to exploit on-line collaboration, digital signature algorithms like DSA that are used for authenticating services like electronic mail, electronic funds transfer, electronic data interchange, software distribution, data storage etc, security in networks and mobile services, as in SSL, which is a Web protocol for establishing authenticated and encrypted sessions between Web

servers and Web clients are cases where a high-speed HMAC mechanism is required for the corresponding server of the application ,which has to reach the highest degree of throughput in order to satisfy immediately all requests for service from all users-clients

The rest of this paper is organized as follows. In section 2 a general description of hash functions algorithms is given and the existing implementations techniques are presented. In section 3 the proposed implementation is presented in depth, providing details regarding the architecture, the logic and the modifications to decrease the critical path. In section 4 examples of implemented hash functions in FPGA technology are compared to other implementations. Finally in section 5 the paper concludes.

2 Proposed SHA-256 Implementation

Hash functions are iterative algorithms which in order to compute the final message digest they perform a number of identical or slightly different operations. The Secure Hash Standard [4] describes in detail the SHA-256 hash function. Throughput is kept low due to the large number of the required operations. An approach to increase significantly throughput is the application of pipeline.

This has a small area penalty but leads to a significant higher throughput which is the main need of market considering the servers for VPN's, DSA etc. For this reason, most of the proposed optimized implementations exploit the benefits that pipeline offers, balancing the achieved throughput with the introduced area penalty.

From a survey to all hash functions it is clear enough that the best compromise is to apply four pipeline stages so as to quadruple throughput and keep the hash core small as well. This approach enables four operations to be carried out concurrently. Applying more pipeline stages is something that will violate the area constraints.

Applying pipeline the architecture of a SHA-256 core is formed as illustrated in Fig. 1 where there are four pipeline stages and a single operation block. The SHA-256 core needs many more components in order to function properly such as a Padding Unit, a MS-RAM, a Control Unit and a Constants Array Bank. These components exist in the whole HMAC architecture which is studied in the next section. In this section we will focus on the "pure" SHA-256 core and how can this be optimized beyond the application of pipeline.

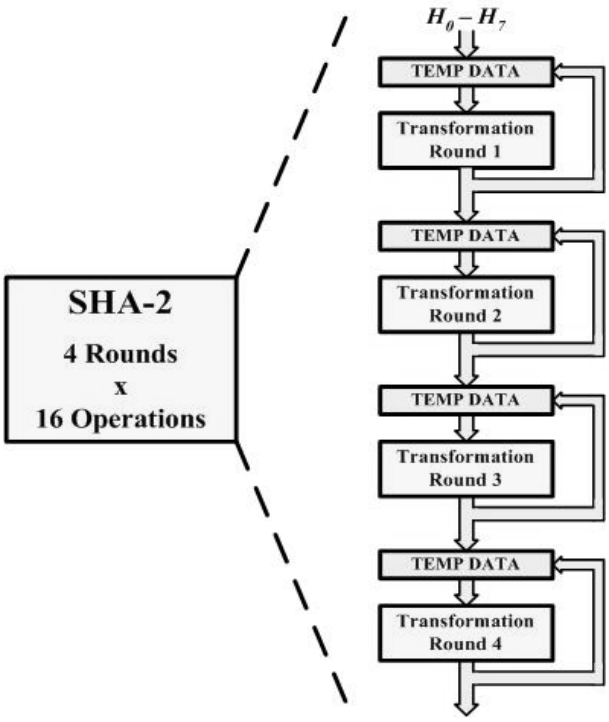

Fig. 1: SHA-256 core with 4 pipeline stages

The critical path of the illustrated architecture is obviously located between the pipeline stages where the operation block of the hashing core exists. In Fig.1 the operation blocks are referred as transformation rounds following the standard's terminology. Even in the whole HMAC implementation the critical path remains between the pipeline registers of the hashing core since the other units like MS RAM and Constants' Array, do not contribute due to their nature (memory and hardwired logic respectively), while control unit is a block containing very small counters which also don't contribute to the overall maximum delay.

Although the need for high throughput is recognized the performance of all hardware implementations are degraded because there has not been much effort in optimizing the conventional implementation of the transformation rounds which are responsible for performing every single operation.

