Commun. Fac. Sci. Univ. Ank. Sér. A1 Math. Stat.
Volume 66, Number 1, Pages 254–266 (2017)
DOI: 10.1501/Commua1_0000000794
ISSN 1303–5991

A SECOND PRE-IMAGE ATTACK AND A COLLISION ATTACK TO CRYPTOGRAPHIC HASH FUNCTION LUX

FATIH SULAK, ONUR KOÇAK, ELIF SAYGI, MERVE ÖĞÜNÇ, AND BEYZA BOZDEMIR

Abstract. Cryptography is a science that provides the security of information in communication. One of the most important sub-branches of cryptography is the hash functions. Hash functions are known as the digital fingerprints. Following the recent attacks on the widely used hash functions MD5 and SHA-1 and the increase in computational power, the need for a new hash function standard has arisen. For this purpose, US National Institute of Standards and Technology (NIST) had announced a competition to select a standard hash function algorithm which would eventually become the Third Secure Hash Algorithm, SHA-3. Initially 64 algorithms were submitted to NIST and 51 of them were announced as the First Round Candidates. After an analysis period, 14 of these algorithms were announced as the Second Round Candidates, and 5 algorithms were announced as Finalists. The winner of the competition, Keccak, was announced in 2012.

LUX is one of the 64 algorithms submitted to the SHA-3 competition by Nikolic et al. It is designed as a byte oriented stream cipher based hash function. For LUX-256, Schmidt-Nielsen gave a distinguisher and later Wu et al. presented collision attacks, both of which for reduced rounds of LUX. As a result of these attacks, LUX is eliminated in the first round. In this work, we first give a procedure for the second preimage attack. Then we extend this to the collision and second preimage attacks for the reduced rounds of LUX hash family. Moreover, we implement the attacks and give the specific examples by taking the padding into consideration.

1. Introduction

Cryptography is the study of the design and security of the cryptographic algorithms, providing both the information security in communication. The information security is ensured by the cryptographic algorithms using several ways such as data confidentiality, data integrity, authentication and non-repudiation. Cryptography is related to the fields of mathematics, physics, statistics, computer engineering

Received by the editors: April 15, 2016, Accepted: Oct. 10, 2016.
2010 *Mathematics Subject Classification.* 11T71, 94A60, 68P25.
Key words and phrases. Cryptography, cryptanalysis, hash function, SHA-3 competition, LUX.
This work is supported by TÜBİTAK under the project number 114F130.

and electrical engineering. To put it all in simple terms, cryptography is based on the mathematical theory in order to provide secure transformation and safekeeping of the information. With the increase in developments in technology the security of information against attacks becomes more important than ever, since accessing and using information systems have become much easier. Therefore, the need for not only the design of the cryptographic algorithms but also cryptanalysis has increased.

Cryptography can be mainly divided into two: Asymmetric and symmetric key cryptography. Asymmetric key cryptography which is generally known as public key cryptography uses different keys for encryption and decryption. Contrary to asymmetric key cryptography, symmetric cryptography is formed by three main subjects: stream ciphers, block ciphers and hash functions. In stream and block ciphers, the same key is used for both the encryption and the decryption. Hash functions produce a hash value of the message without using key. In hash functions, there is no decryption process. In addition to these subjects, message authentication codes (MACs) and authenticated encryption primitives play an important role in symmetric cryptography.

The function in which the arbitrary length input is mapped to a fixed size output is a hash function. In practice, the input size which is not arbitrary is bounded by a very large number. The output of the hash function is called the hash value, message digest, or digest value. The size of hash value changes according to the algorithm and varies generally between 128 bits and 512 bits. Cryptographic hash functions have an important role in many applications such as digital signature, message integrity checking, authentication protocols, password protection and random number generation. Therefore, if there are any flaws or weaknesses in a standard or popular hash algorithm, this affects various applications [1]. For security purposes, cryptographic hash functions must satisfy the following conditions [2]:

(1) **Pre-image resistance**: Given a hash value y, it should be hard to find any input message x satisfying $h(x) = y$ where h is an unkeyed hash function. This is related to hash function being one-way function.
(2) **Second pre-image resistance**: Given a message x_1, it should be hard to find a different message x_2 satisfying $h(x_1) = h(x_2)$ where h is an unkeyed hash function.
(3) **Collision resistance**: It should be hard to find two different messages x_1 and x_2 satisfying $h(x_1) = h(x_2)$ where h is an unkeyed hash function.

