Proceedings of the 5th WSEAS Int. Conf. on MULTIMEDIA, INTERNET AND VIDEO TECHNOLOGIES, Corfu, Greece, August 17-19, 2005 (pp136-141)

# Parallelism Technique for Speeded-Up and Low-Powered Cryptographic Primitives

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

*Abstract:* - The main applications of the hash functions are met in the fields of communication integrity and signature authentication. A hash function is utilized in the security layer of every communication protocol. However, as protocols evolve and new high-performance applications appear, the throughput of most hash functions seems to reach to a limit. Furthermore, due to the tendency of the market to minimize devices' size and increase their autonomy to make them portable, power issues have also to be considered. In this work a new technique is presented for increasing frequency and throughput of all widely used hash functions – and those that will be used in the future- hash functions such as MD-5, SHA-1, RIPEMD (all versions), SHA-256, SHA-384, and SHA-512 etc. Comparing to conventional pipelined implementations of hash functions the proposed parallelism technique leads to a 33%- 50% higher throughput.

*Key-Words:* - Security, Hash functions, SHA-1, SHA-256, Parallel Computations, High-Throughput, Hardware Implementation

## 1 Introduction

Nowadays many applications like the Public Key Infrastracture (PKI), IPSec, Secure Electronic Transactions (SET), and the 802.16 standard for Local and Metropolitan Area Networks incorporate authenticating services. All these applications pre-suppose that an authenticating module that includes a hash function is nested in the implementation of the application. Hash functions are also required for authentication to Virtual Private Networks (VPN's) that companies are establishing in order to exploit on-line collaboration. Moreover 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 are based on using a critical cryptographic primitive like hash functions.

From all the above it is quite clear that all applications that incorporate hash functions are addressing to more users-clients and thus it is a prior necessity the increase of their throughput particularly for the corresponding server of these services. This is because the cryptographic system, especially the server, has to reach the highest degree of throughput in order to satisfy immediately all requests for service from all users-clients. In many of these cryptographic schemes the throughput of the incorporated hash functions determines the throughput of the whole security scheme. The high-speed of the hash functions calculation is strongly related to the streamlined communication of the two subscribers of the latter mentioned applications. Especially in these applications that transmission and reception rates are high, any latency or delay on calculating the digital signature of the data packet leads to degradation of the network's quality of service.

The latter mentioned facts were strong motivation to propose a novel methodology for hardware implementation applicable to almost all kind of hash functions. The proposed implementation introduces a negligible area penalty; increasing the throughput and keeping the area small enough as required by most portable communication devices. Moreover the proposed technique as it will be shown leads to low-power implementations.

This technique is applicable to a wide range of hash function such as MD-5 [1], SHA-1 [2], RIPEMD that are currently widely deployed and even in SHA-256 [3], SHA-384 [3] and SHA-512 [3] that are going to be used in the near and upper future because of the security problems that have recently been discovered in both SHA-1 [4] and MD-5 [5]. To put into practice the presented technique two certain hash functions have been

used: the SHA-1 hash function representing the now-used hash functions and SHA-256 representing the hash functions that are going to be used in the future replacing the ones that are currently used because of their security problems.

The rest of this paper is organized as follows. In section 2 the proposed implementation is presented in depth, providing details regarding the architecture, the logic and the modifications to increase throughput. In section 3 examples of implemented hash functions in FPGA technology are compared to other implementations. Finally in section 4 the paper concludes.

## 2 Proposed Implementation

Hash functions are iterative algorithms that produce a message digest after a number of similar operations i.e. 64 operations for MD-5, SHA-256 and 80 operations for SHA-1, SHA-384, SHA-512 etc. An approach that increases significantly throughput is the application of pipeline that 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.

In Fig. 1, the general architecture for all hash cores with pipelined structure is illustrated, where there are four pipeline stages and a single operation block for each round among with the rest necessary parts.

The critical path of the illustrated architecture is located between the pipeline stages. The other units, 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. Thus, optimization of the critical path is solely focused on the operation block.

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.

