A High-Speed Unified Hardware Architecture for the AES and SHA-3 Candidate Grøstl

Marcin Rogawski and Kris Gaj
Volgenau School of Engineering
George Mason University
Fairfax, Virginia 22030
email: {mrogawsk, kgaj}@gmu.edu

***Abstract*—The NIST competition for developing the new cryptographic hash algorithm SHA-3 is currently in the third round. One of the five reminding candidates, namely Grøstl, is inspired by the Advanced Encryption Standard. This unique feature can be exploited in a large variety of practical solutions. In order to have a better picture of the Grøstl-AES computational efficiency (high-level scheduling, internal pipelining, resource sharing etc.), we designed a high-speed coprocessor for Grøstl-based HMAC and the AES in counter mode. It offers high-speed computations of both authentication and encryption with relatively small penalty in terms of area cost and speed reduction when compared to the authentication (original Grøstl circuitry) functionality only. From our perspective, the main advantage of Grøstl over other finalists is the fact the small overhead in its hardware architecture naturally accommodates the AES.**

***Index Terms*—SHA-3 competition; hardware architectures; Grøstl; AES; resource sharing; pipelining; scheduling; IPSec.**

I. Introduction

The National Institute of Standards and Technology (NIST) is currently holding a hash competition [1] to select a new cryptographic hash function, called SHA-3, for the purpose of superseding the aging functions in the SHA-2 family. Performance in hardware has been one of the major factors taken into account by NIST in the evaluation of Round 2 and Round 3 candidates during the SHA-3 competition [1], [2], [3]. This factor is particularly important in the final round of the contest, because the algorithms qualified to this round are not very likely to have any significant security weaknesses.

Several studies regarding stand-alone implementations of Round 2 and Round 3 SHA-3 candidates in FPGAs have been already reported in the literature [2]. The main objective of most published literature is to evaluate all candidates in unified approach and therefore the unique features of each and every function were not deeply investigated.

There are relatively few works which discuss distinctive hardware architectures for the SHA-3 candidates. A coprocessor supporting Skein in tree hashing mode was presented in [4]. Common architectures of the block cipher AES and the Round 2 specifications of Grøstl-0 and Fugue algorithms were reported in [5]. A compact implementation of block cipher Threefish and the Round 3 hash algorithm Skein was demonstrated in [6].

In this effort we are going to present a new hardware architecture for Grøstl and AES working in an interleaved-pipelined fashion. A practical application to IPSec hardware acceleration will be discussed.

The rest of this paper is organized as follows: In section II we discuss previous work. Section III is devoted to the analysis of the Grøstl-AES structure for the authenticated encryption based on HMAC and counter mode, respectively. Section IV describes the proposed coprocessor. Finally, section V discusses and analyzes the results and we draw conclusions in section VI.

II. Previous work

A. Grøstl hardware implementation

In January 2011, Grøstl team published tweaks to their specification of Grøstl [7], [8]. An algorithm described by the original Grøstl specification [9] has been renamed to Grøstl-0, and the tweaked version of Grøstl, described by the revised specification [8], is from this point-on called Grøstl. The proposed tweaks are aimed primarily at the increase in the algorithm resistance to cryptanalysis [7]. This increased resistance in security, typically comes together with some limited penalty in terms of performance in hardware [10].

Grøstl-0 has been implemented by several groups in FPGAs and ASICs [2]. In this paper, we focus on implementations targeting FPGAs and optimized for high speed rather than low area. High-speed implementations of Grøstl-0 typically use two major architectures. In the first architecture, reported first in [9], permutations P and Q are implemented using two independent units, working in parallel. We call this architecture parallel architecture. In the second architecture, introduced in [11], the same unit is used to implement both P and Q. This unit is composed of two pipeline stages that allow interleaving computations belonging to permutations P and Q. We call this architecture quasi-pipeline architecture, as it is based on the similar principles as the quasi-pipelined architectures of SHA-1 and SHA-2 reported in [12], [13]. The details of the quasi-pipelined architecture of Grøstl-0 are described in [11](Section 9), [14](Section 3.8) and [15](Section V).

An analysis of the influence of the Round 3 tweaks in Grøstl on the performance of this algorithm in FPGAs was conducted

[1]This work has been supported in part by NIST through the Recovery Act Measurement Science and Engineering Research Grant Program, under contract no. 60NANB10D004

in [10]. Comprehensive hardware evaluation across multiple architectures for all SHA-3 finalists, including Grøstl, was investigated in [16]. The implementation results of hardware architectures, for a single stream of messages, in both variants of Grøstl are summarized in Table I.

B. Sharing resources

The idea of hardware resources sharing is very practical and especially attractive in industrial applications. Several companies offer so called all-in-one cryptographic solutions. For example [24] and [25] offer customized cores including sophisticated AES core, which supports 128, 192 and 256 bits main key and several different operational modes in a single chip. The resource sharing concept was also investigated by academia: shared MD5 and SHA-1 implementation was described in [26]–[28], MD5 implemented together with RIPEMD-160 was reported in [29], and finally, SHA-1, MD-5 and RIPEMD-160, implemented together, were discussed in [30]. It seems that even a more practical is the idea to build a coprocessor, which could share resources and support different cryptographic services: confidentiality and integrity. Cryptographic accelerators, in which the datapaths are combined: Fugue with AES core, and Grøstl-0 with AES core, were reported in [5] . A typical application for such coprocessor will be the IPSec protocol suite [31] for securing the Internet Protocol, the basis of Internet. This suite consists of the Authentication Header protocol (providing integrity only) and Encapsulating Security Payload (providing integrity and confidentiality at the same time).

