Hash Function Designs Based on Stream Ciphers

Meltem Sönmez Turan, *Institute of Applied Mathematics, METU,*
Özgür Özuğur, *Institute of Applied Mathematics, METU,*
and Onur Kurt *Institute of Applied Mathematics, METU*

***Abstract*—Hash functions are fundamental components of many cryptographic applications. Due to the recent attacks on classical designs, alternative design approaches are proposed. In this paper, we give a survey for hash function constructions based stream ciphers mainly concentrating on Panama, RadioGatún and RC4-hash. Also, we propose an example hash function Dragon-Hash based on the stream cipher Dragon.**

***Index Terms*—Hash functions, Stream ciphers, Panama, RadioGatún, RC4-Hash**

I. Introduction

Hash functions are fundamental components of many cryptographic applications such as digital signatures, random number generation, integrity protection, e-cash etc. Employing hash functions for these applications both increase the security and improve the efficiency of these systems. However, availability of any weaknesses in hash function designs is a very serious threat against the security of these applications.

MD5 [1], SHA-1 [2] and RIPEMD [3] are widely used hash functions especially in SSL, PGP, S/MINE, SSH and SFTP applications. The design of these hash functions are based on the hash function MD4 [4] as they iteratively use a compression function that inputs state variable and a fixed length block, and outputs another fixed length block. The strength of these hash functions is based on the collision-resistance property of the compression function.

Recently, many attacks against hash functions having similar construction to MD4 are proposed [5], [6], [7], [8]. Nowadays, these constructions are not considered to provide sufficient level of security against collision resistance. These recent studies motivated National Institute of Standards and Technology (NIST) to announce a public competition to select a new cryptographic hash function to be used as the new standard. It is commonly believed that many alternative design approaches are going to be proposed for the competition. Designing hash functions based on stream ciphers is a good approach due to the efficiency and the speed of these ciphers.

Panama [9] is the first hash function based on a stream cipher. However, an attack against Panama is proposed, thereafter RadioGatún, an improved version of Panama, is proposed and claimed to offer better security than MD4 primitives. In this approach, iterative use of a simple round function and inclusion of input blocks are proposed. After inclusion of all input blocks, the state is updated a number of times without producing any output. Another hash function RC4-Hash [10] is based on the very popular stream cipher RC4.

In this paper, we give a survey for hash function constructions based stream ciphers mainly concentrating on Panama, RadioGatún and RC4-hash. Also, we propose an example hash function Dragon-Hash based on the stream cipher Dragon [11]. It should be noted that this new design is for educational purposes and its resistance to available attacks is not completely analyzed.

The organization of the paper is as follows. In the next section, we present some background information on hash functions and stream ciphers. Then, we give descriptions of the stream cipher based hash functions Panama, RadioGatún and RC4-Hash. In section 6, we propose an example hash function Dragon-Hash based on the stream cipher Dragon. Finally, in the last chapter we give some concluding remarks.

II. Preliminaries

A. Hash Functions

Hash functions take arbitrary length input and produce a fixed length output which is commonly called *fingerprint* or *message digest* of the input. Some desired properties of hash functions should satisfy are given below.

- Algorithm of a hash function should be publicly known. There may not be any secret parameters.
- For a given value x and a hash function h, it should be 'easy' to compute $H(x)$.

A cryptographic hash function has to be resistant against three main attacks;

- Preimage resistance: For a given value $H(x)$, it should be 'hard' to compute x.
- Second preimage resistance: Given x and $H(x)$, it should be 'hard' to find x' such that $x \neq x'$ and $H(x') = H(x)$.
- Collision resistance: For any x, it should be 'hard' to find x' where $x \neq x'$ and $H(x) = H(x')$.

A common way to construct a hash function is to use iterations. Firstly, a padding rule is applied to the message x and the sequence $x_1x_2x_3$ is obtained. Adding the sequence $100\ldots0$ at the end of the message is very commonly used padding rule. At each iteration, a compression function $h(x)$ that inputs x_i and $h(x_{i-1})$ is used. Output of i^{th} iteration h_i is called the *chaining variable* or the *intermediate variable*. As the output of the last iteration, the hash of the message is obtained. The first chaining variable is called the *Initial Value* (IV). The iterative hash functions can be summarized as follows:

$$\begin{aligned} h_0 &= IV, \\ h_i &= h(x_i, h_{i-1}),\ i = 1, 2, \ldots, t \\ H(x) &= h_t \end{aligned}$$

For such designs the security of the hash function relies on the security of the compression function h.