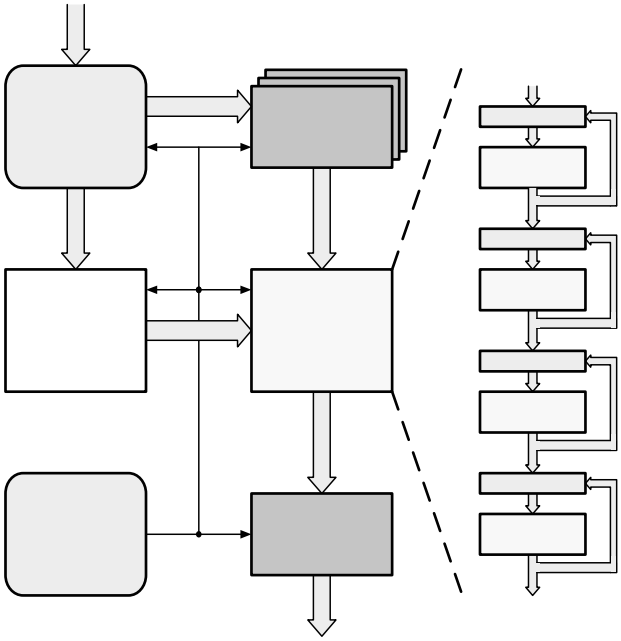

**Fig. 1: Typical hash core architecture with 4 pipeline stages including a single operation block**

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. Making some computations in parallel (parallelism) all implementations that invoke the proposed technique will produce a message digest in the half time meaning in 10 clock cycles instead of 20 for SHA-1 hash function and in 8 clock cycles instead of 16 for SHA-256 hash function

### 2.1 Modifying operation block

The critical path of the illustrated architecture in Fig. 1 is located between the pipeline stages and this is where the parallelism technique is going to be applied. This way the critical path will be increased but in one clock cycle the result of two operations

will have been computed. The proposed technique is applicable to all iterative algorithms and especially for hash functions that are examined in this paper.

The proposed technique is based on a special property of the iterative operation blocks. Only some outputs of the operation block determine the critical paths whereas all the rest outputs arise much sooner. So we can unify two consecutive operations in one where the outputs that arise sooner will take place in computations that are needed for estimating the output values of the next operation. Thus computations needed in two consecutive operations are being performed in parallel. These concurrently computed values are then used in order to estimate the output values of the merged operation block which in one clock cycles computes the values that they would call for data processing in two clock cycles with the conventional implementations. However the computations in the merged operation block take significant less time that the computation of two single conventional operation block and here is our profit. In this way the new (merged) operation block has a 25%-33% increase of the critical path but in a single clock cycle the results of two operations arise.

Examining the throughput equation in (1) the new operating frequency is about 66%-75% of the $f_{operation}$ that conventional implementations are achieving, but the *#operations* are exactly the half comparing to the *#operations* of conventional implementations. This means that finally the achieved throughput of the hashing cores are increased by 33%-50% theoretically.

Moreover this technique leads to low-power implementations since the decrease of the operating frequency of the hashing core results to lower dynamic power dissipation for the whole core. What 's more the fact that the hash value is computed in half clock cycles means that we save the half power from read and writes to registers that save intermediate results.

Let's consider the example of SHA-1 hash function and how the parallelism technique is applied. Unfolding the expressions of $a_t$, $b_t$, $c_t$, $d_t$, $e_t$, as they described in [4], it is observed that $a_{t-1}$, $b_{t-1}$, $c_{t-1}$, $d_{t-1}$ values are assigned directly to outputs $b_t$, $c_t$, $d_t$, $e_t$ respectively. In Eq. (2) the expressions of $a_t$, $b_t$, $c_t$, $d_t$, $e_t$, are defined.

$$\begin{aligned} e_t &= d_{t-1} \\ d_t &= c_{t-1} \\ c_t &= \mathrm{ROTL}_{30}(b_{t-1}) \\ b_t &= a_{t-1} \\ a_t &= \mathrm{ROTL}_5(a_{t-1}) + f_t(b_{t-1},c_{t-1},d_{t-1}) + e_{t-1} + W_t \\ &\quad + K_t \end{aligned} \qquad (2)$$

where $\mathrm{ROT}_x(y)$ represents rotation of word $y$ to the left by $x$ bits and $f_t(z,w,v)$ represents the non-linear function associated to the round.
According to Eq. (2) two consecutive operation blocks of the SHA-1 hash function are represented in Fig.2.