III. Authenticated encryption based on Grøstl and AES in a single coprocessor

The specifications of the block cipher AES and the hash function Grøstl were described in [32] and [8], respectively. The round functions for both algorithms are summarized in Figure 1.

The design described in [16] and the corresponding source codes from [33] will serve in this work as a starting point for our investigations.

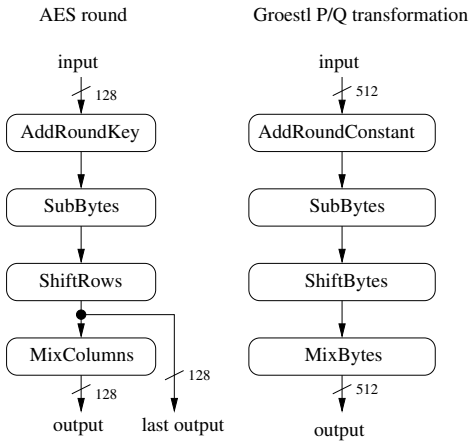


Fig. 1. Block diagram of Grøsl and AES round

A. Grøstl and the AES comparison

TABLE II
Number of rounds and the security level relations for Grøstl and the AES

Security	Grøstl	AES
128-bits	(Grøstl-256) 10	(AES-128) 10
192-bits	(Grøstl-384) 12	(AES-192) 12
256-bits	(Grøstl-512) 14	(AES-256) 14

In order to extend the original Grøstl hardware architecture several facts have to be taken into consideration:

- **The basic round structures** of both algorithms are demonstrated on Figure 1. All four corresponding transformations have the same order in both AES and Grøstl. Due to this fact a resource sharing between both algorithms is especially attractive. It is expected that the delay in the critical path in both cases should be very similar.
- **The SubBytes layers** in both cases are build upon the same substitution box (S-box), therefore it can be fully shared (Figure: 5, pt. 1). In terms of a circuitry area this transformation is the most costly out of all round-building operations.
- **The ShiftRow and ShiftBytes transformations** in the AES and Grøstl, respectively, can be implemented as a permutation of bytes order (simple rewiring). However they are not similar, both operations have to be implemented separately and properly multiplexed (Figure: 5, pt. 2).
- **The AddKey and the AddConstants transformations** in the AES and Grøstl, respectively, can be implemented as a simple network of XOR gates. However they are not similar, both operations have to be implemented separately and properly multiplexed (Figure: 5, pt. 3).
- **The MixColumn and the MixBytes** in the AES and Grøstl, respectively, shares the GF(2^8) multiplication by constants: 0x02 and 0x03. Therefore they can be completely merged together. The networks of output XORs require two separate paths for both algorithms (Figure: 5, pt. 4).
- **The last round** of the AES block cipher is different than the regular round. It is required to build a bypass bus and multiplex it together with round's regular output (Figure: 5, pt. 5).
- For a given security level both Grøstl and the AES require **the same number of rounds**. This dependency is summarized in Table II. This fact help to achieve a full synchronization of input data for both HMAC and Encryption module.
- **The Grøstl double data flow pipe (P and Q transformations) vs. the AES one data flow pipe** determine the optimal number of pipeline stages. The high-speed single stream of data quasi-pipelined hardware architecture of Grøstl, demonstrated in [14], [15], [11], requires two pipeline stages for the P and Q permutations intermediate values. The third pipeline stage is required for the AES

TABLE I
RESULTS OF IMPLEMENTATIONS FOR HIGH-SPEED ARCHITECTURES OF GRØSTL-256, USING XILINX VIRTEX 5 FPGAS.

Source	Architecture	Implementation details	Memory	Frequency	Throughput	Area	Throughput/Area
			[BRAM]	[MHz]	[Mbps]	[Slice]	[Mbps/Slice]
Grøstl-0 - Round 2							
Gauravaram et al. [9]	parallel	N/A*)	N/A*)	200.7	10276	1722	5.97
Jungk et al. [15]	quasi-pipelined	S-boxes in BRAM	17	295.0	7552	1381	5.46
Shahid et al. [17]	quasi-pipelined	T-boxes in BRAM	48	250	6098	1188	5.13
Homsirikamol et al. [14]	quasi-pipelined	64-bits interface	0	323.4	7885	1597	4.94
Gaj et al. [18]	quasi-pipelined	64-bits interface	0	355.9	8676	1884	4.61
Matsuo et al. [19]	parallel	S-boxes in distributed memory	0	154.0	7885	2616	3.01
Baldwin et al. [20]	parallel	ideal interface, no padding unit	0	101.3	5187	2391	2.17
Kobayashi et al. [21]	parallel	S-boxes decomposed into logic	0	101.0	5171	4057	1.27
Guo et al. [22]	parallel	S-box decomposed into logic	0	80.2	4106	3308	1.24
Baldwin et al. [20]	parallel	32-bits interface, no padding unit	0	101.3	3242	2391	1.36
Baldwin et al. [20]	parallel	32-bits interface, padding unit	0	78.1	2498	2579	0.97
Grøstl - Round 3							
Sharif et al. [23]	quasi-pipelined	S-box in BRAM	18	226	5524	1141	4.84
Homsirikamol et al. [16]	quasi-pipelined	64-bits interface	0	249	6072	1912	3.18
Homsirikamol et al. [16]	parallel	64-bits interface	0	158	8081	2591	3.12

*) not reported

intermediate data (Figure: 5, pt. 6).

