NIST Interagency Report 7764

National Institute of
Standards and Technology
U.S. Department of Commerce

Status Report on the Second Round of the SHA-3 Cryptographic Hash Algorithm Competition

Meltem Sönmez Turan
Ray Perlner
Lawrence E. Bassham
William Burr
Donghoon Chang
Shu-jen Chang
Morris J. Dworkin
John M. Kelsey
Souradyuti Paul
Rene Peralta

NIST Interagency Report 7764

Status Report on the Second Round of the SHA-3 Cryptographic Hash Algorithm Competition

Meltem Sönmez Turan
Ray Perlner
Lawrence E . Bassham
William Burr
Donghoon Chang
Shu-jen Chang
Morris J. Dworkin
John M. Kelsey
Souradyuti Paul
Rene Peralta

C O M P U T E R S E C U R I T Y

Computer Security Division
Information Technology Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899-8930

February 2011

U.S. Department of Commerce

Gary Locke, Secretary

National Institute of Standards and Technology

Patrick D. Gallagher, Director

Abstract

The National Institute of Standards and Technology (NIST) opened a public competition on November 2, 2007 to develop a new cryptographic hash algorithm – SHA-3, which will augment the hash algorithms currently specified in the Federal Information Processing Standard (FIPS) 180-3, Secure Hash Standard. The competition was NIST's response to advances in the cryptanalysis of hash algorithms.

NIST received sixty-four submissions in October 2008, and selected fifty-one candidate algorithms as the first-round candidates on December 10, 2008, and fourteen as the second-round candidates on July 24, 2009. One year was allocated for the public review of the second-round candidates. On December 9, 2010, NIST announced five SHA-3 finalists to advance to the third (and final) round of the competition. This report summarizes the evaluation and selection of the five finalists – BLAKE, Grøstl, JH, Keccak and Skein.

KEY WORDS: Cryptographic hash algorithm; Cryptographic hash function; Cryptography; Cryptographic hash competition; SHA-3 competition.

Acknowledgements

NIST thanks the submitters of all fourteen second-round candidates. NIST is also grateful for the efforts of those in the cryptographic community that provided security, implementation and performance analyses of the candidate algorithms during the second round, including the eBASH and SHA-3 Zoo organizers, and those who provided feedback on the hash forum or published papers on the various technical aspects of the candidates.

The authors of this report also thank the other members of NIST's SHA-3 team, who reviewed the candidate algorithms and the public comments; performed testing; provided technical and administrative support; and participated in numerous meetings to discuss the selection of the finalists. They are: Elaine Barker, Sara J. Caswell, Lily Chen, Quynh Dang, James Nechvatal and Andrew Regenscheid.

TABLE OF CONTENTS

1. Introduction

1.1 Purpose of this Document

This report summarizes the second round of the SHA-3 (Secure Hash Algorithm-3) competition, which began on September 28, 2009 when the second-round candidates were posted on NIST's hash web site (http://www.nist.gov/hash-competition), and ended on December 9, 2010 when the following second-round candidates were announced as the SHA-3 finalists:

- BLAKE,
- Grøstl,
- JH,
- Keccak, and
- Skein.

The report explains the evaluation and selection process for the second round of the competition. A summary of the public analyses of each second-round candidate, and justifications for whether or not a candidate is advanced to the third (and final) round of the competition are provided in this report.

1.2 Background

The National Institute of Standards and Technology (NIST) opened a public competition on November 2, 2007 to develop a new cryptographic hash algorithm (referred to as SHA-3) to augment the hash algorithms currently specified in Federal Information Processing Standard (FIPS) 180-3, Secure Hash Standard [1]. The competition was NIST's response to advances in the cryptanalysis of hash algorithms in recent years. An attack by Wang et al. [2], and extended by many others, has seriously called into question the security of the SHA-1 government standard hash algorithm when used for the generation of digital signatures and in other security applications that require collision resistance.

NIST published a *Draft Minimum Acceptability Requirements, Submission Requirements, and Evaluation Criteria for candidate hash algorithms* for public comment in a Federal Register Notice in January 2007 (FRN-Jan07) [3]. These requirements and evaluation criteria were updated, based on public feedback, and included in a later, second Federal Register Notice published on November 2, 2007 (FRN-Nov07) [4], which called for the submission of candidate algorithms and launched the "SHA-3" competition.

The competition has generated great interest in the cryptographic community, and inspired numerous entries from around the world. NIST received sixty-four entries by October 31, 2008, and selected fifty-one candidate algorithms to advance to the first round in December 2008, and fourteen to advance to the second round on July 24, 2009. The selection of the second-round candidates was summarized in the first-round report [5]. Submitters of the second-round candidates were allowed to make minor modifications to their algorithms. The final submissions for the second round were posted on NIST's web site on September 28, 2009, and the second round officially began. A year was allocated for the public review of the fourteen second-round candidates, which are listed in Table 1, and their submission packages are available at http://csrc.nist.gov/groups/ST/hash/sha-3/Round2/submissions_rnd2.html.

NIST hosted the Second SHA-3 Candidate Conference [6] at the University of California, Santa Barbara (UCSB) on August 23-24, 2010, to discuss the security and performance analyses of the second-round candidates; the submitters also provided a status update of their algorithms.

NIST received significant feedback from the cryptographic community, both before and after the conference. Most of the comments were sent to both NIST and a public hash forum (`hash-forum@nist.gov`); in addition, many of the cryptanalysis and performance studies were published as papers in major cryptographic conferences or leading cryptographic journals. Based on this public feedback and internal review of the second-round candidates, NIST chose five finalists – BLAKE, Grøstl, JH, Keccak, and Skein to advance to the final round of the competition on December 9, 2010, which ended the second round of the competition.

Table 1. Second-round Candidates

Algorithm	*Designers*
BLAKE [7]	Jean-Philippe Aumasson, Luca Henzen, Willi Meier, Raphael C.-W. Phan
BLUE MIDNIGHT WISH [8]	Danilo Gligoroski, Vlastimil Klima, Svein Johan Knapskog, Mohamed El-Hadedy, Jørn Amundsen, Stig Frode Mjølsnes
CubeHash [9]	Daniel J. Bernstein
ECHO [10]	Ryad Benadjila, Olivier Billet, Henri Gilbert, Gilles Macario-Rat, Thomas Peyrin, Matt Robshaw, Yannick Seurin
Fugue [11]	Shai Halevi, William E. Hall, Charanjit S. Jutla
Grøstl [12]	Praveen Gauravaram, Lars R. Knudsen, Krystian Matusiewicz, Florian Mendel, Christian Rechberger, Martin Schläffer, Søren S. Thomsen
Hamsi [13]	Özgül Küçük
JH [14]	Hongjun Wu
Keccak [15]	Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche
Luffa [16]	Christophe De Cannière, Hisayoshi Sato, Dai Watanabe
Shabal [17]	Emmanuel Bresson, Anne Canteaut, Benoît Chevallier-Mames, Christophe Clavier, Thomas Fuhr, Aline Gouget, Thomas Icart, Jean-François Misarsky, Marìa Naya-Plasencia, Pascal Paillier, Thomas Pornin, Jean-René Reinhard, Céline Thuillet, Marion Videau
SHAvite-3 [18]	Eli Biham, Orr Dunkelman
SIMD [19]	Gaëtan Leurent, Charles Bouillaguet, Pierre-Alain Fouque
Skein [20]	Niels Ferguson, Stefan Lucks, Bruce Schneier, Doug Whiting, Mihir Bellare, Tadayoshi Kohno, Jon Callas, Jesse Walker

Below is a timeline of major events in the SHA-3 competition, including events leading to the start of the competition, and up to the close of the second round.

October 31- November 1, 2005	Cryptographic Hash Workshop, NIST, Gaithersburg, MD.
August 24-25, 2006	Second Cryptographic Hash Workshop, UCSB, CA.
January 23, 2007	Federal Register Notice - Announcing the Development of New Hash Algorithm(s) for the Revision of Federal Information Processing Standard (FIPS) 180-2, Secure Hash Standard.
November 2, 2007	Federal Register Notice - Announcing a Request for Candidate Algorithm Nominations for a New Cryptographic Hash Algorithm (SHA-3) Family.
	SHA-3 competition began.
October 31, 2008	SHA-3 Submission Deadline.
December 10, 2008	First-round candidates announced. The first round began.
February 25-28, 2009	First SHA-3 Candidate Conference, K.U. Leuven, Belgium.
July 24, 2009	Second-round candidates announced. End of the first round.
September 28, 2009	Second-round candidates posted for public review. The second round began.
August 23-24, 2010	Second SHA-3 Candidate Conference, UCSB, CA.
December 9, 2010	SHA-3 finalists announced. End of the second round.

1.3 Organization of this Document

Section 2 summarizes the evaluation criteria specified in FRN-Nov07. Section 3 describes the selection process of the SHA-3 finalists. Section 4 summarizes the fourteen second-round candidates, each with a brief description and a summary of publicly-available security analyses and implementation and performance results. Section 5 concludes the report and describes the next steps forward for the competition.

2. Evaluation Criteria

Throughout the competition, NIST used the evaluation criteria specified in FRN-Nov07 to evaluate the candidates. These criteria were discussed and further clarified at the First SHA-3 Candidate Conference in Katholieke Universiteit Leuven, Belgium in February 2009 [21]. In relative order of importance, NIST considered the security, cost, and algorithm and implementation characteristics of a candidate in its selection decision.

Security

FRN-Nov07 lists the major security-evaluation factors as:

(i) applications of the hash functions[1];
(ii) specific requirements when hash functions are used to support HMAC, Pseudo Random Functions (PRFs), or Randomized Hashing;
(iii) additional security requirements of hash functions;
(iv) evaluations relating to attack resistance; and
(v) other consideration factors.

For its security evaluations, NIST studied the large amount of feedback received from the cryptographic community via e-mail discussions in the official mailing list (`hash-forum@nist.gov`), and the SHA-3 Zoo web page [22], as well as the security arguments presented by the designers. NIST researchers also conducted their own internal cryptanalysis of the candidates.

Cost and Performance

FRN-Nov07 identified cost as the second evaluation criterion, which includes:

(i) *computational efficiency*, which refers to the speed of the algorithm; and
(ii) *memory requirements*, which includes the code size and the random-access memory (RAM) requirements for software implementations, as well as the gate-counts for hardware implementations.

For its software evaluations, NIST considered various benchmarking comparisons. The primary contributor was the ECRYPT Benchmarking of All Submitted Hashes (eBASH) project [23] that provided speed measurements of the SHA-3 candidates on a wide variety of 32- and 64-bit platforms. eBASH recompiled the implementations submitted by the designers and other parties with various compiler options, presented the best results in medians and quartiles for message sizes from eight to 4096 bytes and provided extrapolations for longer messages. Another software-benchmarking project – the eXternal Benchmarking eXtension (XBX) – provided performance results on small devices [24]. Other studies on the software performance of the candidates include [25, 26].

For hardware evaluations, NIST mainly considered the results from studies [27-34, 127] that evaluated the performance of all second-round candidates.

Algorithm and Implementation Characteristics

The last evaluation criterion identified in FRN-Nov07 is related to the *flexibility* and the *simplicity* of the design. The FRN stated that candidates with greater flexibility are preferable. Flexibility includes algorithms capable of running efficiently on a wide variety of platforms, as well as algorithms that use parallelism or instruction-set extensions to achieve higher performance. FRN-Nov07 also stated that a candidate would be judged according to its relative design simplicity; the intent was to encourage designs that would be easy to understand and analyze, thus providing more confidence in the security of the design.

[1] In this report, the terms “hash function” and “hash algorithm” are used interchangeably.

3. Selection Process

Selecting the SHA-3 finalists was a challenging task, since all second-round candidates were very strong contenders for the hash algorithm standard.

Security was NIST's primary criterion in selecting the finalists. None of the second-round candidates was completely broken according to the criteria NIST laid out in the FRN-Nov07 and further clarified at the First SHA-3 Candidate Conference. NIST received many security observations, including "partial attacks" on several candidates, i.e., cryptanalysis of weakened variants. Absent more conclusive information on the security of these candidates, a principal concern was the potential for extending these cryptanalyses. NIST tried to extrapolate the severity of any partial attacks, and thus to estimate the "security margin", as an estimate of susceptibility of the full hash algorithms to future attacks.