The weaknesses of commonly-used SHA-0(Secure Hash Algorithm-0), RIPEMD and MD5 were discovered in 2004 [3]. After that, these algorithms were strengthened; however, that was not enough to fulfill the need for a new hash standard. In the following years, the weaknesses of SHA-1(strengthened version of SHA-0) and MD5 were discovered [3]. As a result, the need for ensuring the security of long-term applications of hash functions arose. Therefore, NIST [4] announced a

competition, Secure Hash Algorithm-3 Competition [5], to select the new hash function algorithm. The hash function competition was arranged to design a new hash function which is called Secure Hash Algorithm-3, SHA-3, for the standardization due to the deficiencies of SHA-0 and SHA-1 Algorithms. SHA-3 Competition began on November 2, 2007 [6]. SHA-3 Competition was the response of NIST to the advances in the cryptanalysis of hash algorithms [5].

NIST called for SHA-3 Competition submission deadline and received 64 entries from cryptographers around the world by the second half of 2008. The First Round of the competition began on November 1, 2008 and 51 submissions were selected as the first round candidates at the end of 2008. The First SHA-3 Candidate Conference was announced in the beginning of 2009. 37 algorithms are eliminated due to security and performance issues and the second round of the competition began with 14 candidates in July, 2009 [5]. After 5 finalists were selected from 14 candidates, the winner of the SHA-3 Competition was announced as Keccak on October 2012 and the five year competition had ended [7].

LUX is one of the 64 algorithms submitted to the SHA-3 competition by Nikolic et al. It is designed as a byte oriented stream cipher based hash function. For LUX-256, Schmidt-Nielsen gave a distinguisher and later Wu et al. presented collision attacks, both of which were for reduced rounds of LUX. As a result of these attacks, LUX is eliminated in the first round. In this work, we give a procedure for the second pre-image attack and carry out the collision and second pre-image attacks for reduced blank round LUX hash function family. Moreover, we implement the attacks and give the specific examples by taking the padding into consideration.

In this work, we first give a procedure for the second preimage attack. Then we extend this to the collision and second preimage attack for LUX hash family. Moreover, we implement and give the specific examples of the attacks by taking the padding into consideration.

This paper is organized as follows: In Section 2 and 3, we describe LUX and the known attacks against LUX respectively. In Section 4, we present our attacks in details. In Section 5, we give the conclusion.

2. LUX Hash Function Family

LUX is a byte oriented stream based hash function design, which is submitted by Nikolic et al. [8] to the SHA-3 competition. The structure of LUX compared to the MD (Merkle-Damgard) structure is different in terms of the buffer size and the parallelism. In other words, the buffer of LUX is bigger, and it is not parallel [9].

The Design of LUX:

The LUX family consists of four hash functions with hash values 224, 256, 384 or 512 bits namely LUX-224, LUX -256, LUX -384 and LUX -512. Message is processed as the blocks of 32-bits for LUX-224 and LUX-256, 64-bits for LUX-384 and LUX-512. The internal state is 768-bits and 1536-bits for LUX-224/256 and LUX-384/512 respectively. LUX algorithm can hash messages of length up to

2^{64} bits. The internal state contains a round function consisting of Rijndael-like transformations. The internal state is updated by the state update function, ϕ, and has two buffers shown in Figure 1.

-the buffer, B, is a matrix of 4×16 for 224 and 256 bit digests and a matrix of 8×16 for 384 and 512 bit digests,

-the core, C, is a matrix of 4×8 for 224 and 256 bit digests or a matrix of 8×8 for 384 and 512 bit digests.