- **Both algorithms input block sizes differ**. They are 128-bits and 512-bits for the AES and Grøstl, respectively. The encryption of 512-bits single stream of data, by four instances of algorithm which can accommodate 128-bits input only, prohibits the feedback mode utilization. In order to increase the security level of non-feedback mode based encryption the counter mode (Figure: 5, pt. 7) was applied (Figure: 3).
- The encryption process requires **an extra storage space** for the plain/cipher text (Figure: 5, pt. 8).
- For a given security level **the output block** of both algorithms is different. This fact implies the size extension (doubled) of the Parallel Input Serial Output (PISO) module (Figure: 5, pt. 9).
- **The Key scheduling algorithm** for the AES algorithm requires an additional circuitry (Figure: 5, pt. 10).
- **Second hashing in the HMAC** requires message padding (Figure: 5, pt. 11).

Motivated by above observations, we will show how to efficiently share the resources of Grøstl and AES in our coprocessor for an authenticated encryption.

B. HMAC/Grøstl

A mechanism for message authentication using cryptographic hash functions, the HMAC (Hash-based Message Authentication Code) was originally defined in [34]. Recently this document was updated by [35]. HMAC has a generic form and it can be used with any iterative cryptographic hash function, e.g. Grøstl, in combination with a secret shared key. The HMAC cryptographic strength rely on the properties of the underlying hash algorithm. Figure 2 demonstrates the HMAC generation process. Since the combination of HMAC with a current standard SHA-2 is denoted as HMAC/SHA-2, we are using corresponding notation for Grøstl algorithm (HMAC/Grøstl).

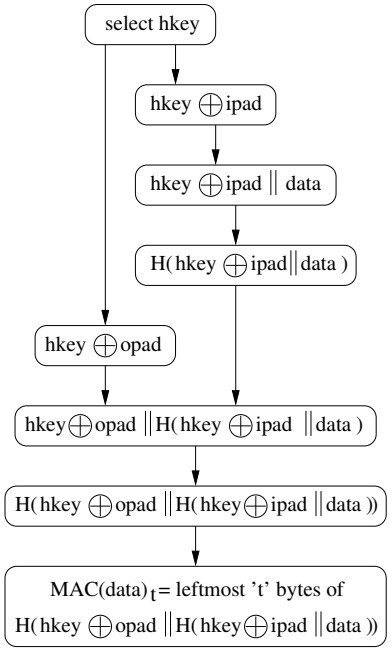


Fig. 2. HMAC generation

In order to compute the HMAC value for a given message (data) and a key (hkey) the selected hash function has to be used twice. The output from the first computations is a function of a ($ipad$ constant) padded key and a given message. The output from the second computations (the hmac-value) is a function of a ($opad$ constant) padded key and the result of first computations. For the sake of simplification of our circuit (padding of the second hash computations) we restricted the range of key size up to the Grøstl block size.

This assumption leads us to the relation between the throughput of HMAC/Grøstl and the throughput of Grøstl:

$$\frac{throughput_{HMAC/Grøstl}}{throughput_{Grøstl}} = \frac{\#blocks}{5 + \#blocks} \quad \text{(III.1)}$$

where:

$\#blocks$ is the number of data chunks for a given message and $throughput_{Grøstl}$ is the maximum Grøstl hardware architecture throughput calculated for long messages.

The constant in the denominator is an overhead from HMAC/Grøstl and it is a sum of

- two HMAC key injections,
- two Grøstl message finalizations,
- and an injection of a message digest from the first to the second hash computation.

In case of long messages the effect of HMAC/Grøstl overhead is marginal, and it can be omitted in the throughput calculations.

C. AES in Counter mode

NIST has defined five confidentiality modes of operation for use with an underlying symmetric key block cipher algorithm: Electronic Codebook (ECB), Cipher Block Chaining (CBC), Cipher Feedback (CFB), Output Feedback (OFB), and Counter (CTR) in [36]. Two of aforementioned modes of operation, namely ECB and CTR, allow parallel computations. In ECB mode, for a given key any given plaintext block encryption process always leads to the same ciphertext block. This property is undesirable in predominant number of applications and due to this fact the ECB mode should not be used.

The CTR mode for a block cipher is presented on Figure 3.

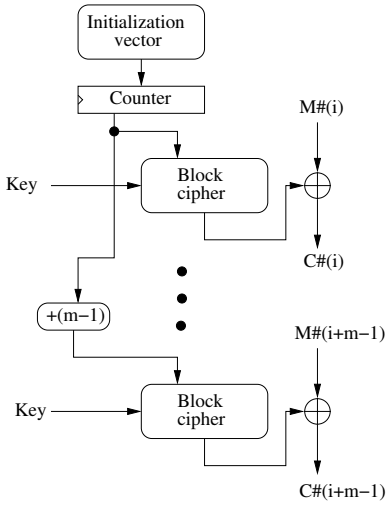


Fig. 3. Block diagram of counter mode in block ciphers

To encrypt using AES/CTR-mode encryption, one starts with an arbitrary bit string(a n-block plaintext), a session key, and an init value for a 128-bits (block size) counter. The output cipher text $C = \{C\#1, C\#2, ...C\#n\}$ is the XOR of corresponding plain text chunks (on Figure 3 the data blocks are represented as $D = \{D\#(1),\ D\#(2)\ ...D\#(n)\}$ and the results of encryption of $E_{key}(ctr + 1)$, $E_{key}(ctr + 2)$ ..., $E_{key}(ctr + n)$. The cipher text is a pair (IV, C), where IV is the starting value for the counter. The decryption process is the same as encryption with D and C interchanged.