An important caveat is that this extrapolation is only meaningful when the algorithm has received a significant amount of cryptanalysis, and when there is a good deal of granularity in choosing any tunable parameter that was provided for the algorithm. When the final SHA-3 algorithm is chosen, it will have been analyzed for only about three years. NIST's goal is that SHA-3 will remain secure for at least twenty years after its selection. Algorithms that received very little cryptanalysis, and algorithms without a clearly adjustable security parameter were, therefore, evaluated with more suspicion.

While having a tunable parameter was helpful in classifying security margins against various attacks, a number of factors complicated the selection process. Firstly, not all attacks are equally important. A computationally expensive distinguisher on a putatively random permutation is of much less concern than a practical collision attack on the hash function as a whole, even if both attacks are on the same number of rounds. Additionally, generic attacks that exploit the large-scale structure of a hash function may not depend on the tunable parameter at all. Finally, it is not always obvious that there is an easily predictable mapping between the difficulty of an attack and the value of the tunable parameter. It is a lot less certain what one out of two rounds means in terms of practical security than ten out of twenty, and even the latter may be questionable if the twenty rounds are very different from each other. Because of all of these complications, it is somewhat misleading to consider the security margin, even under the best of circumstances, as a single, well-defined quantity. NIST, therefore, tried to map the various quantitative data on existing attacks on each hash function into a qualitative judgment of how likely the existing attacks were to evolve into something that meaningfully diminished its security, and whether sufficient confidence could be achieved in each algorithm to select it by the end of the competition.

Performance was also important in choosing the finalists. In FRN-Nov07, NIST stated that it expected SHA-3 to have a security strength that is at least as good as the hash algorithms currently specified in FIPS 180-2[2], and that this security strength would be achieved with significantly improved efficiency. However, during the analysis of the second round candidates, it became apparent that significant improvement in efficiency while fulfilling the security requirements was not easily attainable. NIST chose finalists that could be implemented on as wide a variety of platforms as possible, while still maintaining reasonably good performance. In the performance studies performed during the second round, every candidate achieved at least

[2] FIPS 180-2 [125], Secure Hash Standard, was the Government hash standard when the SHA-3 competition began; it was superseded by FIPS 180-3, Secure Hash Standard, in October 2008, and again by Draft FIPS 180-4 [126], Secure Hash Standard, in February 2011.

tolerable performance on mainstream desktop or server systems, although the performance range was significant. There were bigger differences on constrained platforms and in hardware, where area is as much a performance factor as speed. During the selection process, a couple of algorithms were eliminated or viewed negatively because of very large area requirements, which would preclude their use in much of the potential application space. Discussions included the implementation possibilities for the algorithms: some algorithms allowed very high levels of fine-grain parallelism that could be realized well with hardware, some exploited parallelism with vector units, and some seemed to fully exploit the considerable parallelism that could be achieved by conventional superscalar arithmetic logic units (ALUs) capable of simultaneously launching several instructions per clock cycle. NIST also noted that several algorithms exploited the power of 64-bit-wide ALUs. Candidates that performed poorly in hardware or on platforms without a specialized AES instruction, for example, were not selected, since many platforms do not provide this capability. In the end, the algorithms that were selected as finalists have efficiency at least comparable to that of SHA-2.

Although NIST allowed the designers to tweak (i.e., make minor modifications to) their algorithms for the second round, extensive tweaks in order to avoid certain attacks were a cause for concern. NIST was generally comfortable with tweaks to the number of rounds or to constants, but more suspicious of changes that seemed to affect the structure of the compression functions, since these tweaks made much of the first-round cryptanalysis on the algorithms obsolete. However, NIST did consider whether the best attacks on some of the candidates seemed amenable to mitigation by a simple modification.

NIST also considered positive security arguments and proofs in making its decision. Since block cipher cryptanalysis appears to be more advanced than hash algorithm cryptanalysis, arguments that link the security of the candidates to a component that can be treated as a block cipher or fixed permutation were especially valuable.

Secondary selection considerations for the finalists also included design diversity. As stated in FRN-Nov07, one of NIST's goals was that an attack on SHA-2 would be unlikely to also be applicable to SHA-3. Similarly, NIST preferred to select a set of finalists with dissimilar designs, so that a single attack technique would be very unlikely to break the entire set of finalists. Achieving this level of design diversity was easy, given the candidates. The finalists include many different hash function constructions, and many different underlying compression function structures.

Additional design features, such as a built-in pseudorandom function (PRF) or an authenticated encryption mode were also briefly discussed, but played little part in the selection of the finalists.

Some teams announced the tweaks that they would make if their algorithm was selected for the final round. NIST did not consider these tweaks in the finalists' selection.

4. Summary of the Second-Round Candidates

This section provides a brief description of each second-round candidate, including their basic building blocks, mode of operation and tunable parameters. Each description is followed by a summary of the security analysis and the performance evaluations received before December 9, 2010, and a discussion of NIST's decision whether to advance the candidate as a finalist.

4.1 BLAKE

BLAKE follows a HAIFA iteration mode and operates on an inner state that can be represented as a four by four matrix of words. The inner state is initialized using an Initial Value (IV), a salt and a counter. The state is updated using the *G* function, which is based on the *ChaCha* stream cipher [35]. The *G* function updates columns and the disjoint diagonals of the state using modular addition, XOR, and rotate operations. The input message blocks and constants are selected using round-dependent fixed permutations. The nonlinearity of the design is achieved by the modular addition.

To generate 224-, 256-, 384- and 512-bit outputs, four instances are proposed: BLAKE-28, -32, -48, and -64. BLAKE-28 is similar to BLAKE-32, except for the IVs and the padding structure. A similar relationship holds for BLAKE-48 and BLAKE-64.

The tunable parameter of the design is the number of rounds, which is recommended to be ten for BLAKE-28 and -32, and fourteen for BLAKE-48 and -64, respectively. Four toy versions are defined by the submitters for analysis purposes: *(i)* BLOKE (with identity permutations), *(ii)* FLAKE (with no feed-forward), *(iii)* BLAZE (with zero constants), and *(iv)* BRAKE (with all the changes above).

4.1.1 Security

The first public analysis of BLAKE was done by Li and Xu [36]. The authors used a method to control some of the intermediate hash words by message modification techniques, and presented free-start collision and preimage attacks for BLAKE, with the compression function reduced to 2.5 rounds. The time complexities are 2^{96}, 2^{112}, 2^{160} and 2^{224} for the collision attack, and 2^{209}, 2^{241}, 2^{355} and 2^{481} for the preimage attack, respectively, for BLAKE-28, -32, -48, and -64.

Guo and Matusiewicz [37] presented a near-collision attack for the compression function of BLAKE-32, reduced to four of the middle rounds, starting from round three, with complexity 2^{56}. The *G* function is linearized by replacing the additions by the XOR operation, and then rotation-invariant differences are introduced to the chaining value, the salt, the counter, and the message to obtain near-collisions with 24-bit differences.

Aumasson et al. [38] proved that one round of BLAKE is a permutation of the message and presented an efficient inversion algorithm. The authors presented an improved preimage attack on 1.5 rounds, with complexity 2^{128}, and impossible differentials for the permutation with five and six rounds for BLAKE-32 and -64, respectively. The authors also described near-collisions for four of the middle rounds of the compression function of BLAKE-32.

Turan and Uyan [39] used a hill-climbing approach that introduced random differences to message blocks and showed practical near-collisions with 47 and 72 bits of difference for the compression function of BLAKE-32, reduced to 1.5 and two rounds, respectively, with complexity 2^{26}.

Vidali et al. [40] observed that for a fixed state, it is possible to find a message block such that the state does not change after an arbitrary number of rounds, and used these fixed points to efficiently obtain collisions for the BLOKE toy version of BLAKE. The authors also presented an internal collision attack for BRAKE.

Su et al. [41] used the linear differential techniques to analyze the near-collision resistance of the compression function of BLAKE and showed near-collision attacks on 152, 396, and 306 bits for

the compression functions of BLAKE-32, -64 and -64 reduced to four, four, and five middle rounds, with complexity 2^{21}, 2^{16}, and 2^{216}, respectively.

4.1.2 Performance

BLAKE is among the top performers in software across most platforms for long messages. BLAKE-32 is the best performer on software platforms for very short messages. The computation of 512-bit digests was optimized for 64-bit processors, but BLAKE-64 did considerably less well when implemented on 32-bit platforms. In constrained environments, BLAKE is among the top performers in speed and requires relatively less ROM. Blake-64 requires a little bit more ROM than Blake-32, because the constants are twice as large. In hardware, BLAKE allows four-way parallelism and has average performance in its throughput-to-area ratio; however, BLAKE has a structure that allows for flexible designs. This flexibility allows cost-effective tradeoffs in area usage, with limited impact on the throughput-to-area ratio.

4.1.3 Discussion

BLAKE was selected as a finalist, due to its high security margin, good performance in software, and its simple and clear design.

4.2 Blue Midnight Wish

Blue Midnight Wish (BMW) is a wide-pipe Merkle-Damgård hash algorithm with a unique design strategy. It derives its nonlinearity from mixing modular addition, XOR, rotate and shift operations.

The compression function has three sub-functions: f_0, f_1, and f_2. Function f_0 consists of an initial permutation, which is linear with respect to modular addition, followed by a second permutation, which is computed from modular addition, XOR, rotate, and shift operations. The chaining value is then rotated and fed-forward to the output of the second permutation to generate the resulting value of f_0.

Function f_1 operates on the output of f_0 and the message block, and expands the internal state of BMW into a quadruple-pipe state. Function f_1 consists of sixteen expand rounds, the first two of which use a more complicated round than the other fourteen rounds. Thus, the standard version of BMW is referred to as having two $expand_1$ rounds and fourteen $expand_2$ rounds.

Function f_2 operates on the outputs of f_0, f_1 and the message block, and generates the next chaining value using shift, rotate, XOR and modular addition operations; however, function f_2 is more irregular than f_1.

BMW was significantly tweaked to protect against attacks published during the first round of the competition. In its second-round version, the feed-forward operation was added to f_0, and function f_1 was modified to include the chaining value as an input. Moreover, a finalization round was added.

4.2.1 Security

The most severe attacks on the original version of BMW were by Thomsen, who provided a practical near-collision on the compression function [42], and a pseudo-collision attack [43] that provided a small, but significant, advantage to the attacker over a brute force attack. BMW's extensive tweak, as mentioned above, guards against this attack: either of the feed-forward steps added to the compression function would have foiled the attack as presented by Thomsen.

Since the publication of the tweaked BMW, a number of distinguishers have been presented for the round function of BMW [44, 45] or a slight variant thereof [46]. Of these, the attack by Guo and Thomsen [45] was the most severe for much of the selection process. This provided a practical, probability-one distinguisher on the tweaked compression function by using differential cryptanalysis to force nonrandom behavior of the first word of the chaining value. A similarly severe attack was presented at the Crypto2010 Rump Session by Leurent [47]. This attack produced a collision on the first 96 bits of the chaining value with complexity 2^{32}. Leurent and Thomsen [48] have extended the attack to 300 of the 512 bits of the chaining value at a similar cost.

4.2.2 Performance

BMW is among the top performers in software across most platforms for long messages, and remains a strong performer on software platforms for very short messages. The computation of 512-bit digests is optimized for 64-bit processors, but performs less well when implemented on 32-bit platforms. In constrained environments, BMW is among the top performers in speed, and is average in ROM usage. BMW requires significant amounts of RAM, due to its large state size. In hardware, its throughput-to-area ratio is below average; however, above-average throughputs can be obtained, but at an appreciable cost in area.

4.2.3 Discussion

Although BMW has very good software performance, and good potentials for pipelining and area-efficient high-throughput values, a disadvantage of the algorithm is its irregular and not-well-understood design. Since the compression function of BMW does not have a conventional iterated-round structure, there appears to be no obviously simple way to adjust the security margin, or to trade performance for security. Moreover, the attacks on the algorithm, even after an extensive tweak, did not provide confidence in the security of the algorithm. For these reasons, BMW was not selected as a finalist.

4.3 CubeHash

CubeHash is a sponge-like hash algorithm that is based on the iteration of a fixed permutation, *T*, on a state of 128 bytes. A simple padding rule is applied to the input message. The algorithm consists of three stages: initialization, message injection, and finalization. In addition to the parameter, *h*, for the bit length of the message digest size, there are two tunable parameters: *(i)* the number of rounds of message injection, *r*, and *(ii)* the message word size, *b*, in bytes. Thus, for the message injection stage of CubeHash*r*/*b*-*h*, *b* bytes of the message are XORed into the state before *r* iterations of *T*. The initialization and finalization stages each consist of 10*r* rounds of *T* on the state.