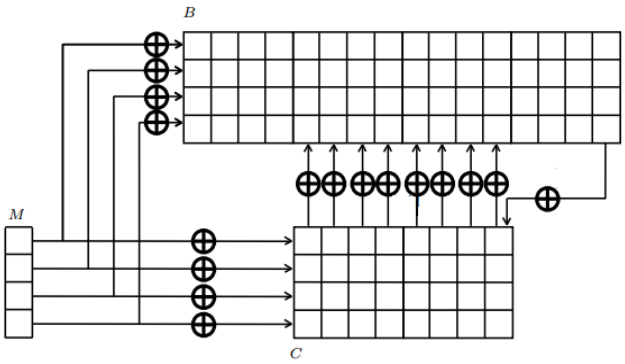

FIGURE 1. The State of LUX [8].

The hash function LUX processes the data in 8-bit words as in Rijndael [10]. In the initialization, the padding is applied to the message. If the size of the last block of the message is less than 8m, where m is 4 or 8 in bytes for 224 and 256 or 384 and 512 bit hash values respectively, then this block is padded with a "1" bit followed by "0" bits until the length of the block becomes 8m. If the length of the last message block is 8m already, then two 4-byte blocks for 224 and 256 bit digests, one 8-byte block for 384 and 512 bit digests are added then the message is XORed to the first column of the buffer and to the first column of the core. A cyclic columnwise right rotation is applied to the content of the buffer. In the core, the messages are applied Rijndael-like transformations, which are:

(1) SubBytes: It is a non-linear byte-wise function. The S-box in Rijndael is used in this transformation [10]. $S(X) = Y$ where $X = X_1 \,||\, X_2$ where X_1 is the first four bits of byte X and X_2 is the last four bits of X because of the S-box property of Rijndael.
(2) ShiftRows: The state is cyclically rotated to left with respect to the given rotation values. The shift vectors are $v = (0, 1, 3, 4)$ for LUX-224 and LUX-256, and $v = (0, 1, 2, 3, 4, 5, 6, 7)$ for LUX-384 and LUX-512 as in Rijndael [10].

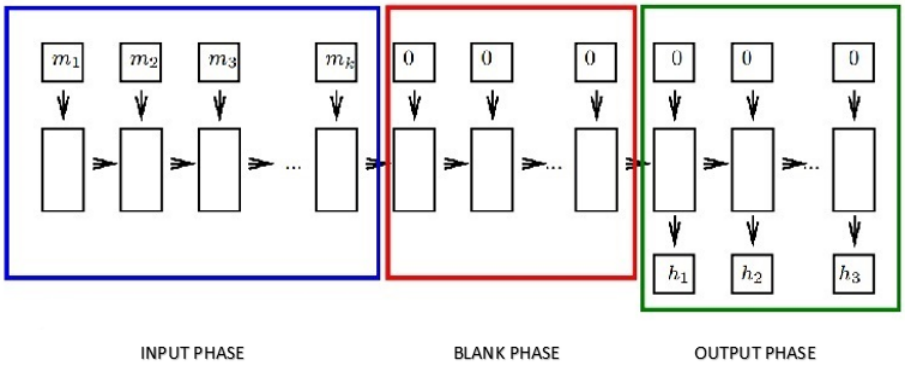

FIGURE 2. The Hashing of LUX [12]

(3) MixColumns: Each column of the core C is multiplied with a fixed matrix. This matrix is also the same as in Rijndael [10].
(4) AddConstant: This operation is similar to AddRoundkey operation of Rijndael [10]. However, in LUX, instead of round keys, a constant $0 \times 2ad01c64$ is XORed to the first column of the core C.

After the state update function, the core matrix C is XORed to the 5^{th} to 12^{nd} columns of the buffer B. Then, the last column B_{15} of the buffer is XORed to the last column C_7 of the core.

LUX produces the hash value in three phases which are given in 2.