The biggest advantage of the CTR-mode for any block cipher, including the AES, is the possibility of a full parallelization of the computations. In order to compute all data chunks: $C\#(i + 1)$, $C\#(i + 2)$, ..., $C\#(i + m)$ we can instantiate m AES coprocessors working simultaneously.

Since Grøstl specifies 512-bits (128-bits security level) and 1024-bits (256-bits security level) input block sizes then the number of corresponding CTR/AES cores is four and eight, respectively. The maximum throughput in such configuration is four (eight in case of Grøstl with 1024-bits input block) times higher than the throughput of the single AES core.

IV. Coprocessor description

A. Block diagram description

A block diagram presented on Figure 5 shows the datapath used in the proposed Grøstl/AES coprocessor. The non-filled components represent the original Grøstl design, available in [33]. The original Grøstl quasi-pipelined structure has one pipeline register inserted between SubBytes and ShiftBytes operations.

In order to perform in parallel encryption and message digest computation the quasi-pipeline architecture was enriched by several extra elements. The filled components show which elements have to be added in order to accommodate the HMAC/Grøstl and the AES-CTR functionality.

First of all, we have added additional pipeline register after the Shared MixColumn/MixBytes operation. Two of pipeline stages contain intermediate values for the P and Q functions from Grøstl, one extra stage is responsible for the encryption intermediate values of the same chunk of data.

B. Grøstl and AES pipelining

In the very first clock cycle, an input message is loaded directly to the state register as an input to the operation Q. A message block is xored with an initialized chain register to create an input for the operation P in the second cycle of processing. Finally, in the third clock cycle the counter values are loaded to the state register. At the same time when the first stage of the pipeline starts executing the first phase of AES round, the second stage of the pipeline continues the execution of the P operation and the third stage is in the last phase of Q operation.

The first stage of the pipeline consists of the Grøstl's P/Q AddRoundConstants, the AES AddRoundKey units and the fully shared SubBytes layer (Figure: 6).

The second stage of the pipeline consists of the ShiftBytes/ShiftRows and modified MixBytes units.

The third stage of the pipeline consist of just two multiplexers.

A part of the function Q is always performed one cycle ahead of the corresponding part of function P and two clock cycles before CTR-mode AES related data.

Fig. 5. Block diagram of Grøstl/AES core

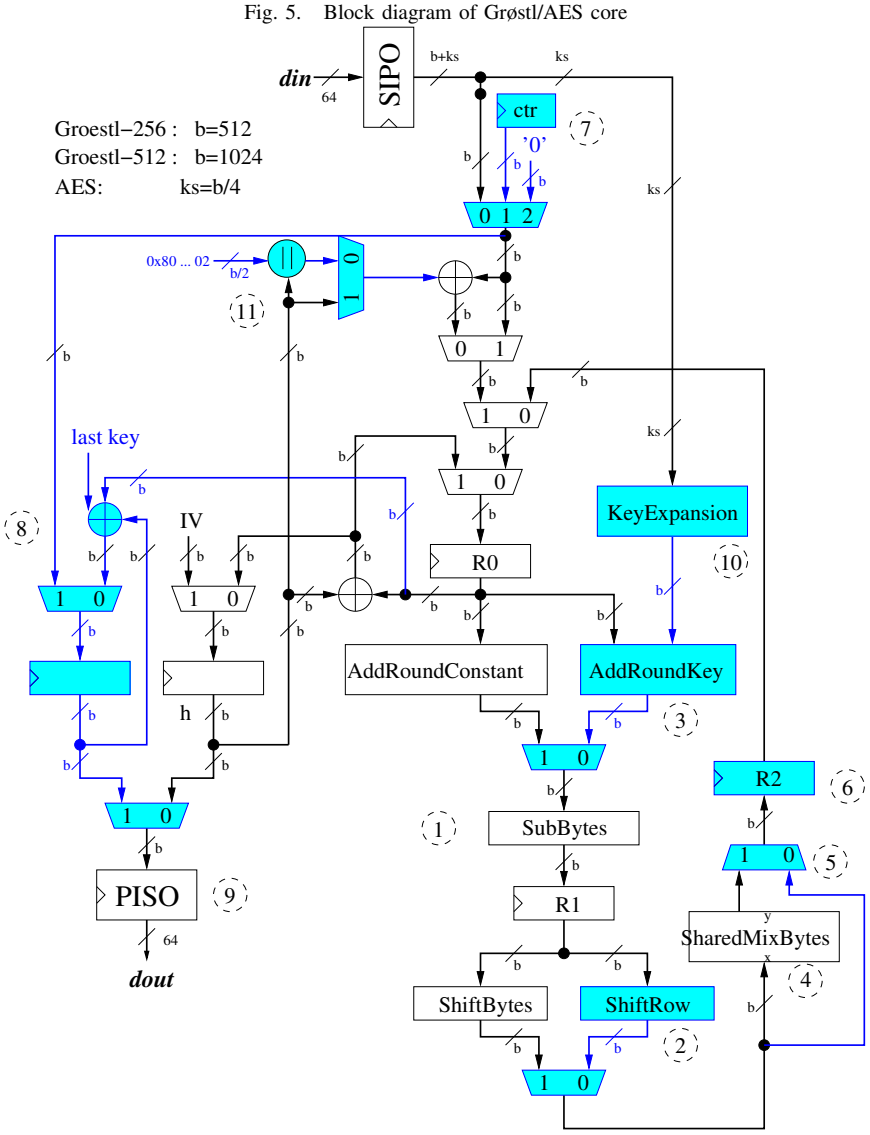
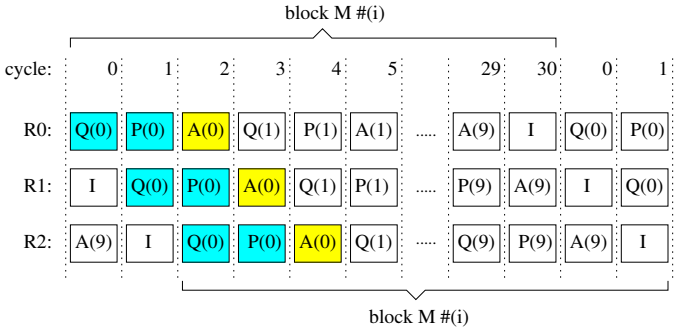