In order to produce a hash function implementation with a higher throughput we should consider how throughput is calculated and then select which term of the formula should be manipulated. The throughput of a hash function implementation is given by the following equation:

$$Throughput_{conf} = \frac{\#bits \cdot f_{operation}}{\#operations} \qquad (1)$$

where *#bits* is equal to the number of bits processed by the hash function, *#operations* corresponds to the required clock cycles between successive messages to generate each Message Digest and $f_{operation}$ indicates the maximum operating frequency of the circuit.

From the above equation and considering that a message block, as provided by the padding unit, is at most 512 bits, the two terms that can be manipulated is either *#operations* or the circuit's operating frequency, $f_{operation}$. In the proposed technique a manipulation of the *#operations* is considered.

To explain the way that this manipulation is done let's consider two consecutive operations of SHA-256 hash function which are illustrated in Fig. 2.
Each one of the a_t, b_t, c_t, d_t, e_t, f_t, g_t and h_t is 32-bit wide resulting in a 256-bit hash value. K_t and W_t are constant values for iteration t and the t_{th} w-bit word of the message schedule ,respectively.

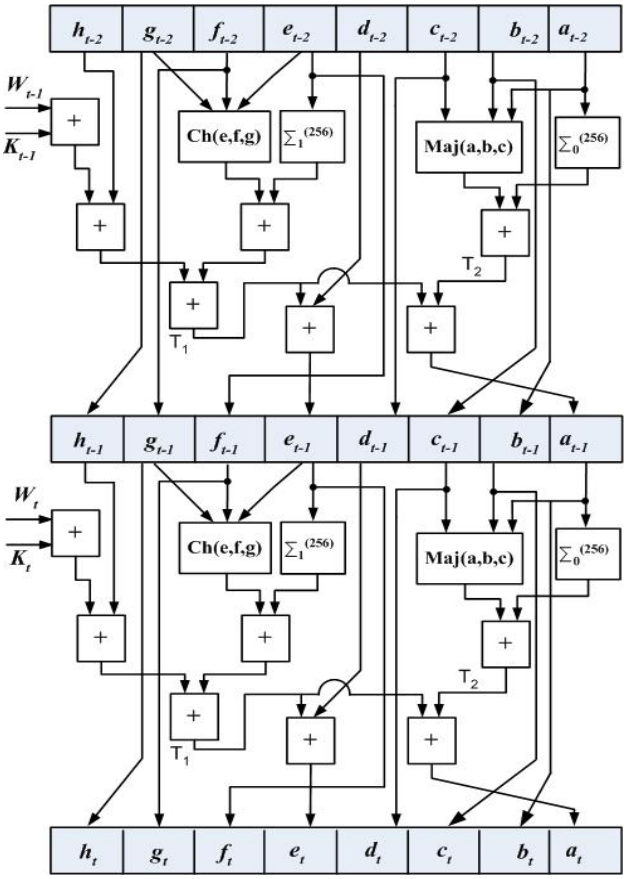

Fig. 2: Two Consecutive SHA-256 operations

The proposed design approach is based on a special property of the SHA-2 operation block. The considered inputs a_{t-2}, b_{t-2}, c_{t-2}, d_{t-2}, e_{t-2}, f_{t-2}, g_{t-2} and h_{t-2} go through a specific procedure in two operations and after that the considered outputs a_t, b_t, c_t, dt, e_t, f_t, g_t and h_t arise. In between the signals a_{t-1}, b_{t-1}, c_{t-1}, d_{t-1}, e_{t-1}, f_{t-1}, g_{t-1} and h_{t-1} exist that are outputs from the first operation and inputs for the second operation. Except of the signal a_{t-1} and e_{t-1} the rest of the signals b_{t-1}, c_{t-1}, d_{t-1}, f_{t-1}, g_{t-1} and h_{t-1} are derived directly from the inputs a_{t-2}, b_{t-2}, c_{t-2}, e_{t-2}, f_{t-2} and g_{t-2} respectively. This means consequently that also c_t, d_t, g_t and h_t can be derived directly from the X_{t-2} inputs.

Furthermore, some calculations during the operation are depended only on the primary operation block's inputs and on intermediate results that are sequentially computed. It seems that some of these calculations can be done in parallel for consecutive operations. In Fig. 3, the proposed implementation is presented in which two consecutive operations have been merged and thus their result is computed in only one clock cycle instead of two. The gray marked areas on Fig. 3 indicate the parts of the proposed SHA-256 operation block that operate in parallel and result to the concurrent computation of the primary operation block's outputs.