(1) Input phase : In this phase, the message is input to the algorithm..
(2) Blank phase : In this phase, 16 rounds without any message input are applied in order to increase the diffusion of the last message blocks. The number of blank rounds is increased to 20 after the update of LUX [11].
(3) Output phase : In this phase, the hash value is produced from the state. The state update function ϕ is applied to the current state S and the message block with zeros. It contains 7, 8, 8 and 6 rounds for 224, 256, 512 and 384 bit hash values, respectively. After this process, the content of the forth column of the core C_3 is output as the hash value. Output phase is discarded after the update of LUX since a correlation between the consecutive output columns of core is found. According to the update of LUX, the hash value is produced from the seventh column of the core C_6 for LUX-224 and the sixth column of the core C_5 for LUX-384 after 20 blank rounds [11].

3. Cryptanalysis of LUX

In this section, we present the attack of Wu et al. and the distinguisher of Schmidt-Nielsen introduced during SHA-3 Competition.

3.1. **Attack of Wu et.al.:** Wu et al. give two kinds of collisions on LUX-256, namely reduced blank round collision and free-start collision. Also the authors apply the free-start pre-image attack on LUX-256 [9]. These reduced blank round collision and free-start collision attacks will be summarized below:

Reduced Blank Round Collision: Wu et al. state that if there were not enough blank rounds, they could easily construct collision messages. The authors give a message difference α in the first byte of the message as $\alpha = (\alpha_0, 0, 0, 0)$ as the first message. Then, the authors choose the second message word β which is pre-calculated from α. Using the message pair generated by α and β after three blank rounds, a collision occurs for LUX-224 and a near collision occurs for LUX-256 as shown in Table 1.

Table 1. Differential Path for Reduced Blank Round Collision [9].

round	Δm	ΔB	ΔC
0		—- —- —- —-	—- —-
1	α	-α– β— —- —-	β— —-
2	β	$\beta\alpha$– -β– —- —-	—- —-
blank-1		-$\beta\alpha$- –β- —- —-	—- —-
blank-2		–$\beta\alpha$ —β —- —-	—- —-
blank-3		—β α— β— —-	—- —-
output-1		—- $\beta\alpha$– -β– —-	—- —-
output-2		—- -$\beta\alpha$- –β- —-	—- —-
output-3		—- –$\beta\alpha$ —β —-	—- —-
output-4		—- —β α— β—	—- —-
output-5		—- —- $\beta\alpha$– -β–	—- —-
output-6		—- —- -$\beta\alpha$- –β-	—- —-
output-7		—- —- –$\beta\alpha$ —β	—- —β
output-8		β— γ-$\delta\epsilon$ —η α—	γ-$\delta\epsilon$ —ζ

3.2. **Schmidt-Nielsen distinguisher:** Schmidt-Nielsen gives a distinguisher for reduced-round LUX by using 256 different messages which differ in the first byte of the first message block. The author used the square property to give the distinguisher [13].

If the first byte in the first column of the core is changed, it will only affect the first column of the core after the first round since the first row is not shifted in the state update function Φ. The next message block can be arranged to cancel out this difference in the core so that the buffer is modified without affecting the core.

Also, because of the structure of LUX, if there exists a difference in the buffer, the core will not be affected until the difference reaches the last column of the buffer.

4. Our Attack

In this section, we present reduced round second preimage and collision the attacks on LUX. Our attacks are applicable to both versions of LUX hash family.

In our attack, we used only 2 messages, instead of 256 messages in Schmidt-Nielsen, to get a distinguisher. To be able to use these 2 messages, first, give a difference to the first byte of the first message block as shown in Figure 3. Then, the difference diffuses to the buffer and the core as in Figure 4 after the first message block is processed. Taking the XOR difference in the second block of the messages equal to the the difference in the first column of the core, the difference in the core is canceled as shown in Figure 5. There will be no difference in the core until the difference in the buffer B reaches the last column. When the difference reaches the last column of the buffer, the difference is XORed to the last column of the core as shown in Figure 6. One round later, the hash value is produced as shown in Figure 7. If one continues the process, difference will spread to the all columns of the core as in Figure 8. Hence, one cannot control the differences in the core any more. Therefore, the collisions should be found before the differences in the buffer affects the core.

