Cryptography Based On Hash Function BLAKE 32 in VLSI

Gibi Sunny[1], C Saranya[2]

**(Department Of Electronics And Communications ,K S Rangasamy College Of Technology ,India*

*** (Department Of Electronics And Communications, K S Rangasamy College Of Technology, India*

ABSTRACT

An important commodity in the world of Electronic Communication is information. The protection of authenticity and integrity of information is necessary to achieve a secure communication between communicating parties. Electronic security is becoming increasingly important as the Internet and other forms of electronic communication become more prevalent. BLAKE is a hash function selected by NIST as one of the 14 second round candidates for the SHA-3 Competition. In this paper, a BLAKE 32 is proposed based on the algorithmic specification and round rescheduling function which indicates a high speed implementation of BLAKE 32. Again, memory architecture is designed to reduce the memory usage which was done in Xilinx Spartan 3 family.

Keywords:

Cryptography, hash functions, MD5, SHA-3, BLAKE

1. Introduction

Cryptography is the practice and study of techniques for secure communication while third parties are present.The word cryptography comes from the Greek words κρυπτο (hidden or secret) and γραφη (writing).It is an art of secret writing. Cryptography is used to protect e-mail messages, credit card information, and corporate data. The basic service provided by cryptography is the ability to send information between participants that prevents others from reading it. Cryptography can provide other services, such as integrity checking and authentication—verifying someone's or something's identity.

Cryptography achieves security by encoding messages to make them non-readable. Cyptanalysis is the technique of decoding messages from a non-readable format to readable format without knowing that how it was initially converted. Cryptology is the combination of these two. A message in its original form is known as plaintext or clear text. The mangled information is known as cipher text. The two disciplines in cryptography are encryption and decryption. The process for producing cipher text from plaintext is known as encryption. The reverse of encryption is called decryption. While cryptographers invent clever secret codes, cryptanalysts attempt to break these codes[13]. There are three main types of cryptography:-Secret key cryptography (symmetric key cryptography) where both the sender and the receiver know the same secret code, called the key. Public key cryptography (asymmetric encryption) where uses a pair of keys for encryption and decryption and Hash Functions.

Cryptographic hash functions play an important central role in cryptology. Hash functions are applied to support digital signatures, data integrity, random number generators. A cryptographic hash function is a hash function, or an algorithm that takes an arbitrary block of data and returns a fixed-size bit string, the (cryptographic) hash value. It satisfies three major cryptographic properties: pre image resistance, second pre image resistance and collision resistance [1]. Due to these properties, hash function has become an important cryptographic tool which is used to protect information authenticity and integrity. The "message," is the data to be encoded and the hash value is sometimes called the message digest or simply digest. In general, the input to a hash function is called as a message or plain text and output is often referred to as message digest, the hash value, hash code, hash result or simply hash.

The most common properties [12], or ideal characteristics, of a hash function for a secure hashing function H with input message x are as follows:

a) $H(x)$ is relatively easy for any given message x, so that the implementations of H in both hardware and software can be efficient.

b) For any given hash digest d, it is computationally impractical to find a message x such that the hash digest of x is the same as the hash digest d. This is to ensure that the hash function is one-way.

c) For any given message x, it is computationally infeasible to find another message y, such that the message y is not the same as message x but the hash digest of y and x are the same. Some sources referred to this as weak collision resistance.

d) It is computationally infeasible to find two different messages, x and y such that their hash digests are the same. Some sources referred to this as strong collision resistance.

1.1 Password Protection

One common place to use cryptographic hash is password storage. The user will have to initially setup a password that the computer stores, in some form onto the hard drive. The next time someone tries to access that account, the stored

Manuscript received January 5, 2014
Manuscript revised January 20, 2014

password will be retrieved and compared to the password entered by the user. This may pose a security problem because if the password is stored in plaintext in a storage device such as the hard-drive, someone may remove the storage device and extract physical data from it to obtain the password of that user. To prevent this, a cryptographic hash function may be used. Instead of storing the password directly into the hard drive, the password can run through a hash function first, and having the digest to be stored instead. When an authorized user tries to access that account later, his or her entry will be hashed and compared to the stored digest. Due to the deterministic nature of the hash function, if the user had entered his or her password correctly the digests will be a match and the user will be granted access to that account.