Initialization may either be pre-computed to save time, or computed on-the-fly to save memory. Finalization is preceded by the flipping of a single bit of the state. No round constants or length indicators are used, other than the encodings of the parameters in the three initial-state words.

The operations in the round permutation, *T*, are modular additions, XORs, and rotations; the source of nonlinearity is the overlap of modular additions and XOR operations.

The submitter's documentation in [49] proposes:

CubeHash16/32-224 for SHA-3-224,
CubeHash16/32-256 for SHA-3-256,
CubeHash16/32-384 for SHA-3-384-normal,

CubeHash16/32-512 for SHA-3-512-normal,
CubeHash16/1-384 for SHA-3-384-formal, and
CubeHash16/1-512 for SHA-3-512-formal.

For the two larger digest sizes, the "normal" proposals are about 32 times faster than the corresponding "formal" proposals, and about sixteen times faster than the original CubeHash8/1 proposal for these digest sizes, at the cost of reducing the theoretical security below the thresholds that NIST established in FRN-Nov07. The "formal" proposals were offered to comply with these security requirements. The document asserts the following, with supporting arguments:

> "For all real-world cryptographic applications, 'formal' versions here can be ignored, and this document amounts to a proposal of CubeHash16/32 as SHA-3."

Therefore, NIST evaluated the "normal" versions when determining whether CubeHash would advance to the next round of the competition.

4.3.1 Security

The submitter's security analysis in [50] includes a "standard generic preimage attack" on CubeHash*r*/*b* using roughly $2^{522-4b-\lg b}$ calls of *T*. The attack does not exploit the specifics of *T*, but rather the size of the internal state, i.e., the "narrow pipe," coupled with the choice of *b*.

For the "normal" variant, CubeHash16/32-512, this attack requires 2^{388} calls of *T*, which meets the threshold for being regarded as a serious attack, according to FRN-Nov07. NIST elaborated at the First SHA-3 Candidate Conference that attacks on the *n*-bit digest size that require between $2^{n/2}$ and 2^n operations would be considered as "wounding" the candidate, but not necessarily as disqualifying it [51].

CubeHash attracted extensive third-party analysis. No practical attacks were found, but some non-ideal properties were identified in connection with symmetric states in three separate papers [52, 53, 54].

The following is a summary of [52]:

1. The submitter's generic preimage attack was improved by a factor of 512/*b*–4, i.e., a factor of 12 when *b*=32, excluding "non-negligible communication costs."
2. The same improved techniques can be used to generate the zero state as a chaining value. Because this state is a fixed point of the round transformation, finding this preimage would yield arbitrarily large multi-collisions by extending the message by the desired number of "zero" words. The complexity of the attack, however, is not more efficient than brute force or Joux's multi-collision techniques [55], unless the word size is much larger than the proposed *b*=32, e.g. in CubeHash*r*/112-512.
3. Fifteen classes of symmetric states, a total of about 2^{516} states, were defined and analyzed. The structure of *T* preserves membership in each of these classes. This observation is the basis for a meet-in-the-middle preimage attack for several ranges of values for *b*. The complexity of the attacks is similar to the submitter's generic preimage attack.
4. A simple, practical collision attack was presented on a variant of CubeHash that is weakened by changing the initial state to a symmetric state.
5. Symmetric-state classes were considered in the analysis of the number of fixed points of *T*, with the conclusion that *T* can be expected to have significantly more fixed points than

a random permutation. The zero state is a fixed point; a non-trivial fixed point was also presented.

6. Lastly, truncated differentials were applied to show how to detect non-randomness over ten iterations of *T*.

The work in [53] extended this analysis. A flaw in the reasoning of the preimage and collision attacks in items three and four above was identified. In the case of the collision attack, the flaw led to an overly optimistic estimate of the attack complexity: one would expect to need about 2^{64} calls of *T*, instead of 2^{33}, when starting from a symmetric initial state in a class with 2^{64} members. A modification of the meet-in-the-middle preimage attack was given to correct a similar flaw in the original reasoning. The resulting complexity is slightly fewer than 2^{384} hash operations.

An analogous preimage attack, based on symmetric states, was presented on variants of CubeHash that are weakened by increasing *b*. For example, when *b*=33, the complexity of the attack is slightly more than 2^{256} hash operations.

Leurent [54] explained how to apply Grover's algorithm with a hypothetical quantum computer to reach a symmetric state in CubeHash. Consequently, if quantum computers are ever developed, the time complexity of the meet-in-the-middle preimage attack, based on symmetric states, would be reduced to 2^{192} for CubeHash16/32-512. Collision attacks with essentially the same complexity also follow from this observation.

Several sets of authors contributed analysis that is unrelated to the symmetric states:

Brier et al. [56] gave an extensive, general framework for the differential analysis of hash algorithms, based on a linearized estimate of the compression function, e.g., replacing modular addition with XOR, in the style of the attacks initiated on SHA-0 in [57]. The framework was applied to a variety of parameter choices for CubeHash. This analysis was extended in [58], which gives an explicit collision for CubeHash5/96-512 and a collision attack on CubeHash 8/96, whose complexity is about 2^{80} calls of the compression function.

Ashur and Dunkelman [59] analyzed variants that are significantly weakened in the following ways: *(i)* reducing the number of rounds, *(ii)* eliminating the finalization stage, *(iii)* allowing the attacker to control the entire input state, and *(iv)* allowing the attacker to observe the entire output state. Under these conditions, linear approximations were presented, with biases that are high enough to allow theoretical distinguishers on up to eleven iterations of *T*. For example, an approximation with bias 2^{-235} was presented on eleven iterations of *T*, which allows this function to be distinguished from a random permutation using about 2^{470} calls.

Earlier results from other authors included the following: Dai [60] reported example collisions for CubeHash1/45-512 and CubeHash2/89-512. Aumasson [61] reported an example collision on CubeHash 2/120-512. Khovratovich et al. [62] specialized the submitter's generic meet-in-the-middle preimage attack to CubeHash*r*/8-512 and CubeHash*r*/4-512, including the details of how to execute the attack with negligible memory.

4.3.2 Performance

CubeHash is a fairly strong performer in software across most platforms for long messages when the processor includes a vector unit. Without a vector unit, the algorithm suffers dramatically. CubeHash has a large overhead during finalization and thus, is a poor performer on software platforms for very short messages. In constrained environments, CubeHash has poor performance, but requires less ROM and RAM than most of the second-round candidates. In

hardware, CubeHash is in the top tier of performers in its throughput-to-area ratio, largely resulting from its very compact design. A valuable property of CubeHash is that all four required digest sizes can be generated from a single, compact, hardware implementation or program, by simple parameter changes.

4.3.3 Discussion

CubeHash is a simple, well-understood design that received extensive third-party analysis. The design is suitable for constrained environments, as it requires a small area in hardware, and little RAM or ROM in software. However, it has poor performance without the use of a vector unit, resulting in a large overhead for short messages. Moreover, no single variant for the 512-bit digest size achieves the required theoretical security with acceptable performance. Another disadvantage of the design is the existence of the symmetric properties that are arguably a potential avenue for future attacks. NIST felt that an additional year of study would not be enough to determine whether or not the symmetric properties pose a threat. Therefore, NIST felt that CubeHash could not be chosen as SHA-3 with confidence, and thus should not be selected as a finalist.

4.4 ECHO

ECHO is an AES-based hash algorithm that has received a great deal of analysis. The design is strongly based on AES at several levels, and the cryptanalysis of ECHO has progressed alongside the cryptanalysis of other AES-based hash algorithms.

4.4.1 Security

Mendel et al. [63] presented a distinguisher on the permutation underlying the ECHO compression function for seven rounds (ECHO-256 has eight rounds, ECHO-512 has ten rounds) requiring 2^{384} time and 2^{64} memory complexity.

Gilbert and Peyrin [64] presented an impractical distinguisher for eight rounds, requiring 2^{768} time and 2^{512} memory complexity.

Peyrin [65] presented a chosen-salt semi-free-start collision on the ECHO compression function for three rounds, requiring 2^{96} time and 2^{32} memory complexity.

Jean and Fouque [66] presented a semi-free-start collision on four rounds of the ECHO compression function, requiring 2^{52} time and 2^{16} memory complexity, and several semi-free-start near-collisions on four rounds of ECHO's compression function, including some that require 2^{36} time and 2^{16} memory complexity.

Schläffer [67] presented a collision attack on four rounds of the full ECHO-256 hash algorithm (out of eight rounds), requiring 2^{64} time and 2^{64} memory complexity. He also claimed a 4.5-round near-collision and a five-round distinguisher on the compression function, as well as a seven-round distinguisher for the ECHO compression function (both 256- and 512- bit versions). Some of these results, but not the four-round full collision, were later disputed.

Schläffer [68] published a collision attack on five rounds of ECHO-256 that requires 2^{112} time and 2^{85} memory complexity, along with a number of related attacks, including a free-start near-collision on up to 6.5 rounds.

Sasaki [69] presented an attack that detects a non-ideal property of the full round ECHO-256 permutation with 2^{182} time and 2^{37} memory complexity. The time complexity of the attack was improved to 2^{151} by Naya-Plasencia [70] with 2^{67} memory complexity.

These attacks do not appear to seriously threaten ECHO; instead, they leave a security margin of three rounds against collision attacks, and somewhat less against distinguishing attacks.

4.4.2 Performance

ECHO is a below-average performer in software across most platforms for long messages. When the processor includes AES-NI instructions, the performance is increased significantly. ECHO is a poor performer on software platforms for very short messages. In constrained environments, ECHO has poor performance, with a slightly higher ROM requirement than average and a fairly high RAM requirement. In hardware, ECHO has very poor performance in its throughput-to-area ratio; however, it can make use of embedded resources in FPGAs to reduce the area needed.

4.4.3 Discussion

Although ECHO appears to be a simple secure design, it was not selected as a finalist, due to its all-around poor performance.

4.5 Fugue

Fugue is a sponge-like design inspired by the hash algorithm Grindahl [71]. It operates on an internal state consisting of columns. Each column is a 32-bit word. The underlying hash algorithm for Fugue is denoted as $F[n, s, k, r, t]$, where

n is the number of words in the output,
s is the number of columns in the state,
k is the number of sub-rounds per round transformation,
r is the number of rounds in the first phase of finalization and
t is the number of rounds in the second phase of the final transformation.

The default values for Fugue-224, -256, -384 and -512 are $F[7, 30, 2, 5, 13]$, $F[8, 30, 2, 5, 13]$, $F[12, 36, 3, 6, 13]$ and $F[16, 36, 4, 8, 13]$, respectively. A weak version of *F-256*, denoted as *wF-*256 has been proposed with parameters $F[8, 30, 1, 5, 5]$.

Fugue-256 operates on an internal state that consists of thirty 32-bit columns, and the state is updated using a function similar to the round function of AES. First, the state is initialized by setting the first 22 columns to zero and the rest to the IV. Then the message blocks are inserted into the state using a round transformation that consists of the following operations: (*i*) **TIX(I)** loads the 32-bit message blocks into the state, (*ii*) **ROR3** rotates the state by three columns, (*iii*) **CMIX** mixes the columns, and (*iv*) **SMIX** applies a nonlinear substitution to the first four columns of the state. After processing all message blocks, the state undergoes a finalization function *G*. After the finalization, the hash output is generated from eight columns of the state.

4.5.1 Security

Fugue is one of the least-analyzed candidates. The first external analysis of Fugue was done by Khovratovich [72], in which structures were used to find collisions. However, the attack requires 2^{352} time and memory complexity, which is significantly higher than the generic attack complexities.

Aumasson and Phan [73] presented an efficient distinguisher for the full eighteen-round finalization of Fugue-256. The distinguisher uses differential characteristics for fifteen of these

rounds with probability one. The authors also showed that the attack is valid, even if the number of rounds increases to thirty.

Turan and Uyan [39] used a hill-climbing approach by introducing random differences to message blocks and showed actual collisions and near-collisions for significantly reduced versions of Fugue.

4.5.2 Performance

Fugue is an average performer in software across most platforms for long messages, but is a below-average performer on software platforms for very short messages. In constrained environments, Fugue has above-average performance, a high ROM requirement, and an average RAM requirement. In hardware, Fugue is average in its throughput-to-area ratio for 256-bit digests, and below-average for 512-bit digests. Fugue also has a flexible structure that allows for various area tradeoffs.