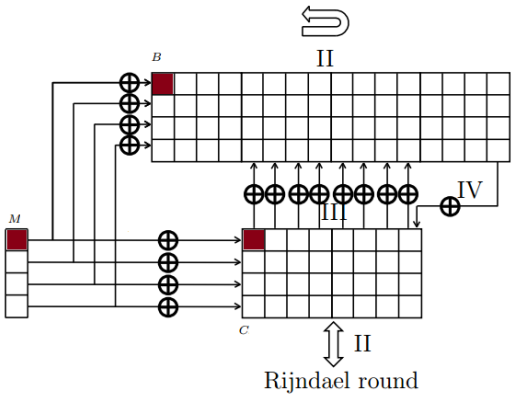

Figure 3. Input difference.

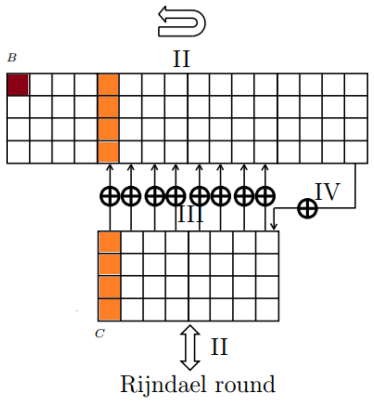

FIGURE 4. After 1 blank round.

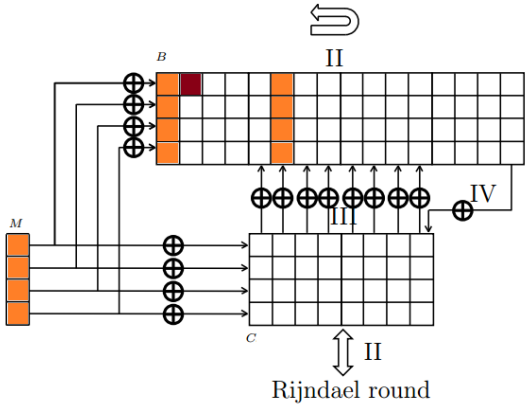

FIGURE 5. Adding new message block.

The attack presented by of Wu et al. does not include padding. In this work, we use the same idea in [9] with taking the padding into consideration. Considering the messages having last bits as "100..0" and find a collision, one can remove these bits and get the same message after padding.

For LUX-256(224) we obtained the following distinguisher results with the reference implementation of LUX, which results in a 32-bit near collision. We need two function calls for the 32-bit near collision, where the generic attack needs 2^16 function calls.

Message 1: 0×525252d505030306
Message 2: 0×5252520903010102

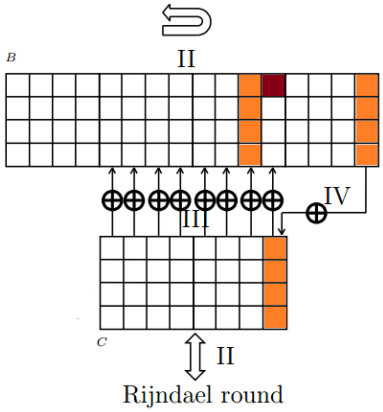

FIGURE 6. After 10 blank rounds.

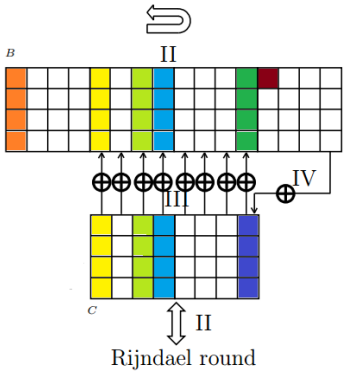

FIGURE 7. After 11 blank rounds.

with message length=63 and with Blank Round=7
$LUX(Message1) \oplus LUX(Message2) =$
0×00000000DAB76D6D72EDF107B6F7D3179E99CD7D5D3847AD4230CE4A

Notice that the reference implementation of LUX inputs the message bytes in reverse order. Therefore, for instance, the first message is 0×06030305d5525252 in fact.

Also these messages yield a collision for LUX-224 with 1 blank round shown in Table 2.