Fig. 4. Pipelining in the Computational Unit of Grøstl/AES core



Finalization of hash process in this design takes two clock cycles. First, the chaining value is xored with the final value of Q, while P is still being processed. In the subsequent cycle the final result of P is mixed with the chaining value as well (Figure:4).

In the following clock cycle the tenth round of the AES transformation is completed. Finally, the last AES key is xored with the output from stage register and with the plain text. Every time when the encryption process is finished the cipher text is ready to be stored in the Parallel Input Serial Output (PISO) unit. The entire process is repeated until all blocks of a message are thoroughly hashed and encrypted.

The HMAC process requires also additional data in front (key xored with the constant $ipad$ value) and at the end (key xored with the constant $opad$ value) of the message. During the time when these pre- ($D\#0$) and post- ($D\#n$) data is processed, the AES module is not producing valid data ($AES(idle)$ on Figure: 6).

Finally, a HMAC value is calculated and it is taken from the bottom half of the chaining value.

For a given chunk of a 512-bits data both Grøstl and

AES cores need 31 clock cycles to complete their operations (3 pipeline stages per 10 rounds + 1 clock for the Grøstl finalization and 1 clock cycle for the final xor in the counter mode).

C. High-level scheduling

In order to make our implementations as practical as possible, we have followed an 64-bits interface and a simple handshaking protocol specification from [33]. Thanks to the assumptions taken from aforementioned paper, it is possible to keep proposed coprocessor's all three pipeline stages busy almost at all the time.

The input-output operations overlap in many cases therefore the separation of input/output bus and control signals is necessary.

A higher level scheduling is summarized on Figure 6. The path of very first chunk of message $D\#1$ for the authenticated encryption is denoted by the filled figures.

During the computations of longer messages (more than three blocks) the coprocessor will be storing result of the $C\#(i-2)$ block, conducting HMAC/Grøstl and CTR/AES operations for the block $C\#(i-1)$ and fetching $i-th$ block of data ($D\#(i)$) at the same time.

D. Throughput discussion

In the most typical scenario the speed of the hardware implementation of cryptographic transformations is understood as a throughput for long messages. The exact throughput formula is defined as follows [18]:

$$throughput = \frac{blocksize}{T * (Time_{HE}(N+1) - Time_{HE}(N))} \quad \text{(IV.1)}$$

where $blocksize$ is a input block size, characteristic for each cryptographic transformation, $Time_{HE}(N)$ is a total number of clock cycles necessary to hash/encrypt an N-block input data and T is the clock period , characteristic for each hardware coprocessor.

In case of the Grøstl/AES-based hardware accelerator, described in this paper the throughput formula for long messages is:

$$throughput = \frac{512}{31 * T} \quad \text{(IV.2)}$$

The typical application for an authenticated encryption-oriented, high-speed hardware coprocessor is the Encapsulating Security Payload (ESP) from the IPSec protocol suite. In this scenario the throughput has to be calculated for relatively short messages (40-1536 bytes).

Due to the fact that the HMAC/Grøstl computations take more time that the CTR/AES encryption, this HMAC/Grøstl throughput is considered as an effective throughput for a given message in our coprocessor. The final throughput formula is a result of both formulas: III.1 and IV.2.

$$throughput = \frac{512 * \#blocks}{(5 + \#blocks) * (31 * T)} \quad \text{(IV.3)}$$

For long messages the formula IV.3 converges to the formula IV.2.

V. Results

The HMAC/Grøstl and CTR/AES based hardware coprocessor was implemented on four high speed FPGA devices: 65nm Altera Stratix III, Xilinx Virtex 5 and 40nm Altera Stratix IV, Xilinx Virtex 6. As our tools, we have used Xilinx ISE 13.1 and Altera Quartus II 11.1. All architectures have been first modeled in VHDL-93, then synthesized, placed and routed using tools of the respective vendor. Maximum clock frequencies have been determined using static timing analysis tools provided as part of the respective software packages ($quartus\_sta$ for Altera and $trace$ for Xilinx). The tool options were selected in such a way, that no embedded resources, such as block memories or DSP units, were used during implementation. This choice was made in order to enable the comparison of all implementations in terms of area and throughput to area ratio. Table III summarizes the results collected after the $Place\text{-}and\text{-}Route$ and $Fitter$ in Xilinx and Altera, respectively.