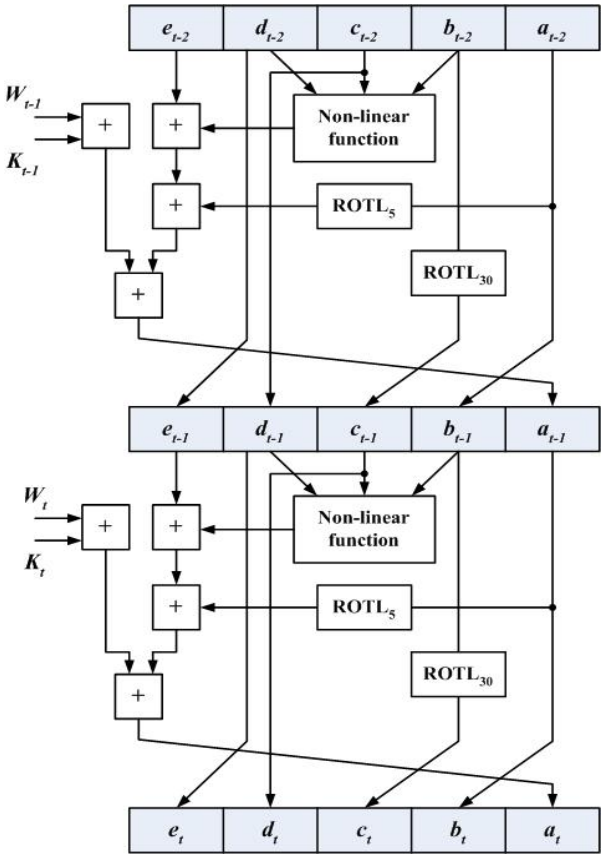


**Fig. 2: Two consecutive SHA-1 operation blocks**

The proposed design approach is based on the basic concept that was previously mentioned. At the first operation block of Fig.2 except of the output $a_{t-1}$, the rest of the outputs $b_{t-1}$, $c_{t-1}$, $d_{t-1}$ and $e_{t-1}$ are derived directly from the inputs $a_{t-2}$, $b_{t-2}$, $c_{t-2}$, and $d_{t-2}$ respectively. This means consequently that also $c_t$, $d_t$ and $e_t$ can be derived directly from $a_{t-2}$, $b_{t-2}$ and $c_{t-2}$ respectively. Furthermore, due to the fact that $a_t$ and $b_t$ calculations require the $d_{t-2}$ and $e_{t-2}$ inputs respectively, which are stored in temporal registers, these calculations can be performed in parallel. In Fig. 3, the consecutive SHA-1 operation blocks of Fig. 2, have been modified so that $a_t$ and $b_t$ are calculated in parallel. The gray marked areas on Fig. 3 indicate the parts of the proposed SHA-1 operation block that operate in parallel. Estimating the critical path in Fig.3 we notice that only a single addition level has been introduced to the critical path. Although, this reduces the maximum operation frequency, the throughput is increased significantly

since the message digest is now computed in only 40 clock cycles instead of 80 comparing to the conventional implementations.

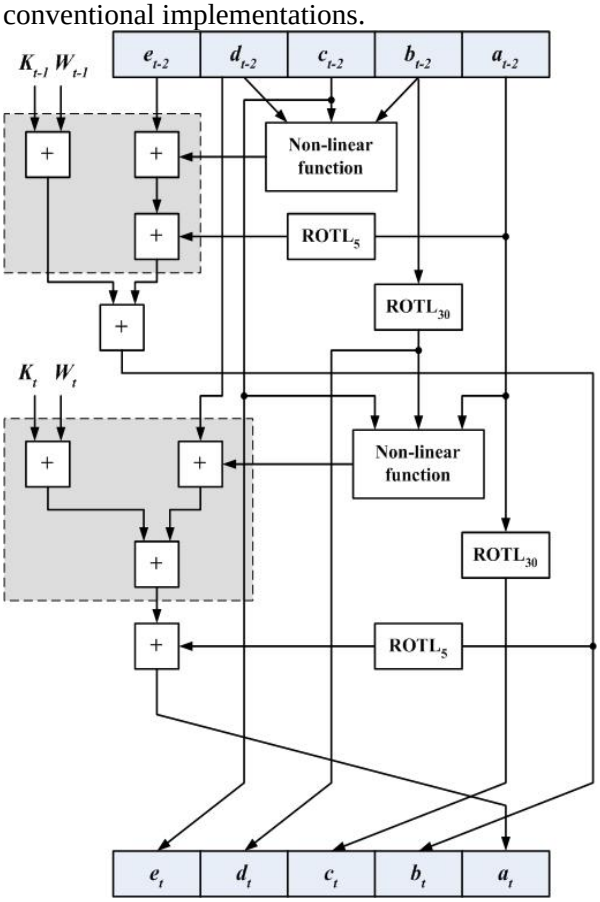


**Fig. 3: Two merged SHA-1 operation blocks**

The modified expressions that give $a_t$, $b_t$, $c_t$, $d_t$ and $e_t$, are now described from Eq.3.