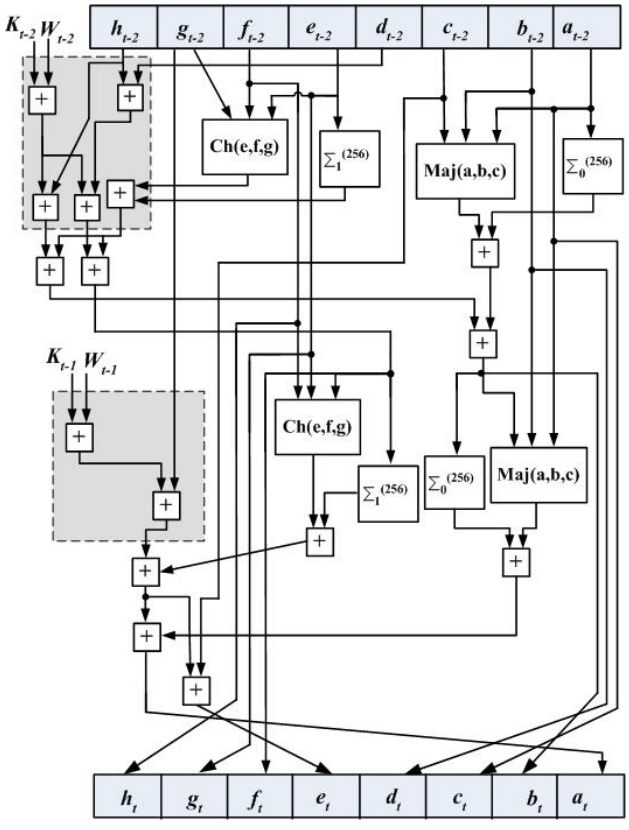

Fig. 3: Two Merged SHA-256 Operations

Inspecting Fig. 2 and Fig. 3 it is obvious that the critical path in the proposed implementation consists of six addition levels instead of the four addition levels consisting the critical path of the non-

concurrent implementation. Although, this fact reduces the maximum operation frequency in the proposed implementation, the throughput is increased significantly since the hash value in the proposed instrumentation is computed in only 32 clock cycles instead of 64 in the non-concurrent implementations. This computations leads to the result shows that the throughput of the proposed implementation increases 50% which in turn will increase by 50% the throughput of the total HMAC mechanism. The experimental result verifies this theoretical assumption.

3 Description of HMAC System

In the proposed HMAC architecture, pipeline technique is applied to increase both speed and throughput. The two process stages are performed with the usage of two pipelined SHA-256 cores, as they were presented in the previous section. The whole HMAC implementation is illustrated in the Fig. 4.

In the MS RAM, all message schedules Wt of the padded message are stored. The Constants Array is a hardwired array that provides the constant values K_t and the constant initialization values H_0 - H_7. Additionally, it includes the W_t generators and the Key Generation unit that is required at the HMAC mechanism. As it was previously mentioned this components are essential for the hashing cores to function properly.

The HMAC must firstly be initialized before processing any message. The initialization procedure corresponds to computing the hash values of two certain 512-bit blocks, which are the corresponding keys, and this is performed independently in the two SHA-256 cores. When the initialization procedure is completed, the hash values from the outputs of the two SHA-256 cores are stored in the Constants Array unit and are then used continuously in the two SHA-256 cores as the new initial values for $H_0 – H_7$. This is the first time that a message can be supplied for process to the first SHA-256 core as long as it has been padded, while the initialization procedure was in progress. In normal operation, the inputs $H_0 – H_7$ are the hash values that were pre-computed during the initialization process until a signal comes that makes the system change the used keys. Then the Key Generation Unit computes the new keys and the HMAC system must be again initialized.

In the proposed implementation each hmac value is computed after 129 clock cycles (64 for each one of the two SHA-256 cores and one clock cycle for the intermediate padding-register).