Generally in terms of area, the coprocessor proposed in this effort can be implemented on the smallest device from every selected family. In case of small messages, the throughput is a function of the message size. For the smallest 40-bytes packages it is just 11% of the long messages throughput, but in case of 1536-bytes messages it reaches almost 83% of long messages throughput.

A. Comparison to the stand-alone Grøstl implementation

In table III we have summarized the implementation results of the proposed Grøstl/AES hardware accelerator.

First of all, in the case of Xilinx Virtex 5 implementation, the coprocessor investigated in this effort requires 31% more area than the basic version of quasi-pipelined architecture presented in [14]. Since this extra pipeline stage refinement breaks the critical path from the aforementioned design the maximum frequency increases by 4.8%. The 3rd stage pipeline register location was investigated by moving it before the multiplexer (Figure: 5, pt. 5). It helps improve the maximum frequency, but at the same time the throughput/area ratio decreased. However due to the fact that the quasi-pipelined hardware architecture of Grøstl from [14] and triple-staged Grøstl/AES in this work require 21 and 31 clock cycles, respectively, the overall throughput for long messages decreases by 29%.

In table III we have presented the impact of IPSec minimum and maximum size messages on the effective throughput. In case of selected FPGA devices it varies between 450-3700Mb/s. The final throughput result depends on the traffic statistics in a given network.

The coprocessor proposed in this work can be easily implemented on the smallest devices available in every selected high-speed family.

Fig. 6. High level scheduling in the Grøstl/AES core

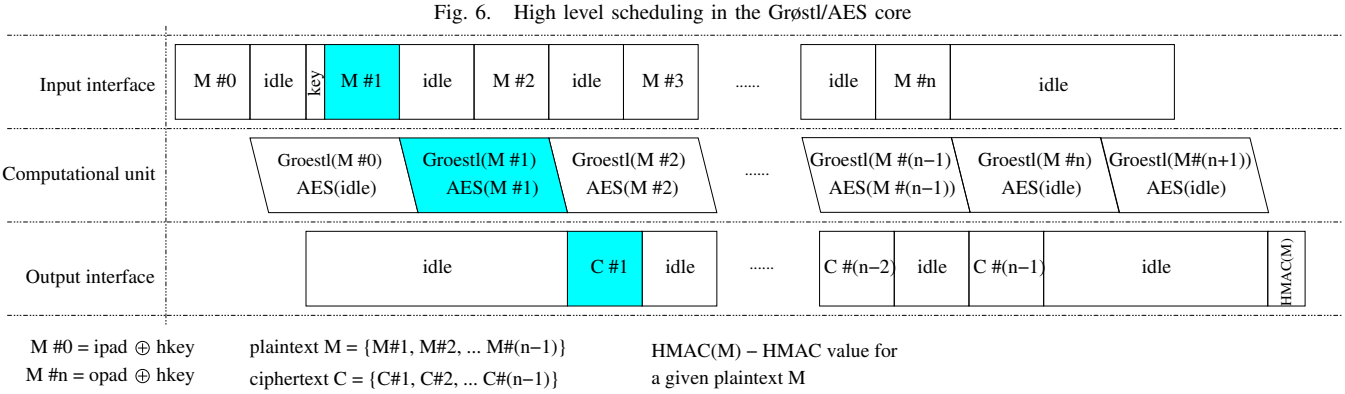


TABLE III
RESULTS OF SHARED-RESOURCES IMPLEMENTATION FOR HMAC-GRØSTL AND AES IN COUNTER MODE ON MODERN FPGA

Family	Frequency	Area	Throughput @40Bytes	Throughput @1536Bytes	Throughput @infinity
Altera					
	[MHz]	[ALUTs, Memory bits]	[Mbps]	[Mbps]	[Mbps]
Stratix III	271	(9337, 0)	466	3704	4476
Stratix IV	264	(9322, 0)	454	3608	4360
Xilinx					
	[MHz]	[CLB Slices, BRAMs]	[Mbps]	[Mbps]	[Mbps]
Virtex 5	261 (+4.8%)	(2505 (+31%), 0)	449	3567	4310 (-29%)
Virtex 6	276	(2221, 0)	474	3773	4558

TABLE IV
RESULTS OF SHARED-RESOURCES IMPLEMENTATION FOR GRØSTL-0 (GRØSTL) AND AES IN ALTERA CYCLONE III

Design	Functionality	Frequency	Area	Latency	Throughput	Throughput/Area
		[MHz]	[Logic Elements]	[Cycles]	[Mbps]	[Mbps/Slice]
Järvinen [5]						
reference Grøstl-0	Grøstl-0	57.2	12086	20	1473	0.12
Grøstl-0 and 4*AES	Grøstl-0	56.0 (-2.6%)	13723 (+13.5%)	20	1434 (-2.6%)	0.10
	AES	56.0	13723	10	2868	0.21
	Grøstl-0 and AES	56.0	13723	30	956*)	0.07
Grøstl-0, 3*AES and Key Expansion	Grøstl-0	53.4 (-7.2%)	13453 (+11.3%)	20	1366 (-2.6%)	0.10
	AES	53.4	13453	10	2049	0.15
	Grøstl-0 and AES	53.4	13453	30	911*)	0.07
*) Throughput calculated for the authenticated encryption based on HMAC-Grøstl and AES-ECB						
This work						
reference Grøstl-0	Grøstl-0	141.1	19005	21	3440	0.18
Grøstl-0, 4*AES and Key Expansion	Grøstl-0 and AES	159.9 (+13.3%)	23039 (+23.4%)	31	2640 (-23.3%)	0.11
reference Grøstl	Grøstl	130.1	19260	21	3171	0.16
reference AES and Key Expansion	AES	129.4	4901	11	1505	0.31
Grøstl, 4*AES and Key Expansion	Grøstl and AES	144.0 (+10.7%)	23758 (+23.4%)	31	2378 (-25.0%)	0.10