MD5 and SHA are commonly used hash functions. Both of them adopt the Merkle-Damgard construction [21]. MD5 is a common Merkle-Damgard-based hash function [2] & [3]. A MD5 block has 512 bits, which can be divided into sixteen 32-bit words. This configuration is only useful when the input is parallel in nature.

1.2 What is SHA?

SHA stands for Secure Hashing Algorithm. It is a group of hash functions published by the National Institute of Standards and Technology as a US Federal Information Standard. All of the current SHA algorithms are developed by the NSA [2] [3].

SHA-0: A 160-bit hash function published in 1993. It was quickly withdrawn due to an undisclosed flaw. It was replaced by SHA-1.

SHA-1: A 160-bit hash function that is similar to the earlier MD5 algorithm but more conservative. It is developed by the National Security Agency to be a part of the Digital Signature Algorithm [4]. It is the most widely used SHA algorithm.

SHA-2: A family of two similar hash functions. It comes with four different sizes for the output, 224, 256, 384, and 512-bit. The 224-bit and 384-bit versions of SHA-2 are simply the 256-bit and 512-bit versions with truncated outputs.

SHA-3: This future hashing function is still under development. The algorithm will be developed by choosing different algorithms to a public competition. The final decision is expected to be announced in 2012 [5].

BLAKE is our candidate for SHA-3.The heritage of BLAKE is threefold [6]:

• BLAKE's **iteration mode** is HAIFA, an improved version of the Merkle-Damgard paradigm proposed by Biham and Dunkelman. It provides resistance to long-message second preimage attacks, and explicitly handles hashing with a salt.

• BLAKE's **internal structure** is the local wide-pipe, which we already used with the LAKE hash function. It makes local collisions impossible in the BLAKE hash functions, a result that doesn't rely on any intractability assumption.

• BLAKE's **compression algorithm** is a modified version of Bernstein's stream cipher ChaCha, whose security has been intensively analyzed and performance is excellent, and which is strongly parallelizable.

The iteration mode HAIFA would significantly provides randomized hashing and structural resistance to second-preimage attacks. The LAKE local wide-pipe structure is a straightforward way to give strong security guarantees against collision attacks. Finally, the choice of stream cipher ChaCha comes from our experience in cryptanalysis of Salsa20 and ChaCha, convinced of their remarkable combination of simplicity and security.

2. Procedure

2.1 Design Principles

The BLAKE hash functions were designed to meet all NIST criteria for SHA-3, including [17]:

• message digests of 224, 256, 384, and 512 bits
• same parameter sizes as SHA-2
• one-pass streaming mode
• maximum message length of at least 2^64 − 1 bits

In addition, we imposed BLAKE to:

• explicitly handle hashing with a salt
• be parallelizable
• allow performance trade-offs
• be suitable for lightweight environments

BLAKE is a family of four hash functions: BLAKE-224, BLAKE-256, BLAKE-384, and BLAKE- 512 (see Table 1). BLAKE has a 32-bit version (BLAKE-256) and a 64-bit one (BLAKE-512) [7], from which other instances are derived using different initial values, different padding, and truncated output.

Table 1. Properties of the BLAKE hash functions (sizes in bits).

algorithm	*word*	*message*	*block*	*digest*	*salt*
BLAKE-224	32	<2^64	512	224	128
BLAKE-256	32	<2^64	512	256	128
BLAKE-384	64	<2^128	1024	384	256
BLAKE-512	64	<2^128	1024	512	256

The sizes were chosen as the SHA-2 sizes doubled, since increases in computing power require increases in hash sizes to maintain acceptable levels of security (against brute force or other attacks). The variants also have slight differences in the size of words processed and maximum message length.