$$
\begin{aligned}
e_t &= c_{t-2} \\
d_t &= \mathrm{ROTL}_{30}(b_{t-2}) \\
c_t &= \mathrm{ROTL}_{30}(a_{t-2}) \\
b_t &= \mathrm{ROTL}_5(a_{t-2}) + f_t(b_{t-2},c_{t-2},d_{t-2}) + e_{t-2} + \\
&\quad W_{t-1} + K_{t-1} \\
a_t &= \mathrm{ROTL}_5(b_t) + f_t(a_{t-2},c_{t-2}, \mathrm{ROTL}_5(b_{t-2})) + \\
&\quad + d_{t-2} + W_t + K_t
\end{aligned}
\qquad (3)
$$

From Eq. 3, it can be assumed that the area requirements are increased significantly. Thus, the small-sized constraint is violated. However this is not true since the hardware to implement the operation blocks of the SHA-1 rounds is only a small percentage of the SHA-1 core. Using the proposed SHA-1 operation block to each round, the temporal register write operations are reduced by 50% (two operations per cycle, hence 40 cycles to generate the hash value). Furthermore, the operating frequency is decreased resulting in reduction of the dynamic power dissipation per operation. Thus, the proposed implementation satisfies every design constraint for small-sized, low-power and high-performing operation.

Let's consider now another example, the SHA-256 hash function and how the parallelism technique is applied.

We assume two consecutive operations of the SHA-2 hash function which are illustrated in Fig 4. 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.

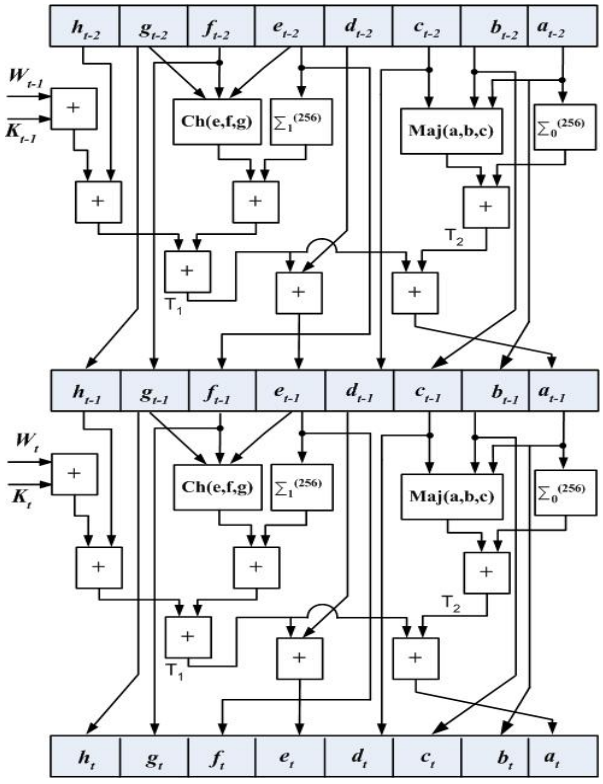


**Fig. 4: Two consecutive SHA-256 operation blocks**

Moreover 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. 5, 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. 5 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.

Inspecting Fig. 4 and Fig. 5 it is obvious that the critical path in the proposed implementation consists of six addition levels instead of the four addition levels that exist in 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 computation lead to the result that theoretically the throughput of the proposed implementation increases 33%.The experimental results verify this theoretical assumption.

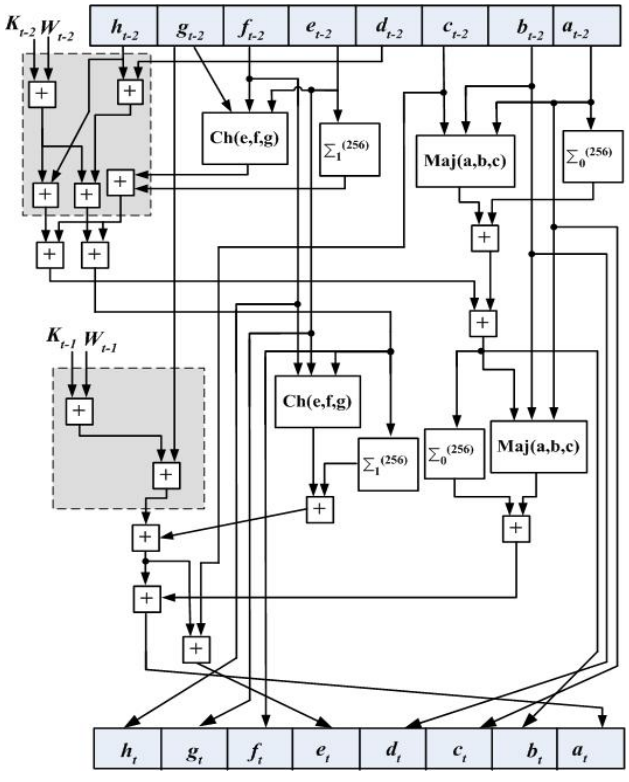


**Fig. 5: Two merged SHA-256 operation blocks**

The parallelism technique can also be applied in other hash functions such as MD-5, RIPEMD, SHA-384, SHA-512 etc with the same way, leading to similar results.