B. Comparison to the Järvinen design [5]

In order to fairly compare our hardware accelerator with the circuit described in [5], an additional implementation in Altera low-cost Cyclone III is provided (Table IV). In both our work and the [5] work, one can observe the penalty in area for introducing extra AES functionality. In case of [5] architectures, negligible frequency penalty was also introduced. This penalty is due to the fact that basic iterative task (P and Q Grøstl-0 functions and AES round) of the coprocessor proposed in [5] is fully combinational and extra multiplexers were added to the original Grøstl-0 design. In case of our architecture, an additional pipeline stage enables frequency improvement. In case of scenario when both encryption and hashing for a given block of data have to be computed, the design from [5] and our core will produce output in 30 and 31 clock cycles respectively. Due to the fact that our core has three pipeline stages, ideally our circuit should have 3 times higher frequency than [5]. The obtained result, 2.85x frequency improvement, proves the validity of this concept. A typical application for high-speed implementation of the combined confidentiality and integrity services is the coprocessor from [31]. This protocol works in two different modes: Encapsulating Security Payload (ESP) and Authentication Headers (AH). The first requires the usage of both block cipher and hash function at the same time for a given chunk of data, second requires a hash function usage only. Table IV summarizes results for both

modes for our and [5] coprocessors. In case of ESP request we can observe 57% and in case of AH 10% improvement in terms of efficiency (throughput/area).

VI. Conclusions

The hash function Grøstl is one of the five finalists of the SHA-3 competition. Hardware performance of this function was investigated thoroughly over the last few years.

In this paper we have investigated very unique feature among all SHA-3 candidates - Grøstl and the current Advanced Encryption Standard have similarities and they can be exploited very efficiently in hardware. Their common structure can be utilized in the combined data-path implementation. The coprocessor was optimized for high-speed implementation of both functions and can find practical application to the IPSec-based secure networks. It outperforms the hardware accelerator proposed in [5] for both IPSec modes: IP Encapsulating Security Payload (ESP) and Authentication Headers (AH) by 57% and 10%, respectively.

The fully functional HMAC/Grøstl with CTR/AES hardware accelerator, compared to the stand alone quasi-pipelined architecture of Grøstl, described in [18], pays the price in terms of throughput and the area on all reported devices and in particular on Virtex 5: 29% in case of throughput and 31% in terms of area. Not surprisingly, the maximum frequency of proposed design increases (+4.8% for Virtex 5) as the number of pipeline stages was increased by one stage.

From our point of view, the main advantage of Grøstl over other SHA-3 finalists is the fact that the relatively small overhead in its hardware architecture enables a natural adoption of the most important to date block cipher - the Advanced Encryption Standard.

References

[1] "SHA-3 Contest," http://csrc.nist.gov/groups/ST/hash/sha3/index.html.

[2] "Sha-3 zoo," http://ehash.iaik.tugraz.at/wiki/SHA-3_Hardware_Implementations, 2011.

[3] "ATHENa results database," http://cryptography.gmu.edu/athenadb/, Automated Tool for Hardware EvaluatioN project.

[4] A. Schorr and M. Lukowiak, "Skein tree hashing on FPGA," in *International Conference on ReConFigurable Computing and FPGAs ReConFig'10*, 2010.

[5] K. Järvinen, "Sharing resources between AES and the SHA-3 second round candidates fugue and grøstl," Jul 2010, presented on SHA-3 workshop in Santa Barbara, Aug. 2010.

[6] N. At, J.-L. Beuchat, and I. San, "Compact Implementation of Threefish and Skein on FPGA," in *5th IFIP International Conference on New Technologies, Mobility and Security (NTMS'12)*, 2012.

[7] P. Gauravaram, L. R. Knudsen, K. Matusiewicz, F. Mendel, C. Rechberger, M. Schäffer, and S. S. Thomsen, "Tweaks on grostl," 2011, http://www.groestl.info/Round3Mods.pdf.

[8] ——, "Grøstl – a SHA-3 candidate," Submission to NIST (Round 3), 2011, http://www.groestl.info/Groestl.pdf.

[9] ——, "Grøstl – a SHA-3 candidate," Submission to NIST, Oct 2008, http://www.groestl.info/.

[10] M. Rogawski and K. Gaj, "Groestl Tweaks and their Effect on FPGA Results," Dec. 2011, http://eprint.iacr.org/2011/635.pdf.

[11] S. Tillich, M. Feldhofer, M. Kirschbaum, T. Plos, J.-M. Schmidt, and A. Szekely, "High-speed hardware implementations of BLAKE, Blue Midnight Wish, CubeHash, ECHO, Fugue, Grøstl, Hamsi, JH, Keccak, Luffa, Shabal, SHAvite-3, SIMD, and Skein," Cryptology ePrint Archive, Report 2009/510, Nov 2009, http://eprint.iacr.org/2009/510.pdf.