For LUX-512(384) we can improve the results since LUX-512(384) takes 8-byte message parts as input and also the message is only one 8-byte block. So we get;

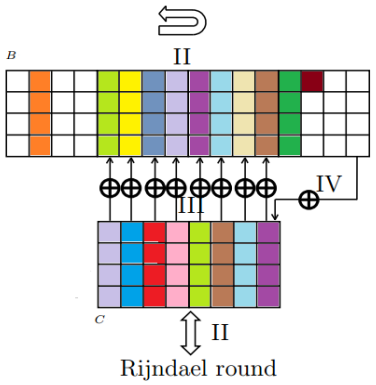

FIGURE 8. After 12 blank rounds.

TABLE 2. DIFFERENTIAL TRIAL FOR LUX-256

Rounds	Δ m	Buffer Difference	Core Difference
0	-	0000 0000 0000 0000	0000 0000
1	a	0a00 b000 0000 0000	b000 0000
2	b	0ba0 0b00 0000 0000	0000 0000
M.Length 1	-	00ba 00b0 0000 0000	0000 0000
M.Length 2	-	000b a00b 0000 0000	0000 0000
Blank-1	0	0000 ba00 b000 0000	0000 0000
Output-1	0	0000 0ba0 0b00 0000	0000 0000
Output-2	0	0000 00ba 00b0 0000	0000 0000
Output-3	0	0000 000b a00b 0000	0000 0000
Output-4	0	0000 0000 ba00 b000	0000 0000
Output-5	0	0000 0000 0ba0 0b00	0000 0000
Output-6	0	0000 0000 00ba 00b0	0000 0000
Output-7	0	0000 0000 000b a00b	0000 000b
Output-8	0	b000 c0de 000f ba00	c0de 000f

Message 3: 0×525252525252522d0000000000000000
Message 4: 0×52525252525252d775d6d6b79fc2ead6
With message length=127 and with Blank Rounds=8
$LUX(Message3) \oplus LUX(Message4) =$
0×000000000000000041ad7e493f4949920bf243bbab566cd82659b0891bb318b2c33ec
da76a66d3ea0f739d9411e239290896f15e9a658de5d17fd5315447c7ac

Also with these messages we get collisions for LUX-512 with 1 blank round and for LUX-384 with 3 blank rounds shown in Table 3.

TABLE 3. DIFFERENTIAL TRIAL FOR LUX-512

Rounds	Δ m	Buffer Difference	Core Difference
0	-	0000 0000 0000 0000	0000 0000
1	a	0a00 b000 0000 0000	b000 0000
2	b	0ba0 0b00 0000 0000	0000 0000
M.Length	-	00ba 00b0 0000 0000	0000 0000
Blank-1	0	000b a00b 0000 0000	0000 0000
Blank-2	0	0000 ba00 b000 0000	0000 0000
Output-1	0	0000 0ba0 0b00 0000	0000 0000
Output-2	0	0000 00ba 00b0 0000	0000 0000
Output-3	0	0000 000b a00b 0000	0000 0000
Output-4	0	0000 0000 ba00 b000	0000 0000
Output-5	0	0000 0000 0ba0 0b00	0000 0000
Output-6	0	0000 0000 00ba 00b0	0000 0000
Output-7	0	0000 0000 000b a00b	0000 000b
Output-8	0	b000 cdef ghij ba00	cdef ghij

Moreover, we perform a second preimage attack on LUX. For this attack, one has to choose a message and should find another message which produces the same hash value. For simplicity, we choose the first message as M1: 0×0000000000000000, but the attack is applicable for any message. We obtain the first 32-bit of the second message M2 as M2=M1$\oplus$ α where $\alpha = (0 \times 01, 0 \times 00, 0 \times 00, 0 \times 00)$. After one round of LUX, we take the values of the first column of the core of both messages. While the value of the first column of core of the first message is 0×077fb349, the value of the first column of core of the second message is 0×2660ac77. By XORing these values, we obtain the second 32-bit of the second message M2. Hence, the second message M2 is 0×00000001211f1f3e. The hash values of these two messages collide for LUX-224 with 1 blank round. Hence, a second preimage is found for the message M1 for LUX-224 with one blank round.