The introduced area penalty is about 10% for the whole security scheme that is included in a corresponding application. This area penalty is worth paying for an increase of throughput at about 33%-50%.

# 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.

The achieved operating frequency is equal to 55.7 MHz for SHA-1. Achieving this frequency, throughput exceeds 2.8 Gbps. In Table 1, the proposed implementation and the implementations of [6], [7], [8], [9], [10] and [11] are compared. From the experimental results, there is about 50% increase of the throughput compared to the previously better performing implementation in [8].

| SHA-1 | Frequency (MHz) | Throughput (Mbps) |
|---|---|---|
| **[6]** | 43.0 | 119.0 |
| **[7]** | 72.0 | 1842.2 |
| **[8]** | 72.0 | 460.8 |
| **[9]** | 55.0 | 1339.0 |
| **[10]** | 38.6 | 900.0 |
| **[11]** | 86.0 | 530.0 |
| **Proposed** | **55.7** | **2816.7** |

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

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 3, the proposed implementation and the implementations of [12], [13],[14] and a conventional pipelined implementation, that was developed for a fair comparison, are compared.

| SHA-256 | Frequency (MHz) | Throughput (Mbps) |
|---|---|---|
| **[12]** | 83.0 | 326.0 |
| **[13]** | 74.0 | 291.0 |
| **[14]** | 77.0 | 606.0 |
| **Conv.Impl** | 50.1 | 1632.0 |
| **Proposed** | **36.1** | **2310.1** |

**Table 3. Throughput Comparison of proposed and other alternatives SHA-256 implementations**

From the experimental results, there is more than 33% increase of the throughput compared to the conventional implementation and more than 330%

compared to the previously better performing implementation.

The area penalty compared to the non-pipelined implementations of [12], [13] is much more 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.

## 5 Conclusion

The parallelism technique has been presented in this paper indicate parts of the operation block that can operate in parallel and result to the concurrent computation of the primary operation block's outputs. 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 2.8 Gbps throughput limit (for the XILINX FPGA technology - v150bg352 device) for SHA-1 hash function and the 2.3 Gbps throughput limit for SHA-256 hash function. From the experimental results, it was proved that SHA-1 proposed implementation was about 50% faster than any previously known implementation whereas SHA-256 proposed implementation was more than 35% faster than the conventional pipelined implementation.

Additionally, the introduced area penalty was negligible while all implementations are considered as low-power. This makes both implementations suitable for every new wireless and mobile communication application that urges for high-performance, low-power 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] R. L. Rivest, The MD5 Message digest Algorithm, *IETF Network Working Group , RFC 1321,* April 1992

[2] FIPS 180-1, Secure Hash Standard. Federal Information Processing Standard, *(FIPS), Publication 180-1, NIST, US Dept of Commerce*, April 1995.

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

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

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

[6] S. Dominikus, A Hardware Implementation of MD4-Family Hash Algorithms, *IEEE International Conference on Electronics Circuits and Systems (ICECS'02)*, Dubrovnik, Croatia, September 15-18, 2002.

[7] Sklavos, N., Alexopoulos, E., and Koufopavlou, O., Networking Data Integrity: High Speed Architectures and Hardware Implementations, *IAJIT Journal*, Vol.1, No.0, 2003, pp.54-59.

[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] Sklavos, N., Dimitroulakos, G., and Koufopavlou, O., An Ultra High Speed Architecture for VLSI Implementation of Hash Functions, *IEEE International Conference on Electronics Circuits and Systems (ICECS'03)*, 2003, pp.990-993.

[10] Diez, J.M., Bojanic, S., Carreras, and Nieto-Taladriz, O., Hash Algorithms for Cryptographic Protocols: FPGA Implementations, *TELEFOR*, 2002.

[11] T.Grembowski, R.Lien, K.Gaj, N.Nguyen, P.Bellows, J.Flidr, T.Lehman and B.Schott, Comparative analysis of the Hardware implementationsof hash functions sha-1 and sha-512.In *A.H Chan and V.Gligor, eds, Information Security Conference*, Springer-Verlag, pp 75-89,2002

[12] 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.

[13] 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.

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