4.5.3 Discussion

Fugue is an innovative design and has decent, all-around performance. Its designers also proved that collisions are unlikely to be found using differential cryptanalysis techniques. Nonetheless, Fugue has received only a very small amount of cryptanalysis. The most important published result, Aumasson and Phan's distinguisher on Fugue's finalization function, raised some concern. NIST felt that it would not be possible to establish confidence in the hash algorithm after another year of cryptanalysis; therefore, it was not selected as a finalist.

4.6 Grøstl

Grøstl is a wide-pipe Merkle-Damgård hash algorithm with an output transformation. The compression function is a novel construction, using two fixed 2*n*-bit permutations together, in order to produce a 2*n*-bit compression function with the goal of achieving the collision and preimage resistance of an ideal *n*-bit-wide compression function. Intermediate chaining values in Grøstl-256 are 512 bits wide; for Grøstl-512, they are 1024 bits wide. The output transformation processes the final chaining state, and discards half the bits of the result, yielding an *n*-bit hash output.

The underlying fixed permutations are themselves based closely on the structure of AES, reusing the S-box, but expanding the size of the block to 512 bits (for Grøstl-256) or 1024 bits (for Grøstl-512) in a straightforward way.

4.6.1 Security

Grøstl has been the subject of a great deal of cryptanalysis, and so its security is probably well-understood, relative to that of most of the other candidates.

Mendel et al. [74] showed a six-round (out of ten) semi-free-start collision on the Grøstl-256 compression function with 2^{120} time and 2^{64} memory complexity. Mendel et al. [63] improved the previous result and showed a six-round semi-free-start collision on the Grøstl-256 compression function with 2^{64} time and memory complexity, a seven-round distinguisher on the permutation component of the Grøstl-256 compression function with 2^{55} time complexity and a seven-round distinguisher on the output transformation of Grøstl-256.

Gilbert et al. [64] provided a seven-round semi-free-start collision on the Grøstl-256 compression function with 2^{120} time and 2^{64} memory complexity, and an eight-round distinguisher for the Grøstl-256 compression function that requires 2^{112} time and 2^{64} memory complexity.

Mendel et al. [75] also presented a four-round collision on the Grøstl-256 hash algorithm, requiring 2^{64} time and memory complexity; a seven-round semi-free-start collision on the Grøstl-256 compression function, requiring 2^{120} time and 2^{64} memory complexity; a five-round (out of fourteen) collision on the Grøstl-512 compression function, requiring 2^{176} time and 2^{64} memory complexity; and a seven-round semi-free-start collision on the Grøstl-512 compression function, requiring 2^{152} time and 2^{64} memory complexity.

Peyrin [76] demonstrated a five-round collision on the Grøstl-256 compression function that requires 2^{79} time and 2^{64} memory complexity; a six-round collision on the Grøstl-512 compression function, requiring 2^{177} time and 2^{64} memory complexity; a full round distinguisher on the Grøstl-256 compression function; a full round distinguisher on the Grøstl-256 component permutations; an eleven-round (out of fourteen) distinguisher on the Grøstl-512 compression function and an eleven-round distinguisher on the Grøstl-512 component permutations.

In addition, Ideguchi et al. [77] showed a six-round collision on the Grøstl-256 hash algorithm, requiring 2^{112} time and 2^{32} memory complexity; a seven-round semi-free-start collision on the Grøstl-256 compression function, requiring 2^{80} time and 2^{64} memory complexity; an eight-round semi-free-start collision on the Grøstl-256 compression function, requiring 2^{192} time and 2^{64} memory complexity; a seven-round distinguisher on the Grøstl-256 component permutations, requiring 2^{19} time complexity with no significant memory requirement; and an eight-round distinguisher on the Grøstl-256 component permutations, requiring 2^{64} time and 2^{64} memory complexity.

Unsurprisingly, the attacks have improved over time, as the basic techniques (rebound attacks, super-S-box attacks, differentials considered between corresponding rounds of the *P* and *Q* component permutations) are extended and refined.

4.6.2 Performance

Grøstl is an average performer in software across most platforms for both long and short messages. Grøstl can gain some advantage from the AES-NI instruction, but not as much as algorithms that use the full round function of AES, i.e., ECHO and SHAvite-3. In constrained environments, Grøstl has poor performance, but requires lower ROM and RAM than average. In hardware, Grøstl is a bit above-average in its throughput-to-area ratio. Grøstl also has a flexible structure that allows for various area tradeoffs.

4.6.3 Discussion

Grøstl was selected as a finalist because of its well-understood design and solid performance, especially in hardware. While Grøstl's security margin is not ideal, NIST views it in light of the extensive amount of cryptanalysis that has been published, both on Grøstl itself and the larger AES structure on which Grøstl is based. Due to the large amount of existing cryptanalysis, NIST feels that future results are less likely to dramatically narrow Grøstl's security margin than that of the other candidates.

4.7 Hamsi

Hamsi is based on a concatenate-permute-truncate design strategy, and it supports different output sizes with two instances: Hamsi-256 and Hamsi-512. The round transformation of Hamsi operates on a four by four and a four by eight matrix for Hamsi-256 and -512, respectively, where each matrix entry is a 32-bit word. The state is initialized using the chaining value and the expanded message block. After the application of the round transformations, the second and the

fourth row of the state are truncated, and the input chaining value is fed-forward to the truncated state.

The compression function consists of the following round transformations: *(i) Addition of Constants,* where the state is XORed with predefined constants and a round counter; *(ii) Substitution,* where the columns of the state go through a four by four S-box (taken from Serpent, and implemented using bit-slicing); and *(iii) Diffusion,* where a linear transformation *L*, which inputs four 32-bit words and outputs four 32-bit words is applied to the independent diagonals of the state.

4.7.1 Security

Aumasson [78] showed that the compression function of Hamsi-256 does not behave as a pseudo-random function, due to its low algebraic degree; this property remains even when two additional rounds are used. According to this result, the degree of the output bits is limited to 81 for five rounds, when the expected amount would be 127 for a random function.

In the rump session of Crypto 2009, Nikolic [79] showed near-collisions with a 25-bit distance for the compression function of Hamsi-256 by introducing differences only in the chaining value. A similar result was obtained by Aumasson et al. [80]. These two results were improved by Wang et al. [81], and Li and Wang [82], with 23- and 20-bit differences, respectively. Turan and Uyan [39] proposed random differences to message blocks and showed practical near-collisions with 24- and 64-bit differences for the one- and two-round compression function of Hamsi-256 using a hill-climbing approach.

Çalık and Turan [83] demonstrated a distinguishing attack that requires a few evaluations of the compression function, and extended the distinguisher to the compression function with two additional rounds at a 2^{83} time complexity. In addition, the authors of [83] presented a message recovery attack with $2^{10.48}$ time complexity in which the attacker is assumed to be able to query the compression function with chosen chaining values. Another result of [83] was a pseudo-preimage attack for the compression function with time complexity $2^{254.25}$ that can trivially be converted to a pseudo-second-preimage attack for the full hash algorithm with the same complexity.

Aumasson et al. [80] also analyzed the resistance of Hamsi-256 to differential and higher-order differential cryptanalysis and showed efficient distinguishers for the compression function and the full finalization of the hash algorithm. The authors presented a differential for six rounds with probability 2^{-148}, and a truncated differential on 180 output bits with probability $2^{-120.8}$.

Fuhl [84] studied the second-preimage resistance of Hamsi-256 and showed some affine equations between input and output bits of the compression function that enable finding second-preimages with time complexity $2^{251.3}$.

Dinur and Shamir [85] used a differential algebraic attack against Hamsi-256 and showed that the attack is slightly better than exhaustive search for long messages.

4.7.2 Performance

Hamsi is a below-average performer in software across most platforms for long messages, but gains an advantage over other algorithms when processing very short messages. Hamsi takes considerable advantage of vector-unit instructions to increase its performance. The computation of 512-bit message digests is optimized for 64-bit processors, but Hamsi-512 performs less well when implemented on 32-bit platforms. In constrained environments, Hamsi has average

performance, a high ROM requirement, and an average RAM requirement. In hardware, Hamsi is average in its throughput-to-area ratio for 256-bit digests, and below-average for 512-bit digests.

4.7.3 Discussion

Hamsi was not selected as a finalist, mainly due to the second-preimage attacks and high ROM requirements.

4.8 JH

The JH family of hash algorithms for different output sizes is based on a single compression function F_8, which uses the fixed permutation E_8. The different members of the JH family are distinguished by using different IVs. Hash values can be obtained just by a truncation of the final output. JH's domain extension is the chopped, wide-pipe Merkle-Damgård construction, where for output size s, the chaining value size n satisfies $n \geq 2s$.

JH can be efficiently implemented in the bitslice mode. There are three components of the underlying permutation E_8: four-bit S-boxes, an eight-bit L-permutation, and a P-permutation. For the S-boxes and L-permutation, JH can be directly implemented in the bitslice mode. However, for the P-permutation, 128 different shift values are used, so there is no efficient bitslice implementation for the P-permutation. Instead of implementing P directly, the submitter used seven different permutations P^0–P^6, which can be efficiently implemented in the bitslice mode.

4.8.1 Security

Several analyses on JH have been published, but none of these gives any serious security concern. Bhattacharyya et al. [86], and Mendel and Thomsen [87] analyzed the preimage resistance of the domain extension of JH; however, they failed to find a preimage with complexity less than an exhaustive search [88]. The attack in [87] requires 2^{524} memory accesses and comparisons, $2^{509.3}$ compression-function calls, and 2^{509} element-sorting complexity, whereas the attack in [86] requires 2^{516} memory accesses and comparisons, 2^{508} compression-function calls, and 2^{507} element-sorting complexity.

Bagheri [89] showed that a pseudo-collision of JH can easily be found, since JH's compression function is invertible. This type of attack can be applied to sponge or sponge-like constructions, such as Keccak, CubeHash, Luffa, and Fugue. However, since the IV of JH is a fixed constant that is determined by the hash output size, the pseudo-collision attack is not considered to be a serious threat to JH.

Rijmen et al. [90] proposed a semi-free-start collision attack for sixteen rounds with 2^{190} time and 2^{104} memory complexity, and a semi-free-start near-collision attack for twenty-two rounds with 2^{168} time and $2^{143.70}$ memory complexity. Moreover, Naya-Plasencia [70] proposed a semi-free-start collision attack for sixteen rounds with $2^{96.12}$ time and $2^{96.12}$ memory complexity, and a semi-free-start near-collision attack for twenty-two rounds with $2^{95.63}$ time and $2^{95.63}$ memory complexity.

4.8.2 Performance

JH is an average to above-average performer in software across most platforms for long messages. The performance of JH is not affected by the size of the message digest. JH takes considerable advantage of vector-unit instructions to increase its performance. The computation of 512-bit digests is optimized for 64-bit processors, but performs less well when implemented on 32-bit platforms. In constrained environments, JH is average in performance, below-average in

ROM usage, and requires relatively less RAM. In hardware, JH is average in throughput-to-area ratio for 256-bit digests and above-average for 512-bit digests.

4.8.3 Discussion

JH was selected as a finalist because of its solid security margin, good all-around performance, and innovative design.

4.9 Keccak

Keccak follows the sponge construction model [91]. The permutation can be considered as a substitution-permutation network with five-bit wide S-boxes, or as a combination of a linear mixing operation and a very simple nonlinear mixing operation. The construction of the permutation is the most innovative part of the Keccak design. The recommended security parameter for Keccak is twenty-four rounds, which was tweaked from eighteen rounds in the original submission. Additionally, the message block size was increased as the part of the first round tweak.

4.9.1 Security

Keccak has received a fair amount of attention from cryptanalysts, but at this time, none of the results seem to seriously question the security of the design.

Aumasson and Khovratovich [92] attempted algebraic attacks against significantly reduced-round versions of Keccak by applying automatic cryptanalysis tools (a triangulation tool, and cube testers) to Keccak's permutation. The analysis was limited to the three- and four-round versions of Keccak, but the authors conjectured that structures may be observed in the permutation for up to ten rounds. The authors of [92] concluded that Keccak has a comfortable security margin to prevent structural distinguishers.

Morawiecki and Srebrny [93] found preimages for the three-round Keccak with forty unknown message bits. This is of mild, theoretical interest only.

Bertoni et al. [94] argued that the cost of implementing Keccak in a way that is resistant to power analysis is low. Assessing the significance of this work would be premature, since the same exercise has not been done for all SHA-3 candidates.

Aumasson and Meier [95] first showed practical distinguishers on a nine-round permutation and impractical distinguishers on a sixteen round variant. Boura and Canteaut [96] later extended this to eighteen rounds. However, at a complexity of 2^{1370} this attack has only theoretical interest. The Keccak team increased the number of rounds to twenty-four after the initial distinguisher was announced. Responses to [95] and [96] from the Keccak team were provided in [97, 98].

Lathrop [99] was able to use a cube attack to conduct a key-recovery attack on a secret prefix MAC with a four-round Keccak variant. This attack does not seem to affect the security of Keccak.

4.9.2 Performance

Keccak is an average performer in software across most platforms for long messages. In constrained environments, Keccak is below-average in performance, average in ROM usage, and below-average in RAM usage. In hardware, Keccak is among the top performers in throughput-to-area ratio (especially for the 256-bit variant). The area usage is average, but the throughput is outstanding, because of the very high level of parallelization inherent in the design.

4.9.3 Discussion

Keccak was selected as a finalist, mainly due to its high security margin, its high throughput and throughput-to-area ratio and the simplicity of its design.

4.10 Luffa

Luffa is a wide-pipe, sponge-variant design based on a family of 256-bit nonlinear permutations. Message blocks are inserted and mixed into the state through the linear message injection map *MI* with a high branch number. The nonlinear permutation *P*, which is divided into *w* parallel sub-permutations Q_j, is applied to the state. The number of sub-permutations depends on the hash output size, i.e. w=3 for Luffa-224 and -256, w=4 for Luffa-384, and w=5 for Luffa-512. The message injection and the permutation *P* form the round function. To differentiate between sub-permutations, different rotation operations are applied. The step function, composed of four-bit S-boxes, four rounds of a Feistel ladder and constant addition, is then applied eight times. After processing all message blocks, a blank round is applied using a zero message block, and the outputs of the Q_j 's are XORed to generate the hash value. The nonlinearity of the design is achieved by the S-boxes.

4.10.1 Security

Jia [100] presented pseudo-collision, pseudo-second-preimage and pseudo-preimage attacks on all variants of Luffa. For Luffa 224/256, the attacks are trivial, since the function is invertible after fixing the value of one of the inputs. Although the attacks for Luffa 384/512 were not practical, due to their time complexity, the authors of [100] proposed using Wagner's generalized birthday attack to reduce the time complexity by using some memory.

Watanabe et al. [101] discussed the resistance of Luffa against higher-order differential attacks, and observed that the algebraic degree of the sub-permutations grows more slowly than the ideal case. This enables the attackers to distinguish Luffa in seven steps, instead of eight steps using 2^{216} messages. To increase the algebraic degree of the sub-permutations, the designers suggested three changes in the design:

- applying a blank round for all message sizes,
- changing the S-box, and
- changing the order of the inputs in the sub-permutation.

The tweaked version is called Luffa *v2*. In [102], a similar method to [101] is applied to the Luffa *v2* compression function to find higher-order differentials.

Khovratovich and Naya-Plasencia [103] applied a modified rebound-attack to find a semi-free-start collision for Luffa reduced to seven rounds with 2^{104} time and 2^{102} memory complexity. The attack was extended to a differential distinguisher for the full permutation of Luffa *v2*.

Preneel et al. [104] presented a collision attack on four-round Luffa with complexity 2^{90}, and extended the attack to five rounds with complexity 2^{224}, showing the security margin of the design against differential collision search with message modification.

4.10.2 Performance

Luffa is an above-average performer in software across most platforms for both long and short messages. In constrained environments, Luffa has above-average performance, and requires relatively little ROM and RAM. In hardware, Luffa is among the top performers and has a flexible design that allows for a variety of area vs. speed tradeoffs.

4.10.3 Discussion

Although Luffa is an above-average performer, especially in hardware, it has not received as much cryptanalysis as many of the other second-round candidates. The little cryptanalysis there has been on Luffa, however, raises significant questions about its security; the security margin of the compression function is quite small, and full distinguishers on the sub-permutations have also been discovered. In addition, there is no published security proof showing that even strong sub-permutations would lead to a strong hash function. Therefore, Luffa was not selected as a finalist.

4.11 Shabal

Shabal uses a novel mode of operation that can be considered as a variant of a wide-pipe, Merkle-Damgård hash construction. The internal state of Shabal consists of three parts, denoted as *A*, *B*, and *C*. The keyed permutation of Shabal updates *A* and *B* using nonlinear feedback shift registers that interact with each other. The main loop of the permutation uses modular multiplication by three and five, modular addition, XOR, complementation, and AND operations.

The chaining mode of Shabal works as follows

$$(A, B) \leftarrow P_{M,C}(A \oplus W, B + M),$$
$$(A, B, C) \leftarrow (A, C - M, B),$$

where *M* is the message block, and *W* is the counter. After processing all message blocks, three finalization rounds are applied in which the message block and the counter values are fixed.

Two tunable parameters (*p, r*) are defined for Shabal, where *p* is the number of loops performed within the key permutation, and *r* is the size of *A*. The default value of (*p, r*) is (3, 12). Additionally, *p* and *r* should satisfy $16p \equiv 0 \bmod r$.

The same internal function is used for all output sizes of Shabal.

4.11.1 Security

Various distinguishers have been proposed for the permutation of Shabal. Using cube testers, Aumasson [105] described a statistical non-randomness property of Shabals' key permutation with time complexity 2^{300}.

Assche [106] presented a rotational distinguisher to the keyed permutation of Shabal using 2^{171} calls. The distinguisher is based on the observation that most of the operations in *P* preserve rotational relationships between inputs. However, the attack cannot be extended to the hash algorithm, due to the non-symmetric IV, the addition of the block counter and the existence of the finalization rounds.

Aumasson et al. [107] presented another distinguisher, based on the observation that multiplication by three preserves the difference in the most significant bit with probability one. Using this observation, the authors presented a method to find semi-equivalent keys for the keyed permutation. Under the related-key model, the authors also presented a method that distinguishes *P* from a random permutation using a single query. The method can be generalized to any security parameter. The authors also presented a method to find pseudo-collisions and pseudo-second-preimages for a variant of Shabal in which the number of iterations in the finalization is $24N$ ($N \geq 2$), instead of 36. However, this attack does not work for the original Shabal, since it is not possible to cancel out the differences when the number of iterations is 36.

Knudsen et al. [108] showed some non-randomness properties for the keyed permutation *P* that are easy to verify. The authors proposed a simple method to find fixed points in *P*. The inputs to the permutation were selected so that they are not affected by the loops in the permutation. The method is independent of the number of words in *A* and the number of rounds; therefore, the method works for any choice of the tunable security parameters. The authors also showed a way to construct many, large multi-collisions, where the only difference is in the key input.

Novotney [109] presented a distinguisher over some of the biased output bits using differential analysis with 2^{23} data complexity.

In [110], a low weight (45-bit) pseudo-collision attack on the Shabal compression function with time complexity 2^{84} was presented. The paper also included a preimage attack with 2^{497} time and 2^{400} memory complexity for Shabal 512 using security parameters (2, 12).

None of the distinguishers directly threatens the claimed security of the hash algorithm. Moreover, the designers have modified the indifferentiability security proof of their chaining mode to require weaker assumptions than ideal ciphers [111].

4.11.2 Performance

Shabal is among the best performers in software across most platforms for long messages. It loses some of its advantage over a few algorithms for extremely short messages. In constrained environments, Shabal is among the top performers in speed, and requires low amounts of ROM and RAM. In hardware, Shabal provides average throughput-to-area ratio performance. Additionally, Shabal can be implemented in a very compact area.

4.11.3 Discussion

Shabal uses a novel design for its mode of iteration and basic building blocks. It was not, however, selected as a finalist, mainly due to security concerns. Although the security of the full hash algorithm was not compromised, the discovery of non-randomness properties with low time complexities raised concerns among NIST's cryptographers about the possibility of more powerful attacks in the future. While Shabal's security proof was modified to take known attacks into account, it did not convince NIST's cryptographers that Shabal would remain secure against future attacks.

4.12 SHAvite-3

Shavite-3 is a HAIFA hash algorithm. The compression function is a block cipher used in the Davies-Meyer mode with an additional counter injected into that compression function. The key for the underlying block cipher is formed by the message block, the counter and the salt value, while the plaintext for the block cipher is the previous chaining value. The block cipher is a balanced Feistel network (when n=224 or 256) or a pair of interleaved, balanced Feistel networks (when n=384 or 512), with the *F*-function constructed from the multiple applications of the AES round function. The key schedule of the SHAvite-3 block cipher can be considered as the composition of *(i)* a non-linear layer through AES S-boxes, and *(ii)* a linear layer similar to a linear feedback shift register transformation. All nonlinearity in the entire construction, thus, relies upon the AES S-box.

SHAvite-3-224/256 has twelve rounds, while SHAvite-3-384/512 has fourteen; the number of rounds is the tunable parameter of SHAvite-3.

4.12.1 Security

The most serious attack on the original SHAvite-3 was demonstrated by Peyrin [112], who discovered fixed points for the underlying block ciphers with a chosen salt and chosen counter. Nandi and Paul subsequently extended the Peyrin-attack to a pseudo-multi-collision attack on the compression function with a practical complexity under the same conditions [113]. In response to these attacks, the SHAvite-3 hash algorithms for the second round included tweaks that complemented a part of the counter value.

The second-round tweak in the counter value was able to repair the weaknesses of SHAvite-3-256, but SHAvite-3-512 is still susceptible to some attacks, although at some extra costs. Gauravaram et al. showed that the fulfillment of a few conditions occurring with non-trivial probabilities in the key schedule of the block cipher is sufficient to mount pseudo-preimage and pseudo-collision attacks on the SHAvite-3-512 compression function [114]. Thus, a pseudo-preimage attack based on cancellations of values among the interleaving branches is effective with 2^{384} compression function calls and 2^{128} memory complexity. A pseudo-collision attack is also possible following a similar method as before, with 2^{192} compression function calls and 2^{128} memory complexity. The above cancellation property leads to a second-preimage attack on the SHAvite-3-512 hash algorithm reduced to ten rounds (out of fourteen rounds) as an application of a generic, unbalanced meet-in-the-middle attack, with 2^{497} compression function calls and 2^{16} memory complexity. This attack is an improvement over a previous attack on the SHAvite-3-512 hash algorithm reduced to nine out of fourteen rounds [115].

The attacks on the tweaked SHAvite-3-512 raised concerns again over the security of the key schedule of the underlying block cipher.

4.12.2 Performance

SHAvite-3 is an average to below-average performer in software across most platforms for long messages when producing 256-bit digests, and performance is reduced further for 512-bit digests. However, SHAvite-3 has good performance with the use of the AES-NI instructions. SHAvite-3 has average performance when it is able to fit on devices with constrained environments, but requires a significant amount of ROM. In hardware, SHAvite-3 has an average to below-average throughput-to-area ratio.

4.12.3 Discussion

SHAvite-3 was not selected as a finalist, due to the lack of security in the key schedule of the underlying block cipher, leading to a relatively low security margin for the 512-bit version of the hash function. In addition, SHAvite-3 has a relatively low throughput-to-area ratio.

4.13 SIMD

SIMD's design principle is, to some extent, similar to those of the MD/SHA family of hash algorithms in that *(i)* they all follow the unbalanced Feistel structure, *(ii)* all of them use block-cipher-based Davies-Meyer type modes of operation, and *(iii)* all of them are built on similar basic operations, such as modular addition, cyclic rotation, bitwise XOR, OR and AND operations.

However, there are major differences between SIMD and the MD/SHA hash algorithms. The first notable difference is that the SIMD chaining value is wide-pipe, compared to the single-pipe chaining value of MD/SHA; therefore, SIMD is able to rule out many generic attacks – such as the multi-collision attack – that are applicable only to the single-pipe hash algorithms. Another

important difference between SIMD and MD/SHA is that SIMD, unlike MD/SHA, uses a modified version of the Davies-Meyer mode, where the feed-forward chaining value is combined with the block cipher output through a keyed permutation, rather than just being XORed with it. Such modification makes many attacks difficult, particularly those based on fixed-points and expandable messages. However, the most distinctive property of SIMD is its parallel execution of several Feistel ladders (four for SIMD-224/256, and eight for SIMD-384/512), making it suitable for efficient implementation with the SIMD instructions that are available on some computers.

The novel SIMD message expansion uses a number-theoretic transform, NTT, to implement a code with high minimal distance. The message expansion of the compression function is the most expensive part of the hash computation.

The original SIMD submission was modified in response to a differential attack by Mendel and Nad [116]. The tweak changed the permutations and also the rotation constants to invigorate diffusion among the parallel Feistel ladders.

4.13.1 Security

The SIMD hash algorithm has not been threatened much so far. However, a few non-trivial weaknesses of several components of the hash algorithm were discovered. Using a modular differential path, free-start near-collisions were found on a reduced-round version of SIMD-256 and SIMD-512 [117]. In addition, a distinguishing attack with high complexity, though better than brute-force, was reported [117]. Very recently, classes of the symmetric states of SIMD have been discovered that can be easily translated into distinguishing attacks on the compression function [118].

4.13.2 Performance

The SIMD hash algorithm requires the use of the SIMD instructions for competitive software performance, but with a vector unit is an above-average performer in software across most platforms for long messages. In constrained environments, SIMD has poor performance – possibly due to a lack of optimization effort. SIMD also requires significant amounts of RAM. In hardware, SIMD is a very poor performer in throughput-to-area ratio. This is largely due to its extremely large area requirements. SIMD implementations can be too large to fit on small devices.

4.13.3 Discussion

SIMD was not selected as a finalist, due to its large area requirements and the existence of the symmetric states.

4.14 Skein

Skein is an iterative hash algorithm built on a tweakable block cipher – Threefish. Threefish is used to build the compression function of Skein using a modified Mateas-Meyer-Oseas construction, which is then iterated in a chaining mode similar to HAIFA. The designers refer to the whole construction as a Unique Block Iteration (UBI) mode. The UBI mode provides indifferentiability from a random oracle in the ideal cipher model [119]. Threefish is a 72-round substitution-permutation network using a 128-bit MIX function consisting of a 64-bit addition, rotate and XOR operations.

For the second round, Skein was tweaked by modifying the rotation constants, which the submitters felt had not been fully optimized for the first round.

4.14.1 Security

The most severe attacks on Skein, thus far, exploited the near-rotational symmetries of its MIX function (where only the lack of a carry bit at either end of the 64-bit adder breaks the symmetry.) The first of these attacks was the related-key, key-recovery attack of Khovratovich and Nikolic [120] on 39 rounds of Threefish-256 and 42 rounds of Threefish-512. Later, Khovratovich et al. [121] provided a rotational distinguisher, based on the concept of rotational collisions. By using what the authors described as a rebound attack, they extended the distinguisher to 53 and 57 rounds of the Skein-256 and Skein-512 compression functions, respectively.

McKay and Vora [122] published an attack on Threefish, using pseudo-linear functions to analyze the block cipher, and yielding a fifteen-round key recovery attack. In addition, linear differential analysis by Su et al. [41] produced near-collisions on twenty-four-rounds of the Skein compression function.

Attacks on the first-round version of Threefish included a thirty-three-round key recovery by Chen and Jia [123], and a thirty-five-round distinguisher by Aumasson et al. [124].

4.14.2 Performance

Skein is an above-average performer in software across most platforms that implement 64-bit operations, without requiring the use of a vector unit. Skein is optimized for 64-bit processors, placing it in the top tier of the submitted algorithms, but performance is reduced to slightly above-average when implemented on 32-bit processors. In constrained environments, Skein has above-average performance, and requires relatively less ROM, and average RAM. In hardware, Skein's throughput-to-area ratio is average to a little below-average.

4.14.3 Discussion

Skein was selected as a finalist, mainly due to its high security margin and speed in software.

5. Conclusion and the Final Round

The announcement of the five SHA-3 finalists – BLAKE, Grøstl, JH, Keccak and Skein marked the end of the second round of the SHA-3 competition. This report summarizes the evaluation criteria, the selection process, the designs of the second-round candidates and the published security and performance results for each.

The submitters of the finalist algorithms were allowed to tweak their algorithms, and submit the final packages to NIST by January 16, 2011. A one-year public comment period is planned for these candidates. NIST plans to host the final SHA-3 Candidate Conference in the spring of 2012 to discuss the feedback on the finalists, and select the SHA-3 winner later in 2012.

References[3]

[1] FIPS PUB 180-3, Secure Hash Standard (SHS), Information Technology Laboratory, National Institute of Standards and Technology, Oct. 2008, http://csrc.nist.gov/publications/fips/fips180-3/fips180-3_final.pdf

[2] X. Wang, Y. L. Yin, H. Yu, "*Collision Search Attacks on SHA-1*", 2005, http://www.c4i.org/erehwon/shanote.pdf

[3] Announcing the Development of New Hash Algorithm(s) for the Revision of Federal Information Processing Standard (FIPS) 180-2, Secure Hash Standard, Federal Register / Vol. 72, No. 14 / Tuesday, January 23, 2007 / Notices 2861, http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Jan07.pdf

[4] Announcing Request for Candidate Algorithm Nominations for a New Cryptographic Hash Algorithm (SHA-3) Family, Federal Register/ Vol. 72, No. 212 / Friday, November 2, 2007 / Notices 62212, http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Nov07.pdf

[5] A. Regenscheid, R. Perlner, S. Chang, J. Kelsey, M. Nandi, S. Paul, "*Status Report on the First Round of the SHA-3 Cryptographic Hash Algorithm Competition*", NIST IR 7620, http://csrc.nist.gov/groups/ST/hash/sha-3/Round1/documents/sha3_NISTIR7620.pdf

[6] The Second SHA-3 Candidate Conference, UCSB, CA, August 23-24, 2010, http://csrc.nist.gov/groups/ST/hash/sha-3/Round2/Aug2010/

[7] J.P. Aumasson, L. Henzen, W. Meier, R. C.-W. Phan, "*SHA-3 proposal BLAKE*", Submission to NIST (Round 2), 2009, http://csrc.nist.gov/groups/ST/hash/sha-3/Round2/submissions_rnd2.html

[8] D. Gligoroski, V. Klima, S. J. Knapskog, M. El-Hadedy, J. Amundsen, S. F. Mjolsnes, "*Cryptographic Hash Function BLUE MIDNIGHT WISH*", Submission to NIST (Round 2), 2009, http://csrc.nist.gov/groups/ST/hash/sha-3/Round2/submissions_rnd2.html

[9] D. J. Bernstein, "*CubeHash specification*", Submission to NIST (Round 2), 2009, http://csrc.nist.gov/groups/ST/hash/sha-3/Round2/submissions_rnd2.html

[10] R. Benadjila, O. Billet, H. Gilbert, G. Macario-Rat, T. Peyrin, M. Robshaw, Y. Seurin, "*SHA-3 Proposal: ECHO*", Submission to NIST (Round 2), 2009, http://csrc.nist.gov/groups/ST/hash/sha-3/Round2/submissions_rnd2.html

[11] S. Halevi, W. E. Hall, C. S. Jutla, "*The Hash Function Fugue*", Submission to NIST (Round 2), 2009, http://csrc.nist.gov/groups/ST/hash/sha-3/Round2/submissions_rnd2.html

[12] P. Gauravaram, L. R. Knudsen, K. Matusiewicz, F. Mendel, C. Rechberger, M. Schläffer, S. S. Thomsen, "*Grøstl A SHA-3 Candidate*", Submission to NIST (Round 2), 2009, http://csrc.nist.gov/groups/ST/hash/sha-3/Round2/submissions_rnd2.html

[13] Ö. Küçük, "*The Hash Function Hamsi*", Submission to NIST (Round 2), 2009, http://csrc.nist.gov/groups/ST/hash/sha-3/Round2/submissions_rnd2.html

[14] H. Wu, "*The Hash Function JH*", Submission to NIST (Round 2), 2009, http://csrc.nist.gov/groups/ST/hash/sha-3/Round2/submissions_rnd2.html

[15] G. Bertoni, J. Daemen, M. Peeters, G. V. Assche, "*Keccak Specifications*", Submission to NIST (Round 2), 2009,

[3] A number of the references below are to online resources provided by individuals or organizations that are not affiliated with NIST. While some effort was made to provide links to resources that NIST expects to remain available over the long term, this was not always possible. In these cases, NIST felt that even temporarily available online resources were of value.

http://csrc.nist.gov/groups/ST/hash/sha-3/Round2/submissions_rnd2.html

[16] C. D. Cannière, H. Sato, D. Watanabe, "*Hash Function Luffa: Specification*", Submission to NIST (Round 2), 2009, http://csrc.nist.gov/groups/ST/hash/sha-3/Round2/submissions_rnd2.html

[17] E. Bresson, A. Canteaut, B. Chevallier-Mames, C. Clavier, T. Fuhr, A. Gouget, T. Icart, J.-F. Misarsky, M. Naya-Plasencia, P. Paillier, T. Pornin, J.-R. Reinhard, C. Thuillet, M. Videau, "*Shabal, A Submission to NIST's Cryptographic Hash Algorithm Competition*", Submission to NIST (Round 2), 2009, http://csrc.nist.gov/groups/ST/hash/sha-3/Round2/submissions_rnd2.html

[18] E. Biham, O. Dunkelman, "*The SHAvite-3 Hash Function*", Submission to NIST (Round 2), 2009, http://csrc.nist.gov/groups/ST/hash/sha-3/Round2/submissions_rnd2.html

[19] G. Leurent, C. Bouillaguet, P.-A. Fouque, "*SIMD is a Message Digest*", Submission to NIST (Round 2), 2009, http://csrc.nist.gov/groups/ST/hash/sha-3/Round2/submissions_rnd2.html

[20] N. Ferguson, S. Lucks, B. Schneier, D. Whiting, M. Bellare, T. Kohno, J. Callas, J. Walker, "*The Skein Hash Function Family*", Submission to NIST (Round 2), 2009, http://csrc.nist.gov/groups/ST/hash/sha-3/Round2/submissions_rnd2.html

[21] The First SHA-3 Candidate Conference, Katholieke Universiteit Leuven, Belgium, February 25-28, 2009, http://csrc.nist.gov/groups/ST/hash/sha-3/Round1/Feb2009/index.html

[22] ECRYPT SHA-3 Zoo, http://ehash.iaik.tugraz.at/wiki/The_SHA-3_Zoo

[23] eBASH: ECRYPT Benchmarking of All Submitted Hashes, http://bench.cr.yp.to/ebash.html

[24] External Benchmarking Extension (XBX), http://xbx.das-labor.org/trac

[25] T. Pornin, "*Comparative Performance Review of the SHA-3 Second-Round Candidates*", the Second SHA-3 Candidate Conference, UCSB, CA, 2010, http://csrc.nist.gov/groups/ST/hash/sha-3/Round2/Aug2010/documents/papers/Pornin_report-sphlib-tp-final.pdf

[26] J. W. Bos, D. Stefan, "*Performance Analysis of the SHA-3 Candidates on Exotic Multi-Core Architectures*", Cryptographic Hardware and Embedded Systems - CHES 2010, vol. 6225 of LNCS, pp. 279-293 Springer, 2010, http://www.springerlink.com/content/9p48014n967455r7/

[27] K. Gaj, E. Homsirikamol, and M. Rogawski, "*Fair and Comprehensive Methodology for Comparing Hardware Performance of Fourteen Round Two SHA-3 Candidates using FPGAs*", Cryptographic Hardware and Embedded Systems - CHES 2010, vol. 6225 of LNCS, pp. 264-278, Springer, 2010, http://portal.acm.org/citation.cfm?id=1881535

[28] S. Matsuo, M. Knezevic, P. Schaumont, I. Verbauwhede, A. Satoh, K. Sakiyama, K. Ota, "*How Can We Conduct Fair and Consistent Hardware Evaluation for SHA-3 Candidates*?", the Second SHA-3 Candidate Conference, UCSB, CA, 2010, http://csrc.nist.gov/groups/ST/hash/sha-3/Round2/Aug2010/documents/papers/MATSUO_SHA-3_Criteria_Hardware_revised.pdf

[29] A. H. Namin and M. A. Hasan, "*Hardware Implementation of the Compression Function for Selected SHA-3 Candidates*", CACR, 2009, http://www.vlsi.uwaterloo.ca/~ahasan/web_papers/technical_reports/web_five_SHA_3.pdf