Message 1: 0×0000000000000000
Message 2: 0×00000001211f1f3e
with message length=63 and with Blank Round=1
$LUX(Message1) \oplus LUX(Message2) =$
0×00000000000000000000000000000000000000000000000000000000000000000000

5. CONCLUSION

Attacks on hash functions are generally theoretic and the complexities of attacks are high. However, in this paper we carry out practical attacks for LUX hash function family. We give near collision and second pre-image attacks considering the padding for all hash functions of the LUX family. Moreover, we implement these attacks and give specific examples for the attacks. We give an example of the second

pre-image attack for LUX-224 with 1 blank round and examples of collisions and near-collisions for LUX 224, 256, 384 and 512 by using our implementation based on the reference implementation of LUX. Our contributions are summarized at Table 4.

TABLE 4. Number of rounds for collision, near collision and second pre-image for LUX

	LUX-224	LUX-256	LUX-384	LUX-512
Collision	1 round	-	Up to 3 rounds	1 round
Near Collision	Up to 7 rounds	Up to 7 rounds	Up to 8 rounds	Up to 8 rounds
Second pre-image	1 round	-	-	-

On the contrary to the previous works on LUX, we perform the practical attacks for LUX and implement the attacks and hence we give the examples of collisions, near-collisions and second pre-image of LUX.

References

[1] Koçak O. Design and analysis of hash functions. MSc, Middle East Technical University, Ankara, Turkey, 2009.

[2] A. Menezes, P. van Oorschot, and S. Vanstone, Handbook of Applied Cryptography, CRC Press, 1996.

[3] Wang X, Feng D, Lai X and Yu H. Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD. Available at http://eprint.iacr.org/2004/199.pdf

[4] US National Institute of Standards and Technology, http://www.nist.gov/

[5] Secure Hash Algorithm-3 Competition, http://csrc.nist.gov/groups/ST/hash/sha-3/index.html

[6] Office of the Federal Register, National Archives and Records Administration, Federal Register / Vol. 72, No. 212 (PDF). Federal Register. Government Printing Office. November 2, 2007. Retrieved 2008-11-06.

[7] Secure Hash Algorithm-3 Competition for 3^{rd} Round, http://csrc.nist.gov/groups/ST/hash/sha-3/Round3/index.html.

[8] Ivica Nikolić, Alex Biryukov, and Dmitry Khovratovich. Hash function family LUX, 2008, submitted to the SHA-3 competition.

[9] Shuang Wu, Dengguo Feng, Wenling Wu. Cryptanalysis of the Hash Function LUX-256. Available at http://ehash.iaik.tugraz.at/uploads/3/36$Analysis_LUX_1.pdf$, 2008.

[10] John Daemen and Vincent Rijmen. The Block Cipher Rijndael, 1998.

[11] Ivica Nikolić, Alex Biryukov, and Dmitry Khovratovich. Specification Update of the Hash family LUX, 2009.

[12] Ivica Nikolić, Alex Biryukov, and Dmitry Khovratovich. Hash function family LUX slide, 2009, submitted to present LUX for First SHA-3 Candidate Conference.

[13] Peter Schmidt-Nielsen. A Distinguisher for Reduced-round LUX. Available at http://ehash.iaik.tugraz.at/uploads/3/3b/LUXATTACKNext.pdf, 2008.

Current address, Fatih Sulak: Atılım University, Dept. of Mathematics, Ankara, TURKEY
E-mail address: fatih.sulak@atilim.edu.tr
Current address, Onur Koçak: TÜBİTAK BİLGEM 41470 Gebze, Kocaeli, Turkey
E-mail address: onur.kocak@tubitak.gov.tr
Current address, Elif Saygı: Department of Basic Education Division, Elementary Mathematical Education, Hacettepe University, Beytepe, Ankara, Turkey
E-mail address: esaygi@hacettepe.edu.tr
Current address, Merve Öğünç: Institute of Applied Mathematics, Cryptography Department, METU, Ankara, Turkey
E-mail address: mrveogunc@gmail.com,
Current address, Beyza Bozdemir: Institute of Applied Mathematics, Cryptography Department, METU, Ankara, Turkey
E-mail address: beyzabozdemir06@gmail.com