2.1.1 Advantages

2.1.1.1 Design

• simplicity of the algorithm
• interface for hashing with a salt

2.1.1.2 Performance

• fast in both software and hardware
• parallelism and throughput/area trade-off for hardware implementation
• simple speed/confidence trade-off with the tunable number of rounds

2.1.1.3 Security

• based on an intensively analyzed component (ChaCha)
• resistant to generic second-pre image attacks
• resistant to side-channel attacks
• resistant to length-extension
2.1.2 Limitations
• message length limited to respectively 264 and 2128 for BLAKE-256 and BLAKE-512
•Resistance to Joux's multi collisions similar to that of SHA-2
• Fixed-points found in less time than for an ideal function.

2.2 Specification

The hash function BLAKE-256 operates on 32-bit words and returns a 32-byte hash value. The hash functions process of this family, is mainly based on two operations: the modular adder of 2n, for unsigned integers, and the bit-by-bit XOR (exclusive OR) on n-bit words [10]. The size of the hash value will be equal to the block length .In addition; the right rotation operation of k-bit is used. The heart of BLAKE is the compression function.

BLAKE-256 starts hashing from the same initial value as SHA-256.

IV0	= 6A09E667	IV1	= BB67AE85
IV2	= 3C6EF372	IV3	= A54FF53A
IV4	= 510E527F	IV5	= 9B05688C
IV6	= 1F83D9AB	IV7	= 5BE0CD19

Ten permutations of {0, . . . , 15} are used by all BLAKE functions, defined in Table 2. The unary operator >>> denotes rotation of words towards least significant bits.

BLAKE-256 uses 16 constants

c0	= 243F6A88	c1	= 85A308D3
c2	= 13198A2E	c3	= 03707344
c4	= A4093822	c5	= 299F31D0
c6	= 082EFA98	c7	= EC4E6C89
c8	= 452821E6	c9	= 38D01377
c10	= BE5466CF	c11	= 34E90C6C
c12	= C0AC29B7	c13	= C97C50DD
c14	= 3F84D5B5	c15	= B5470917

Table 2. Permutations of {0. . . 15} used by the BLAKE

σ0	σ1	σ2	σ3	σ4	σ5	σ6	σ7	σ8	σ9
0	14	11	7	9	2	12	13	6	10
1	10	8	9	0	12	5	11	15	2
2	4	12	3	5	6	1	7	14	8
3	8	0	1	7	10	15	14	9	4
4	9	5	13	2	0	14	12	11	7
5	15	2	12	4	11	13	1	3	6
6	13	15	11	10	8	4	3	0	1
7	6	13	14	15	3	10	9	8	5
8	1	10	2	14	4	0	5	12	15
9	12	14	6	1	13	7	0	2	11
10	0	3	5	11	7	6	15	13	9
11	2	6	10	12	5	3	4	7	14
12	11	7	4	6	15	9	8	1	3
13	7	1	0	8	14	2	6	4	12
14	5	9	15	3	1	8	2	10	13
15	3	4	8	13	9	11	10	5	0

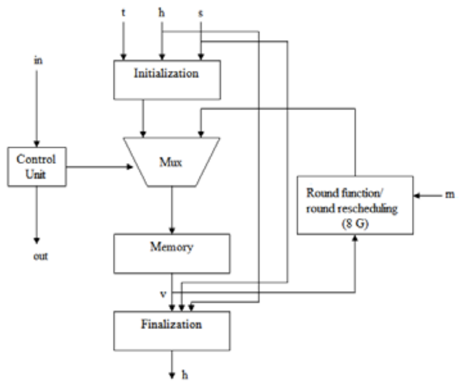
2.2.1 Compression Function

The compression function of BLAKE-256 takes as input four values:
• a chain value h = h0, . . . , h7
• a message block m = m0, . . . ,m15
• a salt s = s0, . . . , s3
• a counter t = t0, t1
These four inputs represent 30 words where one word is equal to 32 bit and therefore in total equal to 960 bits. The output of the function is a new chain value h0 = hd0………hd7 of eight words (i.e., 32 bytes = 256 bits)[8]-[11].The compression of h, m, s, t is given as

Compress (h ,m, s, t).