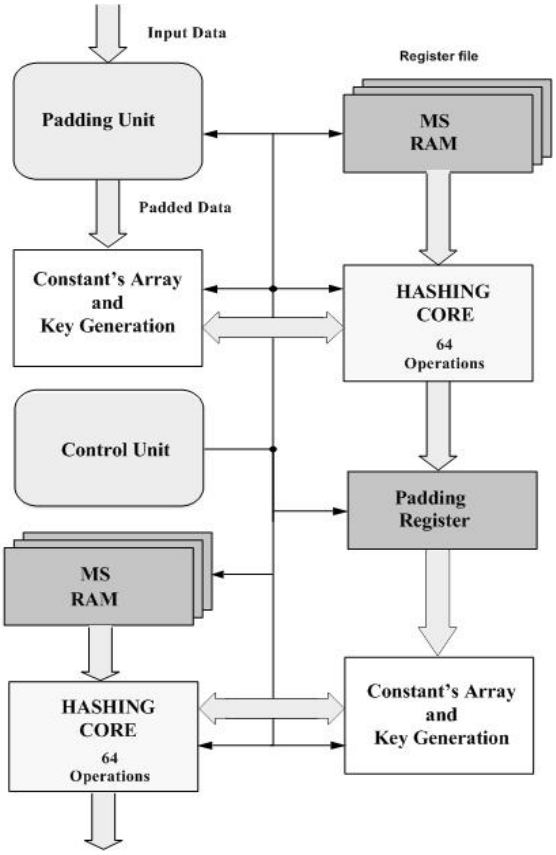

Fig. 4: Typical HMAC core architecture

The messages that are supplied in the HMAC implementation are usually the 256-bit hash value of the whole text that is intended to be digitally signed. This is why the messages, supplied in the HMAC implementation, are assumed as single-block (up to 512-bit length) which enables us to use the pipeline technique.

The method of intermediate storing the values that have arise from the processing of the two keys at the SHA-256 cores saves the time of processing two 512-bit blocks for every message and also allows pipeline technique to be applied, increasing this way both speed and throughput of the HMAC implementation. These stored intermediate values shall be treated and protected in the same way as the secret keys.

The decreasing of the operating frequency of the SHA-256 core results to a lower level of dynamic power dissipation for the whole HMAC core. This can easily be seen regarding the relevant power equations. Moreover the adopted methodology for the implementation of each SHA-256 operation

block combines the execution of two logical SHA-256 operations in only one single clock cycle. This means that the final hmac value is computed in only 65 clock cycles (whereas 129 with conventional pipeline) and thus calls for only 65 write operations in the temporal register that save all the intermediate results until the final hmac value is computed

Moreover in the proposed implementation a 50% higher throughput is achieved comparing to competitive implementations. As a result of this the operating frequency can be reduced about 50% in case we don't need the achieved extra throughput. The reduction of the operating frequency also leads to reduction of the supplying voltage Vdd (in ASIC designs) at about 40 % taking in consideration conservative aspects. On the other hand a significant increase in the effective capacitance of the circuit occurs by a factor of two that has to be taken in consideration. Considering that the power dissipation in a circuit is proportional to the effective capacitance, to the operating frequency and to the square of the supplying voltage, it can be assumed that in this way an extra 60% power saving can be achieved. Thus the proposed implementation is optimized in terms of performance, power dissipation and size.

4 Experimental results

The proposed hashing cores that were presented as examples were captured in VHDL and were fully simulated and verified using the Model Technology's ModelSim Simulator. The designs were fully verified using a large set of test vectors, apart from the test example proposed by the standards.

The achieved operating frequency is equal to 36.1 MHz for the SHA-256 hashing core. Achieving this high frequency, throughput exceeds 2.3 Gbps. In Table 1, the proposed implementation and the implementations of [5], [6],[7] and a conventional pipelined implementation, that was developed for a fair comparison, are compared.

SHA-256	Frequency (MHz)	Throughput (Mbps)
[5]	83.0	326.0
[6]	74.0	291.0
[7]	77.0	606.0
Conv.Impl	50.1	1632.0
Proposed	**36.1**	**2310.4**

Table 1. Throughput Comparison of proposed and other alternatives SHA-256 implementations

From the experimental results, there is more than 40% increase of the throughput compared to the conventional implementation and more than 400% compared to the previously better performing implementation.

The area penalty compared to the non-pipelined implementations is significant (about 20%-30%) but the comparison is unfair both for area and throughput and that is the reason for developing the conventional pipelined implementation of SHA-256.

It has to be mentioned that surprisingly not all the companies that provide cryptographic IP cores support SHA-256. Moreover not many academical proposals have been made for SHA-256 and this is the reason why not many comparisons have been presented in this paper. The reason for this lack on SHA-256 implementations is that nowadays the most widely used has function is SHA-1 in which the security problems that have recently been discovered [2] were announced in February 2005 and not many work has been presented since then. This supports more the motivation to present and propose novel implementations of HMAC mechanism invoking the SHA-256 hash function.