[12] L. Dadda, M. Macchetti, and J. Owen, "The design of a high speed ASIC unit for the hash function SHA-256 (384, 512)," in *Design, Automation, and Test in Europe*, vol. 3. Washington, DC, USA: IEEE Computer Society, 2004.

[13] M. Macchetti and L. Dadda, "Quasi-pipelined hash circuits," in *17th IEEE Symposium on Computer Arithmetic*, P. Montuschi and E. Schwarz, Eds., Computer Arithmetic Society. IEEE, June 2005.

[14] E. Homsirikamol, M. Rogawski, and K. Gaj, "Comparing hardware performance of fourteen round two SHA-3 candidates using FPGAs," Cryptology ePrint Archive, Report 2010/445, 2010, http://eprint.iacr.org/.

[15] B. Jungk and S. Reith, "On FPGA-based implementations of the sha-3 candidate grøstl," in *International Conference on Reconfigurable Computing (ReConFig)*, Dec 2010, pp. 316 – 321.

[16] E. Homsirikamol, M. Rogawski, and K. Gaj, "Throughput vs. area trade-offs architectures of five round 3 SHA-3 candidates implemented using Xilinx and Altera FPGAs," in *Workshop on Cryptographic Hardware and Embedded Systems CHES 2011*, ser. LNCS, B. Preneel and T. Takagi, Eds. Springer Berlin / Heidelberg, Sep 2011.

[17] R. Shahid, M. U. Sharif, M. Rogawski, and K. Gaj, "Use of embedded fpga resources in implementations of 14 round 2 sha-3 candidates," in *The 2011 International Conference on Field-Programmable Technology (FPT'11)*, Dec. 2011.

[18] K. Gaj, E. Homsirikamol, and M. Rogawski, "Fair and comprehensive methodology for comparing hardware performance of fourteen round two SHA-3 candidates using FPGA," in *Cryptographic Hardware and Embedded Systems, CHES 2010*, ser. LNCS. Springer, 2010.

[19] S. Matsuo, M. Knežević, P. Schaumont, I. Verbauwhede, A. Satoh, K. Sakiyama, and K. Ota, "How can we conduct "fair and consistent" hardware evaluation for SHA-3 candidate?" Second SHA-3 Candidate Conference, Tech. Rep., 2010.

[20] B. Baldwin, N. Hanley, M. Hamilton, L. Lu, A. Byrne, M. O'Neill, and W. P. Marnane, "FPGA implementations of the round two SHA-3 candidates," Second SHA-3 Candidate Conference, Tech. Rep., 2010.

[21] K. Kobayashi, J. Ikegami, S. Matsuo, K. Sakiyama, and K. Ohta, "Evaluation of hardware performance for the SHA-3 candidates using SASEBO-GII," http://eprint.iacr.org/2010/010, Jan 2010.

[22] X. Guo, S. Huang, L. Nazhandali, and P. Schaumont, "On the impact of target technology in sha-3 hardware benchmark rankings," 2010, http://eprint.iacr.org/2010/536.pdf.

[23] M. U. Sharif, R. Shahid, M. Rogawski, and K. Gaj, "Use of embedded FPGA resources in implementations of five round three SHA-3 candidates," ECRYPT II Hash Workshop 2011, May 2011.

[24] Algotronix, "http://www.algotronix-store.com/."

[25] Helion, "http://www.heliontech.com/," 2011.

[26] M.-Y. Wang, H. C.-T. Su, Chih-Pin, and C.-W. Wu, "An HMAC processor with integrated SHA-1 and MD5 algorithms," in *Asia and South Pacific Design Automation Conference*, 2004.

[27] K. Jaervinen, M. Tommiska, and J. Skytta, "A compact md5 and sha-1 co-implementation utilizing algorithms similarities," in *International Conference on Engineering of Reconfigurable Systems and Algorithms*. CSREA Press, 2005, p. 48?54.

[28] D. Cao, J. Han, and X.-Y. Zeng, "A reconfigurable and ultra low-cost VLSI implementation of SHA-1 and MD5 functions," in *7th International Conference on ASIC ASICON*, 2007, pp. 862–865.

[29] T.-S. N. Chiu-Wah Ng and K.-W. Yip, "A unified architecture of MD5 and RIPEMD-160 hash algorithms," in *International Symposium on Circuits and Systems*, vol. 2, 2004, p. 889?892.

[30] T. Ganesh, M. Frederick, T. Sudarshan, and A. Somani, "Hashchip: A shared-resource multi-hash function processor architecture on fpga," *Integration the VLSI journal*, vol. 40, pp. 11–19, 2007.

[31] RFC-4301, "http://www.ietf.org/rfc/rfc4301.txt," 2005.

[32] *Advanced Encryption Standard (AES)*, National Institute of Standards and Technology (NIST), FIPS Publication 197, Nov 2001, http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

[33] "GMU SHA-3 source codes," ONLINE, 2011, http://cryptography.gmu.edu/athena/index.php?id=source_codes.

[34] RFC-2104, "http://www.ietf.org/rfc/rfc2104.txt," 1997.

[35] RFC-6151, "http://www.ietf.org/rfc/rfc6151.txt," 2011.

[36] M. Dworkin, *NIST Special Publication 800-38A: Recommendation for Block Cipher Modes of Operation*, 2001, http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf.