FACTA UNIVERSITATIS (NIŠ)
SER.: ELEC. ENERG. vol. 23, no. 3, December 2010, 357-366

Uniform Logical Cryptanalysis of CubeHash Function

Miodrag Milić and Vojin Šenk

Abstract: In this paper we present results of uniform logical cryptanalysis method applied to cryptographic hash function CubeHash. During the last decade, some of the most popular cryptographic hash functions were broken. Therefore, in 2007, National Institute of Standards and Technology (NIST), announced an international competition for a new Hash Standard called SHA-3. Only 14 candidates passed first two selection rounds and CubeHash is one of them. A great effort is made in their analysis and comparison. Uniform logical cryptanalysis presents an interesting method for this purpose. Universal, adjustable to almost any cryptographic hash function, very fast and reliable, it presents a promising method in the world of cryptanalysis.

Keywords: Cryptography, hash functions, uniform logical cryptanalysis, propositional satisfiability, CubeHash.

1 Introduction

Hash functions have a very important role in modern cryptography, primarily in digital signatures and various forms of authentication. Therefore, a great effort is made in order to enhance their design and analysis methods. The most popular hash functions today are MD5 and SHA-1.

There are many principles used for hash function design. Some of them rely on strong mathematical background, i.e. they have a theoretical foundation that supports claims of their security. Some hash functions rely on well-known and heavily-tested block cipher functions. However, most of them are specially designed using small set of fast operations carefully mixed and iterated over input sequence many times. Nevertheless, all designers have the same goal: to create a function resistant to various attacks, i.e. to make any attack method a computationally hard problem.

Manuscript received on September 9, 2010.

The authors are with Faculty of Technical Sciences, Trg Dositeja Obradovića 6, 21000 Novi Sad, Serbia and Montenegro (e-mails: [milic, vojin_senk]@uns.ac.rs).

NP-complete class is probably the most studied class of computationally hard problems and the most famous problem from this class is Boolean or propositional satisfiability problem (shorthand SAT). This was the first problem proved to be NP-complete [1]. Its significance was further enhanced in the last decade with development of many efficient algorithms for solving certain instances of this problem. Programs developed using these algorithms are known as "SAT-solvers".

Different design methods issue different analysis methods, hence there are many methods of how to analyze cryptographic hash functions. In [2], Massacci and Marraro introduced a new approach to cryptanalysis. Using special handcrafted program, they encoded DES algorithm [3] as SAT problem and used SAT-solver programs to analyze the algorithm. They called it *logical cryptanalysis*. Jovanović and Janičić [4] improved this approach combining useful features of C++ programming language. They designed a framework applicable to any hash algorithm, providing fast and reliable way of hash function analysis. They called it *uniform logical cryptanalysis*.

In 2004, one of the most popular hash functions MD5 [5], as well as several other popular hash functions were broken [6]. Then, in 2005, another very popular function SHA-1 [7] was almost broken [8]. Therefore, in 2007, NIST organized an international competition for a new hash algorithm standard called SHA-3 [9]. There were over 50 candidates, but only 14 remained after two rounds, and one of them is CubeHash algorithm [10]. Analysis, testing and comparing of these algorithms presents one of the most exciting topics among cryptography researchers at the moment. The winner will be announced in 2012.

This paper presents results of uniform logical cryptanalysis method applied on CubeHash algorithm. This method is used to analyze and compare several popular cryptographic hash functions with CubeHash as well as different variants of CubeHash.

2 Cryptographic Hash Functions

A hash function (shortly *hash*) is a function h that maps an input x of arbitrary finite bitlength to an output $h(x)$ of fixed bitlength n. Moreover, it is expected that this transformation is computationally feasible, i.e. easy to compute.

A cryptographic hash function has to satisfy three more requirements:

1. *Preimage resistance* — for given output y it is computationally infeasible to find any preimage x such that $h(x) = y$.
2. *Second preimage resistance* — for given input x it is computationally infeasible to find any other input x' such that $x \neq x'$ and $h(x) = h(x')$.

3. *Collision resistance* — it is computationally infeasible to find any pair of inputs x_1 and x_2 such that $h(x_1) = h(x_2)$.

To meet these requirements, hash function designers use various construction principles. Probably the most popular one is Merkle-Damgård hash construction [11, 12]. Algorithms based on this construction principle perform several steps during input sequence transformation. First, an input sequence is extended to meet certain bitlength requirements and its original bitlength is encoded and appended to the end of extended sequence to prevent trivial attacks. Then, extended sequence is broken up into series of smaller equal-sized blocks. After that, every block is processed using compression function[1] which takes output of a previous processing as its initial state. Finally, after the last input block is processed, the finalization function is performed. Compression functions are mostly specially designed for this purpose like in MD5 or SHA-1 and SHA-2 algorithms [7], but there are also algorithms which use well-known cryptographic block ciphers. Recently, some other construction principles like Merkle tree [13] and sponge construction [14] came into focus.

3 CubeHash Algorithm

CubeHash is a cryptographic hash function created by Daniel J. Bernstein. It is one of 14 candidates submitted to the NIST hash function competition which passed two selection rounds. This algorithm establishes a new construction principle different from mostly used Merkle-Damgård principle. Three parameters determine its performance:

- **r** — number of rounds *transform* function processes each input block
- **b** — number of bytes per input block
- **h** — number of output (hash) bits

CubeHash algorithm maintains a 1024-bit state organized as a 32 4-byte integers $(s[0], s[1], \ldots s[31])$ interpreted in a little-endian form. The core of this algorithm presents function *transform*. Its pseudo-code is shown in Algorithm 1. There are 32 32-bit additions and 32 32-bit exclusive-or operations (*xors*) per round, which makes $32r/b$ 32-bit additions and $32r/b$ 32-bit xors per each byte of input sequence. Obviously, security of this algorithm is increased by increasing **r** and decreasing **b** at the cost of its speed. At the same time, memory consumption is not affected. Recommended values for parameters of CubeHash algorithm, which

[1] Finite state machine with state-transition function dependent on the input sequence. Its next state is its output. It is basic building block of iterative hash functions.

represent trade off between security requirements and speed, are $r = 16$ and $b = 32$.

Algorithm 1: Function *transform*

input : CubeHash state as array of integers $s[32]$
output: changed state

note 1: $rotl(a,b) \Leftrightarrow$ left rotation of variable a for b bit positions
note 2: **r** in the outer loop is CubeHash configuration parameter

integer $t[16]$; `//Local array declaration`

for $i \leftarrow 0$ **to r do**
  for $j \leftarrow 0$ **to** 16 **do** $s[j+16] = s[j+16] + s[j]$;
  for $j \leftarrow 0$ **to** 16 **do** $t[j \oplus 8] = s[j]$;
  for $j \leftarrow 0$ **to** 16 **do** $s[j] = rotl(t[j], 7)$;
  for $j \leftarrow 0$ **to** 16 **do** $s[j] = s[j] \oplus s[j+16]$;
  for $j \leftarrow 0$ **to** 16 **do** $t[j \oplus 2] = s[j+16]$;
  for $j \leftarrow 0$ **to** 16 **do** $s[j+16] = t[j]$;
  for $j \leftarrow 0$ **to** 16 **do** $s[j+16] = s[j+16] + s[j]$;
  for $j \leftarrow 0$ **to** 16 **do** $t[j \oplus 4] = s[j]$;
  for $j \leftarrow 0$ **to** 16 **do** $s[j] = rotl(t[j], 11)$;
  for $j \leftarrow 0$ **to** 16 **do** $s[j] = s[j] \oplus s[j+16]$;
  for $j \leftarrow 0$ **to** 16 **do** $t[j \oplus 1] = s[j+16]$;
  for $j \leftarrow 0$ **to** 16 **do** $s[j+16] = t[j]$;
end

4 Uniform Logical Cryptanalysis

Massacci and Marraro in [2] presented *logical cryptanalysis* method. They encoded some properties of DES algorithm [3] as satisfiability (SAT) problems and used SAT-solver programs to examine those properties. Their method was specially designed for DES algorithm. Later, in [4], Jovanović and Janičić introduced improved method of logical cryptanalysis independent of any particular algorithm and they called it *uniform logical cryptanalysis*.

In order to transform any algorithm related problem or property into SAT problem, one first has to transform the algorithm into propositional formulae and then encode the problem or property as a SAT problem. Satisfiability represents a problem of deciding if there is a truth assignment for the variables of a given propositional formula under which the formula evaluates to true. The opposite problem, i.e. deciding if there is no truth assignment under which the formula evaluates to truth is called unsatisfiability problem. In both cases, propositional

formulae consist only of Boolean variables and logical operations AND, OR and NOT. Moreover, it is common to represent formulae in conjunctive normal form i.e. conjunction of clauses, where clause is a disjunction of literals and literal is a variable or its negation.

4.1 Uniform encoding of Hash functions

The basic rule of modern cryptography is publicity of all its algorithms and protocols, i.e. their strength does not rely on their secrecy. Therefore, every good hash algorithm is publicly available. Moreover, almost all of them are encoded in C programming language because of its performances. The C++ programming language, the successor of C, has some additional properties that make it easy to transform the original program that calculates hash value into program that generates appropriate propositional formulae. To make this transformation possible, one can create a polymorphic [15] user defined type (class) that will represent propositional formulae (*Formula*) and another class that will represent an array of formulae (*ModifiedUInt*). Figure 1 shows diagram of these classes and their relations. Since *ModifiedUInt* class represents array of formulae, every operator

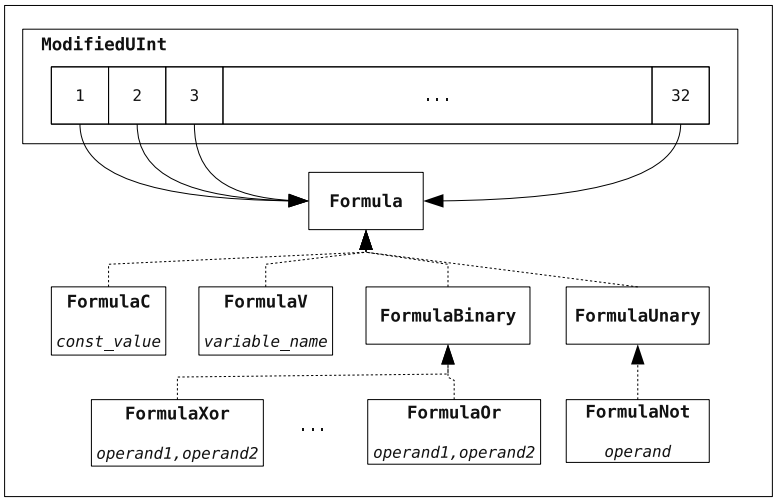

Fig. 1. Relation between *ModifiedUInt* and *Formula* classes.

has to be adapted, i.e. overloaded [15], to be able to generate formulae. To make use of created classes, data types of some variables from the original program have to be replaced with *ModifiedUInt*. After that, simple compiling and running of the program, instead of hash value (array of bits), generates an array of propositional formulae.

4.2 Cryptanalysis to SAT problem transformations

Analysis and comparison of cryptographic hash algorithms is a challenging task. One way to accomplish this task, presented in [2] and [4], is to express a problem using propositional formulae in a conjunctive normal form (CNF) suitable as input for SAT-solver programs. It is shown that the process of generating propositional formulae is feasible, but the process of their transformation into CNF is of exponential complexity. Therefore, a compromise is made using Tseitin's method [16] that transforms a problem given in propositional formulae into SAT-equivalent problem with linear complexity. The result of this transformation is still too complex for modern SAT-solver programs, i.e. we cannot break good hash functions using them. For that reason, attention is focused to number of clauses and literals in CNF of SAT-equivalent problems. It is believed that these numbers can be used as indicators of complexity of observed problems which imply strength of examined hash functions.

In Section 2, hash functions are described as functions that map an input sequence of arbitrary finite bitlength to an output sequence of fixed bitlength. Let input and output sequences be presented as a finite length vectors of Boolean values $\vec{X} = (x_1, x_2, \ldots x_{M-1}, x_M)$ and $\vec{Y} = (y_1, y_2, \ldots y_{N-1}, y_N)$ respectively. Every good cryptographic hash function ensures that every value of the output sequence $\vec{Y}$ is a function of all values of the input, i.e. $\forall n \in [1,N],\ y_n = f_n(x_1, \ldots x_M)$.

After replacing appropriate data types in original algorithm with *ModifiedUInt* class, input and output sequences become vectors of instances of data type *Formula*, i.e. $\vec{X}' = (x'_1, x'_2, \ldots x'_{M-1}, x'_M)$ and $\vec{Y}' = (y'_1, y'_2, \ldots y'_{N-1}, y'_N)$ respectively. Moreover, relation between output and input values becomes $\forall n \in [1,N],\ y'_n = F_n(x'_1, \ldots x'_M)$ where every F_n represents a propositional formula. Now, we have a hash value expressed as a series of propositional formulae.

Next, three main hash cryptanalysis problems will be described in a way suitable for transformation into SAT-equivalent problems.

Preimage resistance of a cryptographic hash function is defined in Section 2. For a given output (hash) value $\vec{C} = (c_1, \ldots c_N)$, where for $\forall n \in [1,N]$ c_n is a constant value (true or false), the problem of preimage resistance Π_1 can be represented as

$$\Pi_1(x'_1, \ldots x'_M) \iff \bigwedge_{n=1}^{N} \left(y'_n = c_n\right),$$

where $y'_n = F_n(x'_1, \ldots x'_M)$ represents propositional formulae of hash value.

Second preimage resistance of a cryptographic hash function is defined in Section 2. For a given first image (input) value $\vec{B} = (b_1 \ldots b_M)$, where $\forall m \in$

$[1,M]$, b_m is a constant value (true or false), and $F_n(b_1,\dots b_M)$ represents propositional formulae of hash value of $\vec{B}$, the problem of second preimage resistance Π_2 can be represented as

$$\Pi_2(x'_1,\dots x'_M) \iff \left(\bigwedge_{n=1}^{N}\left(F_n(x'_1,\dots x'_M) = F_n(b_1,\dots b_M)\right) \wedge \left(\neg \bigwedge_{n=1}^{N}\left(x'_n = b_n\right)\right)\right).$$

Collision resistance of a cryptographic hash function is defined in Section 2. Two input values are given $\vec{X'} = (x'_1 \dots x'_M)$ and $\vec{X''} = (x''_1 \dots x''_M)$. Propositional formulae of their hash values are represented as $F_n(x'_1,\dots x'_M)$ and $F_n(x''_1,\dots x''_M)$ respectively. Problem of collision resistance Π_3 can be represented as

$$\Pi_3(x'_1,\dots x'_M, x''_1,\dots x''_M) \iff \left(\bigwedge_{n=1}^{N}\left(F_n(x'_1,\dots x'_M) = F_n(x''_1,\dots x''_M)\right) \wedge \left(\neg \bigwedge_{n=1}^{N}\left(x'_n = x''_n\right)\right)\right).$$

5 Experimental Results

Following the principles of uniform logical cryptanalysis described in Subsection 4.1, we implemented a library in the C++ programming language consisting of classes like *ModifiedUInt* and *Formula* with their member functions and overloaded operators, as well as other classes and functions. Next, we choose the following algorithms to test and compare: MD5, SHA-1, SHA-2 and CubeHash(r,b) with configuration parameters $(1,32),(2,32),(4,32),(8,32)$ and $(16,3)$. Then, their source code was slightly modified to use our classes instead of built-in types. Finally, problems of **preimage**, **second preimage** and **collision resistance**, following the theory from Subsection 4.2 were encoded as a C++ program.

Figure 2 shows comparison of functions CubeHash$(1,32)$, CubeHash$(2,32)$, MD5, SHA-1 and SHA-2. As expected, there are much more variables as well as clauses for SHA-2 than for SHA-1 or MD5 function, which indicates much bigger complexity of preimage resistance problem and therefore better security margin for SHA-2 than for the other two popular functions. This figure also shows that CubeHash$(1,32)$ has complexity close to SHA-1, whereas CubeHash$(2,32)$ is slightly better than SHA-2.

In Figure 3, we can see comparison between several CubeHash variants. Obviously, CubeHash$(16,32)$ is superior to other variants.

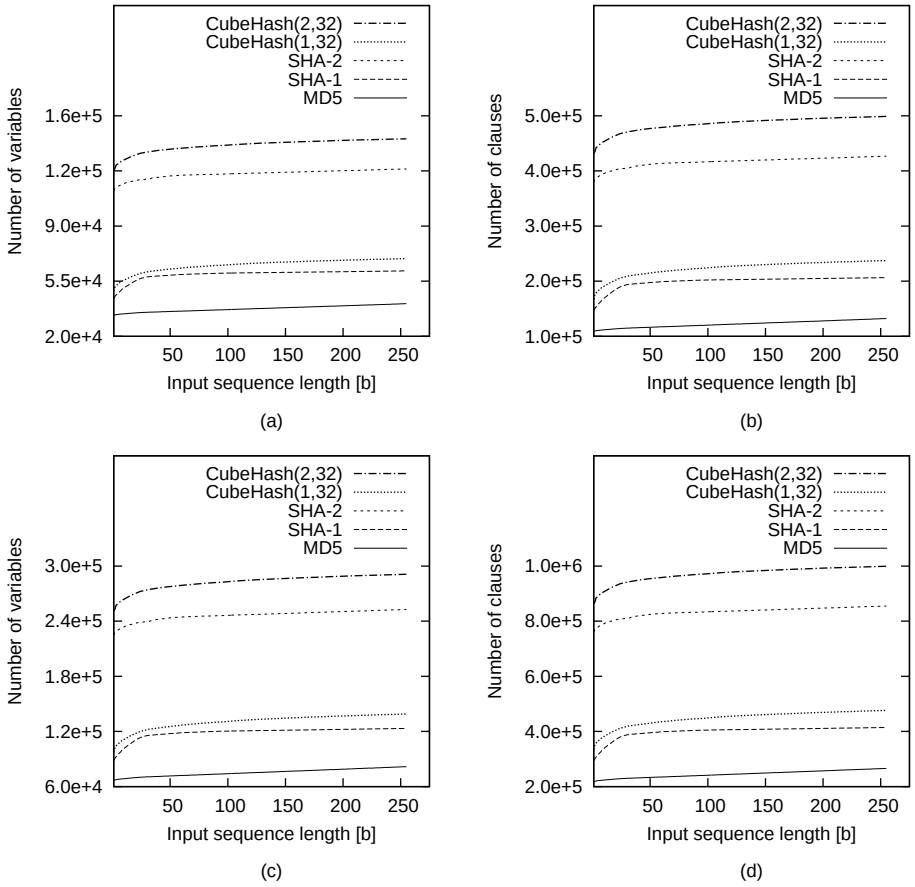

Fig. 2. Results of preimage and collision resistance problems transformed into SAT-equivalent problems for MD5, SHA-1, SHA-2, CubeHash$(1,32)$ and CubeHash$(2,32)$. (a) and (b) show results for preimage problem while (c) and (d) show results for collision problem, all against number of variables, i.e length of an input sequence given in bits.

Results for number of variables and clauses for second preimage resistance problem are very close to preimage ones, therefore they are excepted from Figures 2 and 3.

6 Conclusion

In this paper we presented results of uniform logical cryptanalysis applied on cryptographic hash function CubeHash. Compared to other logical cryptanalysis methods, this one provides easy, fast and reliable way of transforming cryptanalysis

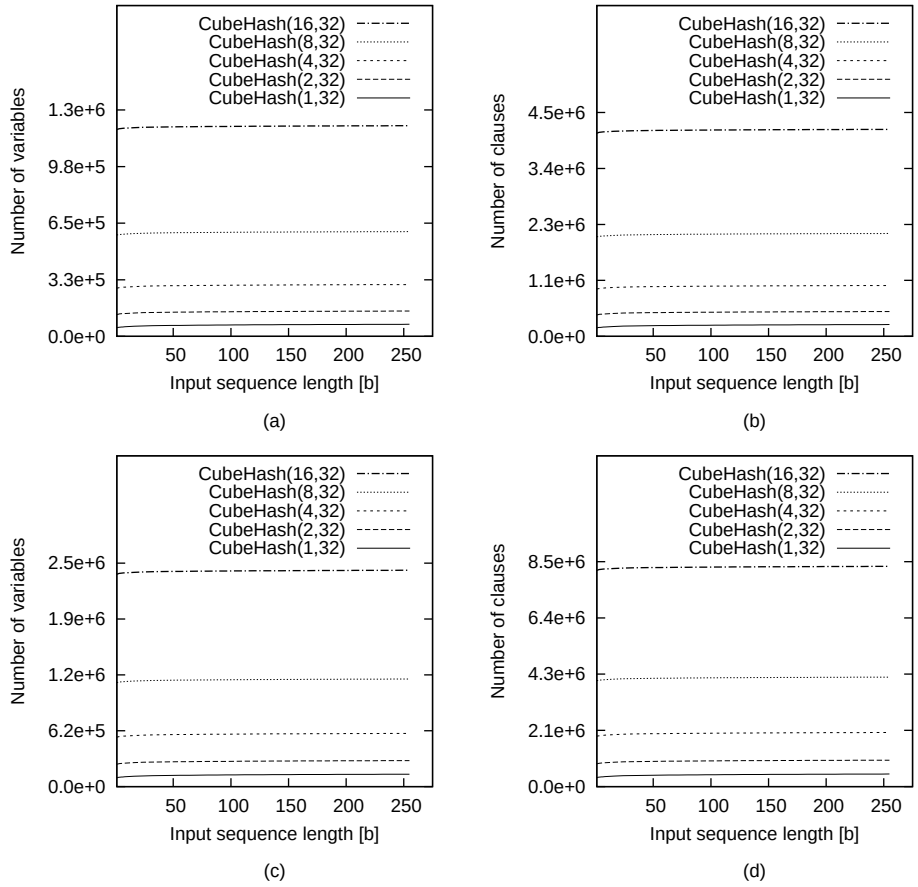

Fig. 3. Results of preimage and collision resistance problems transformed into SAT-equivalent problems for various CubeHash variants. (a) and (b) show results for preimage problem while (c) and (d) show results for collision problem, all against number of variables, i.e length of an input sequence given in bits.

problems into SAT-equivalent ones. Given analysis results show that CubeHash function configuration parameters provide a great flexibility in performance tuning. They also show that officially proposed configuration parameters for CubeHash $\mathbf{r} = 16$ and $\mathbf{b} = 32$ provide a much bigger number of variables and clauses in SAT-equivalent representation than for SHA-2 and other currently popular hash functions, which indicates a much bigger security margin, i.e cryptographic strength of this algorithm.

References

[1] S. Cook, "The complexity of theorem-proving procedures," in *Proc. 3rd Annual ACM Symposium on Theory of Computing*, 1971, pp. 151–158.

[2] F. Massacci and L. Marraro, "Logical cryptanalysis as a SAT problem," *J. Autom. Reasoning*, vol. 24, no. 1/2, pp. 165–203, 2000.

[3] United States. National Bureau of Standards, *Data Encryption Standard*, ser. Federal Information Processing Standards publication. pub-NBS:adr: U.S. National Bureau of Standards, 1977, vol. 46.

[4] D. Jovanovic and P. Janicic, "Logical analysis of hash functions," in *FroCoS*, ser. Lecture Notes in Computer Science, B. Gramlich, Ed., vol. 3717. Springer, 2005, pp. 200–215.

[5] R. Rivest, "The MD5 message-digest algorithm," Internet Engineering Task Force, Internet Request for Comment RFC 1321, Apr. 1992. [Online]. Available: ftp://nic.ddn.mil/rfc/rfc1321.txt

[6] X. Wang, D. Feng, X. Lai, and H. Yu, "Collisions for hash functions md4, md5, haval-128 and ripemd," Cryptology ePrint Archive, Report 2004/199, 2004, http://eprint.iacr.org/.

[7] FIPS, *Secure Hash Standard*, National Institute for Standards and Technology, pub-NIST:adr, Aug. 2002. [Online]. Available: http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf

[8] E. Biham, R. Chen, A. Joux, P. Carribault, C. Lemuet, and W. Jalby, "Collisions of SHA-0 and reduced SHA-1," in *EUROCRYPT*, ser. Lecture Notes in Computer Science, R. Cramer, Ed., vol. 3494. Springer, 2005, pp. 36–57.

[9] Federal Register, *Announcing Request for Candidate Algorithm Nominations for a new Cryptographic Hash Algorithm (SHA-3) Family.*, U.S. Department of Commerce, National Institute of Standards and Technology(NIST), Nov. 2007. [Online]. Available: http://csrc.nist.gov/groups/ST/hash/documents/FR%5FNotice%5FNov07.pdf

[10] D. J. Bernstein, "Cubehash specification (2.b.1)," Submission to NIST (Round 2), 2009. [Online]. Available: http://cubehash.cr.yp.to/submission2/spec.pdf

[11] Merkle, "One way hash functions and DES," in *CRYPTO: Proceedings of Crypto*, 1989.

[12] I. B. Damgård, "A design principle for hash functions," in *Advances in Cryptology—CRYPTO '89*, ser. Lecture Notes in Computer Science, G. Brassard, Ed., vol. 435. Springer-Verlag, 1990, 20–24 Aug. 1989, pp. 416–427.

[13] R. C. Merkle, "A digital signature based on a conventional encryption function," in *Advances in Cryptology – CRYPTO ' 87*, ser. Lecture Notes in Computer Science, C. Pomerance, Ed., vol. 293, International Association for Cryptologic Research. Santa Barbara, CA, USA: Springer-Verlag, Berlin Germany, 1988, pp. 369–378.

[14] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche, "On the indifferentiability of the sponge construction," in *EUROCRYPT*, ser. Lecture Notes in Computer Science, N. P. Smart, Ed., vol. 4965. Springer, 2008, pp. 181–197. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-78967-3_11

[15] B. Stroustrup, *The C++ Programming Language*, 2nd ed. Addison-Wesley, Dec. 1991.

[16] G. S. Tseitin, "On the complexity of derivation in the propositional calculus," *Zapiski nauchnykh seminarov LOMI*, vol. 8, pp. 234–259, 1968, english translation of this volume: Consultants Bureau, N.Y., 1970, pp. 115–125.