The compression function performs three main operations as shown in fig 1

a) Initialization
b) Round function
c) Finalization



t	=	counter
h	=	chaining values
s	=	salt values
v	=	internal values
m	=	message inputs

Fig 1. Overall flow diagram of BLAKE 32

2.2.1.1 Initialization

The internal states of BLAKE will first be initialized using a set of initial values, counter values, and some constants. This is the first step of the compression function. A 16-word state v0. . . v15 is initialized such that different inputs produce different initial states[8] [19] [20]. State is used as a fundamental word data structure. The state is represented as a 4×4 matrix. The output vectors v0…v7 is very easy to be implemented on hardware, since their integration is a matter of wiring. For the implementation of v9….v15 a XOR chain is used, between the values of salt (s0….s3) and the constant c (c0…c7) and are filled as follows:

$$\begin{pmatrix} v0 & v1 & v2 & v3 \\ v4 & v5 & v6 & v7 \\ v8 & v9 & v10 & v11 \\ v12 & v13 & v14 & v15 \end{pmatrix}$$

will be obtained from the following matrix as shown below:

$$\begin{pmatrix} h0 & h1 & h2 & h3 \\ h4 & h5 & h6 & h7 \\ s0 \text{ Xor } c0 & s1 \text{ Xor } c1 & s2 \text{ Xor } c2 & s3 \text{ Xor } c3 \\ t0 \text{ Xor } c4 & t0 \text{ Xor } c5 & t1 \text{ Xor } c6 & t1 \text{ Xor } c7 \end{pmatrix}$$

2.2.1.2 Round Function

The next step of compression is the round function. The round function of Blake is based on the ChaCha stream cipher. It is composed of two layers of G functions in parallel with four G functions in each layer. A G function uses modular addition, XOR, and rotational shifts. This operates for a specified number of rounds [11]. In every round, the state v is transformed based on certain operations addition, XOR and right rotation computations, which are the components of Gi (i=1,…,4) functions. For this purpose the Gi functions are used. The input message data first goes through a permutation process and then it is sent to one layer of G functions along with the stored internal states. Since one round requires two layers of G functions, each cycle only completes half a round. The output of the half round function will be stored inside the internal state registers for the next half round.

A round is a transformation of the state v that computes G0(v0 , v4 , v8 , v12) G1(v1 , v5 , v9 , v13) G2(v2 , v6 , v10, v14) G3(v3 , v7 , v11, v15) parallely because each updates a distinct column of the state. And G4(v0 , v5 , v10, v15),G5(v1 , v6 , v11, v12) G6(v2 , v7 , v8 , v13) G7(v3 , v4 , v9 , v14) diagonally as shown in fig 2. The sequence G0….G3 is called a column step [20]. And similarly, the last four calls G4…G7 are called diagonal step.

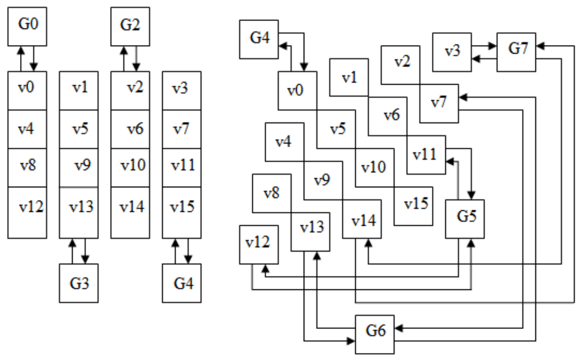


Figure 2. G functions in column step and diagonal step

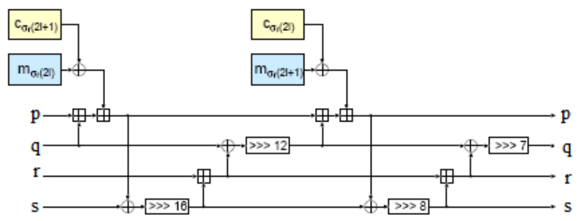


Figure 3. Gi Function

Algorithm for rounding (Figure 3) where, at round r, Gi (p, q, r, s) sets:-

$p := p+q+(m_{\sigma r(2i)} \text{ xor } c_{\sigma r(2i+1)})$

$s:=(s \text{ xor } p)>>> 16$

$r:= r + s$

$q:=(q \text{ xor } r)>>> 12$

p:= p+q+($m_{\sigma r(2i+1)}$ xor $c_{\sigma r(2i)}$)

s:=(s xor p)>>> 8

r:= r + s

q:=(q xor r)>>> 7

where p,q,r,s are the four internal states used for the round fnction and m and c are the message input and constants BLAKE 32 takes 16,12,8,7 rounds during the iteration process.