[30] K. Kobayashi, J. Ikegami, S. Matsuo, K. Sakiyama, and K. Ohta, "*Evaluation of Hardware Performance for The SHA-3 Candidates using SASEBO-GII*", Cryptology ePrint Archive, Report 2010/010, http://eprint.iacr.org/2010/010.pdf

[31] 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*", the Second SHA-3

Candidate Conference, UCSB, CA, 2010, http://csrc.nist.gov/groups/ST/hash/sha-3/Round2/Aug2010/documents/papers/BALDWIN_FPGA_SHA3.pdf

[32] X. Guo, S. Huang, L. Nazhandali, and P. Schaumont, "*Fair and Comprehensive Performance Evaluation of 14 Second Round SHA-3 ASIC Implementations*", the Second SHA-3 Candidate Conference, UCSB, CA, 2010, http://csrc.nist.gov/groups/ST/hash/sha-3/Round2/Aug2010/documents/papers/SCHAUMONT_SHA3.pdf

[33] ATHENa Project, 2010, http://cryptography.gmu.edu/athena/

[34] E. Homsirikamol, M. Rogawski, K. Gaj, "*Comparing Hardware Performance of Fourteen Round Two SHA-3 Candidates using FPGAs*", Cryptology ePrint Archive, Report 2010/445, http://eprint.iacr.org/2010/ 445.pdf

[35] D. J. Bernstein. "*ChaCha, A Variant of Salsa20*", http://cr.yp.to/chacha.html

[36] J. Li, L. Xu, "*Attacks on Round-reduced BLAKE"*, Cryptology ePrint Archive, Report 2009/238, http://eprint.iacr.org/2009/238.pdf

[37] J. Guo, K. Matusiewicz, "*Round-reduced Near-Collisions of BLAKE-32*", WEWoRC 2009, http://www1.spms.ntu.edu.sg/~guojian/doc/blake-col.pdf

[38] J.P. Aumasson, J. Guo, S. Knellwolf, K. Matusiewicz, W. Meier, "*Differential and Invertibility Properties of BLAKE*", Cryptology ePrint Archive, report 2010/043, http://eprint.iacr.org/2010/043.pdf

[39] M. Sönmez Turan, E. Uyan, "*Practical Near-Collisions for Reduced Round Blake, Fugue, Hamsi and JH*", the Second SHA-3 Candidate Conference, UCSB, CA, 2010, http://csrc.nist.gov/groups/ST/hash/sha-3/Round2/Aug2010/documents/papers/TURAN_Paper_Erdener.pdf

[40] J. Vidali, P. Nose, E. Pašalic, "*Collisions for Variants of the BLAKE Hash Function*", Proceedings of Information Processing Letters, vol. 110, Issue 14-15, 2010, http://lkrv.fri.uni-lj.si/~janos/blake/collisions.pdf

[41] B. Su, W. Wu, S. Wu, L. Dong, "*Near-Collisions on the Reduced-Round Compression Functions of Skein and BLAKE*", Cryptology ePrint Archive, Report 2010/355, http://eprint.iacr.org/2010/355.pdf

[42] S. S. Thomsen, "*A Near-Collision on the Blue Midnight Wish Compression Function*", 2008, http://www2.mat.dtu.dk/people/S.Thomsen/bmw/nc-compress.pdf

[43] S. S. Thomsen, "*Pseudo-cryptanalysis of the Original Blue Midnight Wish*", in S. Hong and T. Iwata, editors, Fast Software Encryption 2010, Proceedings, vol. 6147 of LNCS, pages 304-317. Springer, 2010, http://www.springerlink.com/content/05663v32l1k34181/

[44] J.P. Aumasson, "*Practical Distinguisher for the Compression Function of Blue Midnight Wish*", 2010, http://www.131002.net/data/papers/Aum10.pdf

[45] J. Guo and S. S. Thomsen, "*Distinguishers for the Compression Function of Blue Midnight Wish with Probability 1*", 2010, http://www2.mat.dtu.dk/people/S.Thomsen/bmw/bmw-distinguishers.pdf

[46] I. Nikolić, J. Pieprzyk, P. Sokołowski, R. Steinfeld, "*Rotational Cryptanalysis of (Modified) Versions of BMW and SIMD*", 2010. https://cryptolux.org/mediawiki/uploads/0/07/Rotational_distinguishers_(Nikolic,_Pieprzyk,_Sokolowski,_Steinfeld).pdf

[47] G. Leurent, "*Self-Defence against Fresh Fruit*", Crypto 2010 Rump Session, August, 17, 2010, http://rump2010.cr.yp.to/c659ebaf681758e01ccf824fd58f3c42.pdf

[48] G. Leurent and S. Thomsen, "*Practical Partial-Collisions on the Compression Function of BMW*", 2010, http://www.di.ens.fr/~leurent/files/BMW_Distinguisher.pdf

[49] D. J. Bernstein, "*CubeHash Parameter Tweak: 16 times faster*", 2009, http://cubehash.cr.yp.to/submission/tweak.pdf

[50] D. J. Bernstein, "*CubeHash Appendix: Complexity of Generic Attacks*", http://cubehash.cr.yp.to/submission/generic.pdf

[51] M. Nandi, "*NIST's Views on SHA-3's Security Requirements and Evaluation of Attacks*", the First SHA-3 Candidate Conference, Katholieke Universiteit Leuven, Belgium, 2009, http://csrc.nist.gov/groups/ST/hash/sha-3/Round1/Feb2009/documents/Nandi.pdf

[52] J.P. Aumasson, E. Brier, W. Meier, M. Naya-Plasencia, T. Peyrin, "*Inside the Hypercube*", Cryptology ePrint Archive, Report 2008/486, http://eprint.iacr.org/2008/486.pdf

[53] N. Ferguson, S. Lucks, K. A. McKay, "*Symmetric States and their Structure: Improved Analysis of CubeHash*", Cryptology ePrint Archive, Report 2010/273, http://eprint.iacr.org/2010/273.pdf

[54] G. Leurent, "*Quantum Preimage and Collision Attacks on CubeHash*", Cryptology ePrint Archive, Report 2010/506, http://eprint.iacr.org/2010/506.pdf

[55] A. Joux, "*Multicollisions in Iterated Hash Functions. Application to Cascaded Constructions*", Matthew K. Franklin (Ed.): Advances in Cryptology- CRYPTO 2004, 24th Annual International Cryptology Conference, Santa Barbara, CA, August 15-19, 2004, Proceedings, LNCS vol. 3152, Springer pp. 306-316, 2004, http://www.iacr.org/cryptodb/archive/2004/CRYPTO/1472/1472.pdf

[56] E. Brier, S. Khazaei, W. Meier, T. Peyrin, "*Linearization Framework for Collision Attacks: Application to CubeHash and MD6*", Cryptology ePrint Archive, Report 2009/382, http://eprint.iacr.org/2009/382.pdf

[57] F. Chabaud, A. Joux, "*Differential Collisions in SHA-0*", Crypto'98 LNCS vol. 1462, pp. 56-71, 1998, http://www.springerlink.com/content/p795v6nj1vj525kp/

[58] S. Khazaei, S. Knellwolf, W. Meier, D. Stefan, "*Improved Linear Differential Attacks on CubeHash*", Africacrypt, 2010, http://www.ee.cooper.edu/~stefan/pubs/conference/africacrypt2010.pdf

[59] T. Ashur, O. Dunkelman, "*Linear Analysis of Reduced-Round CubeHash*", Cryptology ePrint Archive, Report 2010/535, http://eprint.iacr.org/2010/535.pdf

[60] W. Dai, "*Collisions for CubeHash1/45 and CubeHash2/89*", 2008, http://www.cryptopp.com/sha3/cubehash.pdf

[61] J.P. Aumasson, "*Collision for CubeHash2/120-512*", NIST mailing list, 2008-12-04, http://ehash.iaik.tugraz.at/uploads/a/a9/Cubehash.txt

[62] D. Khovratovich, I. Nikolic, R.-P. Weinmann, "*Preimage attack on CubeHash512-r/4 and CubeHash512-r/8*", 2008, http://ehash.iaik.tugraz.at/uploads/6/6c/Cubehash.pdf

[63] F. Mendel, T. Peyrin, C. Rechberger, M. Schläffer, "*Improved Cryptanalysis of the Reduced Grøstl Compression Function, ECHO Permutation and AES Block Cipher*", Proceedings of SAC, 5867, pp. 16-35, 2009, https://online.tugraz.at/tug_online/voe_main2.getvolltext?pCurrPk=44420

[64] H. Gilbert, T. Peyrin, "*Super-Sbox Cryptanalysis: Improved Attacks for AES-like permutations*", Cryptology ePrint Archive, Report 2009/531, http://eprint.iacr.org/2009/531.pdf

[65] T. Peyrin, "*Improved Differential Attacks for ECHO and Grostl*", Cryptology ePrint Archive, Report 2010/223, http://eprint.iacr.org/2010/223.pdf

[66] J. Jean, P.-A. Fouque, "*Practical Near-Collisions and Collisions on Round-Reduced ECHO-256 Compression Function*", Cryptology ePrint Archive, Report 2010/569, http://eprint.iacr.org/2010/569.pdf

[67] M. Schläffer, "*Subspace Distinguisher for 5/8 Rounds of the ECHO-256 Hash Function*", Cryptology ePrint Archive, Report 2010/321,

http://eprint.iacr.org/2010/321.pdf

[68] M. Schläffer, "*Improved Collisions for Reduced ECHO-256*", Cryptology ePrint Archive, Report 2010/588, http://eprint.iacr.org/2010/588.pdf

[69] Y. Sasaki, Y. Li, L. Wang, K. Sakiyama, K. Ohta, "*New Non-Ideal Properties of AES-Based Permutations: Applications to ECHO and Grøstl*", the Second SHA-3 Candidate Conference, UCSB, CA, 2010, http://csrc.nist.gov/groups/ST/hash/sha-3/Round2/Aug2010/documents/papers/SASAKI_ECHOanalysisFinal.pdf

[70] M. Naya-Plasencia, "*Scrutinizing rebound attacks: new algorithms for improving the complexities*", Cryptology ePrint Archive, Report 2010/607, http://eprint.iacr.org/2010/607.pdf

[71] L. R. Knudsen, C. Rechberger, S. S. Thomsen, "*Grindahl - A Family of Hash Functions*", in Biryukov, A. (ed.) Fast Software Encryption - FSE 2007, LNCS, Springer, Heidelberg, 2007, http://www2.mat.dtu.dk/people/Lars.R.Knudsen/grindahl/grindahl.pdf

[72] D. Khovratovich, "*Cryptanalysis of Hash Functions with Structures*", Proceedings of SAC, 2009, https://cryptolux.org/mediawiki/uploads/9/99/Struct2.pdf

[73] J.P. Aumasson, R. C.-W. Phan, "*Distinguisher for Full Final Round of Fugue-256*", the Second SHA-3 Candidate Conference, UCSB, CA, 2010, http://csrc.nist.gov/groups/ST/hash/sha-3/Round2/Aug2010/documents/papers/Aumasson_fugue.pdf

[74] F. Mendel, C. Rechberger, M. Schläffer, S. S. Thomsen, "*The Rebound Attack: Cryptanalysis of Reduced Whirlpool and Grøstl*", Proceedings of FSE, LNCS 5665, pp. 260-276, Springer, 2009, http://www2.mat.dtu.dk/people/S.Thomsen/MendelRST-fse09.pdf

[75] F. Mendel, C. Rechberger, M. Schläffer, S. S. Thomsen, "*Rebound Attacks on the Reduced Grøstl Hash Function*", Proceedings of CT-RSA, LNCS 5985, pp. 350-365, Springer, 2010, http://www.springerlink.com/content/p016144684v473r6/

[76] T. Peyrin, "*Improved Differential Attacks for ECHO and Grøstl*", Cryptology ePrint Archive, Report 2010/223, http://eprint.iacr.org/2010/223.pdf

[77] K. Ideguchi, E. Tischhauser, B. Preneel, "*Improved Collision Attacks on the Reduced-Round Grøstl Hash Function*", Cryptology ePrint Archive, Report 2010/375, http://eprint.iacr.org/2010/375.pdf

[78] J.P. Aumasson, "*On the Pseudorandomness of Hamsi*", NIST mailing list, 2009-02-02, http://cio.nist.gov/esd/emaildir/lists/hash-forum/msg01342.html