The achieved operating frequency is equal to 34.7 MHz for the whole HMAC construction which corresponds to an increase of more than 50% compared to the implementations of [8], [9],[10] ([9],[10] commercial IP) and to the conventional pipelined implementation that was developed for a fair comparison. However it is important to point out that in [8], [9],[10] the underlying hash function is SHA-1 with the already known security problems.

Achieving this high frequency, throughput exceeds 2.2 Gbps for the HMAC implementation. In Table 2, the proposed implementation and the implementations of [8], [9],[10] and a conventional pipelined implementation are compared.

HMAC	Frequency (MHz)	Throughput (Mbps)
[8]	82.0	328.0
[9]	-	555.0
[10]	-	1500.0
Conv.Impl	47.1	1507.0
Proposed	**34.7**	**2220.8**

Table 2. Throughput Comparison of proposed and other alternatives HMAC implementations

From the experimental results, there is more than 50% increase of the throughput compared to the conventional implementation and that in [10].The introduced area overhead for the HMAC core is

approximately 11.5% compared to the conventional implementation that was implemented by the authors.

Unfortunately for the reasons that we have already mentioned before, so far not many works concerning HMAC with SHA-256 have been presented and thus the comparison is done among only three alternatives and one that has been developed by the authors in order to make fair comparisons.

5 Conclusion

The pre-computation technique has been presented in this paper introducing a pre-computational stage which allows timing transformation of the calculation formulas to generate disjoint intermediate signals and spatial transformation to the pipeline stages. This technique is forming a generic methodology to design high-speed implementations for various families of hash functions.

A high-speed implementation of the SHA-1 hash function and the SHA-256 hash function was developed in this paper applying the pre-computation technique. It is the first known implementation that exceeds the 2.5 Gbps throughput limit (for the XILINX FPGA technology - v150bg352 device) for SHA-1 hash function and the 2 Gbps throughput limit for SHA-256 hash function. From the experimental results, it was proved that SHA-1 proposed implementation was about 40% faster than any previously known implementation whereas SHA-256 proposed implementation was more than 25% faster than the conventional pipelined implementation.

Additionally, the introduced area penalty was approximately 7.5% compared to the nearest performing implementation for SHA-1 and 9.5% for SHA-256 compared to the conventional pipelined implementation. This makes both implementations suitable for every new wireless and mobile communication application that urges for high-performance and small-sized solutions.

Acknowledgments

We thank European Social Fund (ESF), Operational Program for Educational and Vocational Training II (EPEAEK II) and particularly the program PYTHAGORAS, for funding the above work.

References:

[1] FIPS PUB #HMAC, *The Keyd-Hash Message Authentication Code*, National Institute of Standards and Technology, 2001.

[2] X. Wang, Y.L. Yin, H. Yu, Finding collisions in the full SHA1, *Crypto 2005*.

[3] H. Dobbertin, The Status of MD5 After a Recent Attack, *RSALabs' CryptoBytes*, Vol.2, No.2, Summer 1996.

[4] SHA-2 Standard, National Institute of Standards and Technology (NIST), Secure Hash Standard, FIPS PUB 180-2.

[5] N.Sklavos, and O. Koufopavlou, Implementation of the SHA-2 Hash Family Standard Using FPGAs, *Journal of Supercomputing, Kluwer Academic Publishers*, Vol. 31, 2005, pp. 227-248.

[6] N. Sklavos, and O. Koufopavlou, On the Hardware Implementations of the SHA-2(256, 384, 512) Hash Functions, *IEEE International Symposium on Circuits & Systems (ISCAS'03)*, Vol. V, 2003, pp. 153-156.

[7] Helion Technology Ltd. Web page, available at http://www.heliontech.com.

[8] Selimis, G., Sklavos, N., and Koufopavlou, O., VLSI Implementation of the Keyed-Hash Message Authentication Code for the Wireless Application Protocol, *IEEE International Conference on Electronics Circuits and Systems (ICECS'03)*, 2003, pp.24-27.

[9] Interface Masters , Inc. Web page available at http://www.interfacemasters.com/products

[10] SnapGearInternet Security Appliances Ltd, Web page available at http://www.cankey.com.cn