2.2.1.3 Finalization Process

When the round sequence is taken over, the new chain value h = hd0, …, hd7 is produced from the state v, with inputs of the initial chain value h0, …, h7 and the salt s = s0, …, s3.The new chaining values are formed by the Xor operation as shown below.

h0 xor s0 xor v0 xor v8	=hd0
h1 xor s1 xor v1 xor v9	=hd1
h2 xor s2 xor v2 xor v10	=hd2
h3 xor s3 xor v3 xor v11	=hd3
h4 xor s0 xor v4 xor v12	=hd4
h5 xor s1 xor v5 xor v13	=hd5
h6 xor s2 xor v6 xor v14	=hd6
h7 xor s3 xor v7 xor v15	=hd7

2.2.2 Hashing The Message

When hashing a message, the function starts from an initial value, and the iterated hash process computes intermediate hash values that are called chaining values. Before being processed, a message is first padded so that its length is a multiple of the block size (512 bits). It is then processed block per block by the compression function.

2.2.3 Round Rescheduling

The introduction of the addition with the message/constant (MC)-pair in the function will lead to an increase in the propagation delay. The maximum delay is given by the total delay of four XORs and four modular adders in the core function. The slightly modified function inserts an addition with the MC-pair [10] [11]. In modular adders, rotation is a simple rerouting of the word without effective propagation delay. The maximum frequency values of BLAKE architectures are slightly lower than those obtained for the stream cipher ChaCha. However, with a rescheduling function, it is possible to recover the original maximum path of ChaCha decreasing the overall propagation delay of the core function. Observing the flow dependencies in it is clear that the addition with the MC-pair is independent; message word and constant are unrelated to the state and can be computed in parallel to the other computations. If in a single call of G, each update of the state has been conceived to operate sequentially, the MC-pair addition can be shifted within the computations. It is thus possible to anticipate it, reducing the critical path. The rescheduled Gi (p*,q , r, s) computes the algorithm as :-

p:= p* +q

s:=(s xor p)>>> r0

r:= r + s

q:=(q xor r)>>> r1

p:= p+q+($m_{\sigma r(2i+1)}$ xor $c_{\sigma r(2i)}$)

s:=(s xor p)>>> r2

r:= r + s

q:=(q xor r)>>> r3

p*:= p+($c_{\sigma r+1(2i+1)}$ xor $m_{\sigma r+1(2i)}$)

Where ri are the rotation indices for BLAKE-32 and BLAKE-64, and p* corresponds to the modified first input/output variable after the MC addition [8].

2.2.4 Memory Architecture

The VLSI implementation of BLAKE-32 needs memory to store 16 words of internal state and eight words of chaining value, plus additional registers to store the salt (four words), the counter (two words), and the message block (16 words), i.e., in total 1472 bits of memory. The counter is used during four clock cycles and needs thus to be stored. The memory units are the main contribution in terms of area and energy consumption. It is thus necessary to design special-purpose register elements, to decrease [8] the global resource requirements of the hash core.BLAKE-32 semi-custom memories was introduced based on clock-gated latch arrays, able to store at most one new word per cycle. Depending on the word number of the target value that is to be stored, these memories replaces the standard flip-flop cells by latch cells. The latches are organized in 32-bit banks and each bank stores a single word and is triggered by a dedicated gated clock.

A four-word latch array (Figure 4) is used to store the salt value. In the address decoder, the different one-hot enable signals are activated, depending on the write address [11]. An input flip-flop bank is added to prevent timing loops inside the logic, caused by the transparent behavior of latches. This bank is driven by a gated clock generated with the write enable signal, while the outputs of the flip-flops are connected to the inputs of all latch banks. When a write enable occurs, the input word is first stored inside the flip-flop bank and is passed to the activated latch bank.The codes are designed for each block in VHDL and simulated in Xilinx using Spartan 3 FPGA family.