[79] I. Nikolic, "*Near Collisions for the Compression Function of Hamsi-256*", CRYPTO 2009 rump session, 2009, http://rump2009.cr.yp.to/936779b3afb9b48a404b487d6865091d.pdf

[80] J. P. Aumasson, E. Käsper, L. R. Knudsen, K. Matusiewicz, R. Ødegaard, T. Peyrin, M. Schläffer, "*Distinguishers for the Compression Function and Output Transformation of Hamsi-256*", Proceedings of ACISP, LNCS 6168, pp. 87-103, Springer, 2010, http://homes.esat.kuleuven.be/~ekasper/papers/hamsi.pdf

[81] M. Wang, X. Wang, K. Jia, W. Wang, "*New Pseudo-Near-Collision Attack on Reduced-Round of Hamsi-256*", Cryptology ePrint Archive, Report 2009/484, http://eprint.iacr.org/2009/484.pdf

[82] Y. Li, A. Wang, "*Near Collisions for the Compress Function of Hamsi-256 Found by Genetic Algorithm*", Cryptology ePrint Archive, Report 2010/423, http://eprint.iacr.org/2010/423.pdf

[83] Ç. Çalık, M. Sönmez Turan, "*Message Recovery and Pseudo-Preimage Attacks on the Compression Function of Hamsi-256*", Cryptology ePrint Archive, Report 2010/057, http://eprint.iacr.org/2010/057.pdf

[84] T. Fuhr, "*Finding Second Preimages of Short Messages for Hamsi-256*", Advances in Cryptology - ASIACRYPT 2010, Proceedings, 2010, http://www.springerlink.com/content/u3r51n7273476077/

[85] I. Dinur, A. Shamir, "*An Improved Algebraic Attack on Hamsi-256*", Cryptology ePrint Archive, Report 2010/602, http://eprint.iacr.org/2010/602.pdf

[86] R. Bhattacharyya, A. Mandal, M. Nandi, "*Security Analysis of the Mode of JH Hash Function*", FSE 2010, LNCS 6147, pp. 168-191, 2010, http://www.isical.ac.in/~rishi_r/FSE2010-146.pdf

[87] F. Mendel, S. S. Thomsen, "*An Observation on JH-512*", 2008, http://ehash.iaik.tugraz.at/uploads/d/da/Jh_preimage.pdf

[88] H. Wu, "*The Complexity of Mendel and Thomsen's Preimage Attack on JH-512*", 2009, http://ehash.iaik.tugraz.at/uploads/6/6f/Jh_mt_complexity.pdf

[89] N. Bagheri, "*Re: Free start collision, second preimage, and Preimage on JH*", NIST Mailing List, 2008-11-29, http://cio.nist.gov/esd/emaildir/lists/hash-forum/msg00981.html

[90] V. Rijmen, D. Toz, and K. Varıcı, "*Rebound Attack on Reduced-Round Versions of JH*", FSE 2010, LNCS 6147, pp. 286-303, 2010, https://www.cosic.esat.kuleuven.be/publications/article-1431.pdf

[91] G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, "*Sponge Functions*", Ecrypt Hash Workshop, 2007, http://sponge.noekeon.org/SpongeFunctions.pdf

[92] J.P. Aumasson, D. Khovratovich, "*First Analysis of Keccak*", 2009, http://www.131002.net/data/papers/AK09.pdf

[93] P. Morawiecki, M. Srebrny, "*A SAT-based Preimage Analysis of Reduced KECCAK Hash Functions*", Cryptology ePrint Archive, Report 2010/285, http://eprint.iacr.org/2010/285.pdf

[94] G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, "*Building Power Analysis Resistant Implementations of Keccak*", the Second SHA-3 Candidate Conference, UCSB, CA, 2010, http://csrc.nist.gov/groups/ST/hash/sha-3/Round2/Aug2010/documents/papers/BERTONI_KeccakAntiDPA.pdf

[95] J.P. Aumasson, W. Meier, "*Zero-sum Distinguishers for Reduced Keccak-f and for the Core Functions of Luffa and Hamsi*", NIST mailing list, 2009-09-09, http://www.131002.net/data/papers/AM09.pdf

[96] C. Boura, A. Canteaut, "*A Zero-Sum Property for the Keccak-f Permutation with 18 Rounds*", NIST mailing list, 2010-01-14, http://www-rocq.inria.fr/secret/Anne.Canteaut/Publications/zero_sum.pdf

[97] G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, "*Note on Zero-sum Distinguishers of Keccak-f*", NIST mailing list, 2010-01-15, http://keccak.noekeon.org/NoteZeroSum.pdf

[98] G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, "*Note on Keccak Parameters and Usage*", NIST mailing list, 2010-02-23, http://keccak.noekeon.org/NoteOnKeccakParametersAndUsage.pdf

[99] J. Lathrop, "*Cube Attacks on Cryptographic Hash Functions*", Master's Thesis, Rochester Institute of Technology, May 21, 2009, https://ritdml.rit.edu/bitstream/handle/1850/10821/20782_pdf_C9905D10-60E8-11DE-AE1A-0950D352ABB1.pdf?sequence=1

[100] K. Jia, "*Pseudo-Collision, Pseudo-Preimage and Pseudo-Second-Preimage Attacks on Luffa*", Cryptology ePrint Archive, Report 2009/224, http://eprint.iacr.org/2009/224.pdf

[101] D. Watanabe, Y. Hatano, T. Yamada, T. Kaneko, "*Higher Order Differential Attack on Step-Reduced Variants of Luffa v1*", Fast Software Encryption - FSE 2010, LNCS, pp. 270-285. Springer, 2010, http://www.springerlink.com/content/e52411jx1611v1t1/

[102] C. Boura, A. Canteaut, C. De Cannière, "*Higher-order Differential Properties of Keccak and Luffa*", Cryptology ePrint Archive, Report 2010/589, http://eprint.iacr.org/2010/589.pdf

[103] D. Khovratovich, M. Naya-Plasencia, A. Röck, M. Schläffer, "*Cryptanalysis of Luffa v2 Components*", Proceedings of SAC, LNCS, Springer, 2010, http://www-rocq.inria.fr/secret/Maria.Naya_Plasencia/papers/luffa.pdf

[104] B. Preneel, H. Yoshida, D. Watanabe, "*Finding Collisions for Reduced Luffa-256 v2*", NIST mailing list, 2010-11-08, http://www.sdl.hitachi.co.jp/crypto/luffa/FindingCollisionsForReducedLuffa-256v2_20101108.pdf

[105] J.P. Aumasson, "*On the Pseudorandomness of Shabal's Keyed Permutation*", 2009, http://www.131002.net/data/papers/Aum09.pdf

[106] G. Van Assche, "*A Rotational Distinguisher on Shabal's Keyed Permutation and Its Impact on the Security Proofs*", 2010, http://gva.noekeon.org/papers/ShabalRotation.pdf

[107] J.P. Aumasson, A. Mashatan, W. Meier, "*More on Shabal's permutation*", 2009, http://www.131002.net/data/papers/AMM09.pdf

[108] L. R. Knudsen, K. Matusiewicz, S. S. Thomsen, "*Observations on the Shabal Keyed Permutation*", OFFICIAL COMMENT to NIST, 2009-04-08, http://www2.mat.dtu.dk/people/S.Thomsen/shabal/shabal.pdf

[109] P. Novotney, "*Distinguisher for Shabal's Permutation Function*", Cryptology ePrint Archive, Report 2010/398, http://eprint.iacr.org/2010/398.pdf

[110] T. Isobe, T. Shirai, "*Low-weight Pseudo Collision Attack on Shabal and Preimage Attack on Reduced Shabal-512*", Cryptology ePrint Archive, Report 2010/434, http://eprint.iacr.org/2010/434.pdf

[111] E. Bresson, A. Canteaut, B. Chevallier-Mames, C. Clavier, T. Fuhr, A. Gouget, T. Icart, J.-F. Misarsky, M. Naya-Plasencia, P. Paillier, T. Pornin, J.-R. Reinhard, C. Thuillet, M. Videau, "*Indifferentiability with Distinguishers: Why Shabal Does Not Require Ideal Ciphers*", Cryptology ePrint Archive, Report 2009/199, http://eprint.iacr.org/2009/199.pdf

[112] T. Peyrin, "*Chosen-salt, Chosen-counter, Pseudo-collision on SHAvite-3 Compression Function*", NIST mailing list, 2009-01-19, http://ehash.iaik.tugraz.at/uploads/e/ea/Peyrin-SHAvite-3.txt

[113] M. Nandi, S. Paul, "*OFFICIAL COMMENT: SHAvite-3*", OFFICIAL COMMENT to NIST, 2009-06-16, http://ehash.iaik.tugraz.at/uploads/5/5c/NandiP-SHAvite-3.txt

[114] P. Gauravaram, G. Leurent, F. Mendel, M. Naya-Plasencia, T. Peyrin, C. Rechberger, M. Schläffer, "*Cryptanalysis of the 10-Round Hash and Full Compression Function of SHAvite-3-512*", Proceedings of Africacrypt, LNCS 6055, pp. 419-436, Springer, 2010, http://www.di.ens.fr/~leurent/files/SHAVITE_AFC10.pdf

[115] C. Bouillaguet, O. Dunkelman, G. Leurent, P.-A. Fouque, "*Attacks on Hash Functions based on Generalized Feistel - Application to Reduced-Round Lesamnta and SHAvite-3-512*", Cryptology ePrint Archive, Report 2009/634, http://eprint.iacr.org/2009/634.pdf

[116] F. Mendel, T. Nad, "*A Distinguisher for the Compression Function of SIMD-512*", Proceedings of INDOCRYPT, LNCS 5922, pp. 219-232, Springer, 2009, http://www.springerlink.com/content/7487g5076g087v48/

[117] H. Yu, X. Wang, "*Cryptanalysis of the Compression Function of SIMD*", Cryptology ePrint Archive, Report 2010/304, http://eprint.iacr.org/2010/304.pdf

[118] C. Bouillaguet, P.-A. Fouque, G. Leurent, "*Security Analysis of SIMD*", Cryptology ePrint Archive, Report 2010/323, http://eprint.iacr.org/2010/323.pdf

[119] M. Bellare, T. Kohno, S. Lucks, N. Ferguson, B. Schneier, D. Whiting, J. Callas, J. Walker, "*Provable Security Support for the Skein Hash Family*", 2009,

http://www.schneier.com/skein-proofs.pdf
[120] D. Khovratovich, I. Nikolic, "*Rotational Cryptanalysis of ARX*", proceedings of FSE2010, pp. 333-346, 2010, http://www.skein-hash.info/sites/default/files/axr.pdf
[121] D. Khovratovich, I. Nikolic, C. Rechberger, "*Rotational Rebound Attacks on Reduced Skein*", Cryptology ePrint Archive, Report 2010/538, http://eprint.iacr.org/2010/538.pdf
[122] K. McKay, P. Vora, "*Pseudo-Linear Approximations for ARX Ciphers: With Application to Threefish*", Cryptology ePrint Archive, Report 2010/282, http://eprint.iacr.org/2010/282.pdf
[123] J. Chen, K. Jia, "*Improved Related-Key Boomerang Attacks on Round-Reduced Threefish-512*", Cryptology ePrint Archive, Report 2009/526, http://eprint.iacr.org/2009/526.pdf
[124] J.P. Aumasson, Ç. Çalık, W. Meier, O. Özen, R. Phan, K. Varıcı, "*Improved Cryptanalysis of Skein*", Cryptology ePrint Archive, Report 2009/438, http://eprint.iacr.org/2009/438.pdf
[125] FIPS PUB 180-2, Secure Hash Standard (SHS), Information Technology Laboratory, National Institute of Standards and Technology, Aug. 2002, http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf
[126] Draft FIPS PUB 180-4, Secure Hash Standard (SHS), Information Technology Laboratory, National Institute of Standards and Technology, Feb. 2011, http://csrc.nist.gov/publications/drafts/fips180-4/Draft-FIPS180-4_Feb2011.pdf
[127] S. Tillich, M. Feldhofer, M. Kirschbaum, T. Plos, J. Schmidt, and A. Szekely, "*Uniform Evaluation of Hardware Implementations of the Round-Two SHA-3 Candidates*", the Second SHA-3 Candidate Conference, UCSB, CA, 2010, http://csrc.nist.gov/groups/ST/hash/sha-3/Round2/Aug2010/documents/papers/TILLICH_sha3hw.pdf