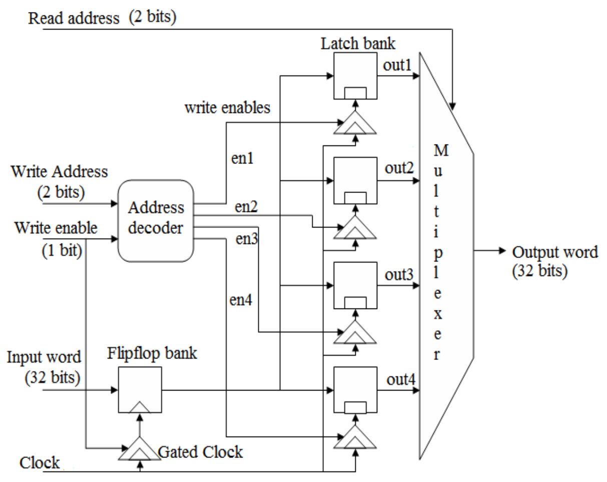


Figure 4. Memory architecture

Table 3.Synthesis Result For Algorithm specification Of BLAKE 32

DELAY	62.826 ns
GATE COUNT	40,434
MEMORY USAGE	460332 kbytes

The synthesis result for the BLAKE 32 with the round function is specified in the Table 3. The input given is of 32 bit. The table gives the delay, gate count and memory usage.

Table 4. Synthesis Result For High Speed Implementation Of BLAKE 32

DELAY	53.298ns
GATE COUNT	29,396
MEMORY USAGE	398444 kbytes

The synthesis result for the BLAKE 32 with the round rescheduling is specified in the Table 4. This indicates smaller delay than the round function which gives more speed than the previous.

Table 5. Synthesis Result For Compact architecture Of BLAKE 32

DELAY	40.648ns
GATE COUNT	13,390
MEMORY USAGE	170916 kbytes

The synthesis result for the BLAKE 32 with a memory architecture is specified in the Table 5. This stores the salt values which reduces the area and also increases the speed further.

3. Conclusion and future work

Research in cryptographic hash function has recently witnessed an unprecedented spike of interest. Around 50-60 hash functions were available in 1993, followed by at least 30-40 others developed, in addition to the 64 SHA-3 submissions in 2007. The cryptographic hash standard SHA-3 should be suitable and flexible for a wide range of applications, featuring at the same time an optimal security strength. A complete hardware characterization of the BLAKE candidate, using G functions to generate fully-autonomous high-speed and compact implementations. A round rescheduling technique and a special-purpose memory design are also proposed. The wide spectrum of achieved performances paves the way for the application of the BLAKE function to various hardware implementations.

The future study can be the reduction of area further by removing the message block memory and the salt support and compare for BLAKE (28, 32, 48 and 64). For the first case, we can suppose the presence of an external tamper resistant memory that stores the secret message and for the second case we omit an added functionality of the BLAKE algorithm.

References

[1] A. Sotirov, M. Stevens, J. Appelbaum, A. Lenstra, D. Molnar, D. A. OsvikB, and B. de Weger, "MD5 considered harmful today. Creating a rogue CA certificate," presented at the 25th Chaos Commun. Congr., Berlin, Germany, 2008.

[2] C. D. Cannière and C. Rechberger, "Finding SHA-1 characteristics: General results and applications," in Advances in Cryptology—ASIACRYPT 2006, ser. Lecture Notes in Computer Science. Berlin, Germany: Springer, 2006, vol. 4284, pp. 1–20.

[3] D. J. Bernstein, "CubeHash Specication (2.b.1), submission to NIST," 2008. [Online]. Available: http://cubehash.cr.yp.to/

[4] D. J. Bernstein, "ChaCha, a Variant of Salsa20," 2007. [Online]. Available: http://cr.yp.to/chacha.html

[5] G. Bertoni, J. Daemen, M. Peeters, and G.Van Assche, "Keccak sponge function family, submission to NIST," 2008. [Online]. Available: http:// keccak.noekeon.org/

[6] J.-P. Aumasson, L. Henzen, W. Meier, and R. C.-W Phan, "SHA-3 Proposal BLAKE, submission to NIST," 2008.

[7] J. Kelsey and B. Schneier, "Second preimages on n-bit hash functions for much less than □ work," in *EUROCRYPT, ser. Lecture Notes in Computer Science*, R. Cramer, Ed. New York: Springer, 2005, vol. 3494, pp. 474–490.

[8] Luca Henzen, student member, IEEE, Jean-Philippe Aumasson, Willi Meier, and Raphael C.-W. Phan, member, IEEE,"VLSI Characterization of The Cryptographic Hash Function-BLAKE" IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 19, no. 10, october 2011.

[9] L. Henzen, F. Carbognani, N. Felber, and W. Fichtner, "VLSI hardware evaluation of the stream ciphers Salsa20 and ChaCha, and the compression function Rumba," in *Proc.*

IEEE Int. Conf. Signals, Circuits Syst. (SCS), Nov. 2008, pp. 1–5.

[10] L. Lu, M. O'Neill, and E. Swartzlander, "Hardware evaluation of SHA-3 hash function candidate ECHO," presented at the Claude Shannon Workshop Coding Cryptography, Cork, Ireland, 2009.

[11] M. Bernet, L. Henzen, H. Kaeslin, N. Felber, and W. Fichtner, "Hardware implementations of the SHA-3 candidates Shabal and CubeHash," in *Proc. IEEE Midw. Symp. Circuits Syst. (MWSCAS)*, Cancun, Mexico, Aug. 2009, pp. 515–518.

[12] M. Stevens, A. Lenstra, and B. de Weger, "Chosen-prefix collisions for MD5 and colliding X.509 certificates for different identities," in Advances in Cryptology—EUROCRYPT 2007, ser. Lecture Notes in Computer Science. Berlin, Germany: Springer, 2007, vol. 4515, pp. 1–22.

[13] M. Tehranipoor and C. Wang , Introduction to Hardware Security and Trust, Springer Science+Business Media, LLC 2012," Zhijie Shi, ChujiaoMa, Jordan Cote, and Bing Wang," Hardware Implementation of Hash Functions" DOI 10.1007/978-1-4419-8080-9 2.

[14] NIST, Gaithersburg, MD, "Announcing the secure hash standard," FIPS 180-2, 2002.

[15] NIST, Gaithersburg, MD, "SP 800-106, randomized hashing digital signatures," 2007.

[16] O. Küçük, "The Hash Function Hamsi, submission to NIST,"2008.[Online].Available:http://homes.esat.kuleuven.be/~okucuk/hamsi/

[17] Proceedings of IEEE Computer Society Annual Symposium on VLSI (IEEE ISVLSI'10), "BLAKE HASH Function Family on FPGA: From the Fastest to the Smallest", Kefalonia, Greece, July 5-7, 2010.

[18] P. Gauravaram, L. R. Knudsen, K. Matusiewicz, F. Mendel, C. Rechberger, M. Schläffer, and S. S. Thomsen, "Grøstl—A SHA-3 Candidate Submission to NIST," 2008. [Online]. Available: http://www.groestl.Info

[19] 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, Rep. 2009/510, 2009, .

[20] S. Tillich, M. Feldhofer, W. Issovits, T. Kern, H. Kureck, M. Mühlberghuber, G. Neubauer, A. Reiter, A. Köfler, and M. Mayrhofer, "Compact hardware implementations of the SHA-3 candidates ARIRANG, BLAKE, Grøstl, and Skein," Cryptology ePrint Archive: , Rep. 2009/349, 2009.

[21] X. Wang and H. Yu, "How to break MD5 and other hash functions," in Advances in Cryptology—EUROCRYPT 2005, ser. Lecture Notes in Computer Science. Berlin, Germany: Springer, 2005, vol. 3494, pp. 19–35.