B. Stream Ciphers

Synchronous stream ciphers are an important part of symmetric cryptosystems and they are suitable for applications where high speed and low delay are required. As examples, the stream cipher family A5 is used in the GSM standard and the cipher E0 is used to supply privacy in Bluetooth applications. The basic design philosophy of stream ciphers is inspired by the perfectly secure One Time Pad cipher that encrypts the plaintext with a random key using the XOR operation. To simulate One Time Pad, stream ciphers generate long pseudo-random keystream from a short random key.

In the generic structure of a *synchronous stream cipher*, firstly the secret key K and publicly known IV is loaded to state of the cipher. Then, it is clocked a number of times without producing any output. After this initialization phase, desired length of keystream is produced using keystream generation phase.

III. Panama

Panama [9] is designed to be a cryptographic module that can be used as both a stream cipher and a hash function. It is based on a finite state machine of size 544 bits denoted by $(a_0, a_1, a_2, \ldots, a_{16})$ where each a_i is a 32 bit word. The state is also denoted by (a_0, a^s, a^t) where a^s is the words from a_1 to a_8 and a^t is the words from a_9 to a_{16}. The buffer of Panama is a LFSR that includes 32 stages each containing 256 bits. b^j is used to denote 8-word stage and b_i^j for its words ($0 \leq j \leq 31$ and $0 \leq i \leq 7$), namely b_2^3 is third word of the fourth stage of the buffer.

Panama has three different modules; *reset*, *push*, *pull*. In the reset mode, the state and buffer is initialized as zero. In the push mode, 8-word input is applied to the stages without producing any output. In the final mode, without any input, the output 8-bit word is obtained. Panama uses 4 transformations with different purposes; γ, π, θ and σ, for nonlinearity, permutation, diffusion and injection of the buffer and input bits, respectively. The composition of transformations $\theta \circ \pi \circ \gamma$ is denoted as ρ.

- *The Transformation* θ is an invertible linear transformation which is defined by:

$$y = \theta(a) \Leftrightarrow y_i = a_i \oplus a_{i+1} \oplus a_{i+4} \text{ for } i = 0, \ldots 16$$

- *The Transformation* π combines cyclic word shift and a permutation of the word positions. $[j]$ denotes the bit position in a word $0 \leq j < 31$, $7i$ is in the modulo 17, $S(i) = i(i+1)/2$

$$y = \pi(a) \quad \Leftrightarrow \quad y_i[j] = a_{7i}[j + S(i)]$$

  where $j + S(i)$ is in the modulo 32.
- *The Transformation* γ is an invertible nonlinear transformation defined by:

$$y = \gamma(a) \Leftrightarrow y_i = a_i \oplus (a_{i+1} \vee (a_{i+2} \oplus 1)) \text{for } i = 0, \ldots 16$$

A. The Panama Hash Function

The Panama hash function takes a message M of arbitrary length and generates a hash of 256 bits. It is designed for 32-bit architectures, it is as fast as MD4 which is the fastest hash function in the MD family. However, the performance of Panama hash function is not very efficient for the short messages due to large number of iterations in initialization.

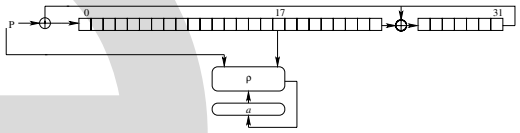
Time step t	Mode	Input	Output
0	$Reset^1$	-	-
$1, \ldots, n$	$Push$	m^t	-
$n+1, \ldots, n+32$	$Pull$	-	-
$n+33$	$Pull$	-	h

TABLE I: Iteration steps of the Panama[9]

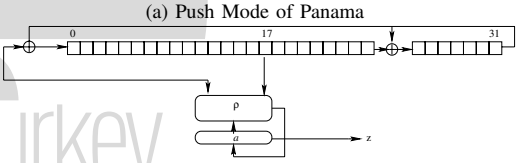
There are two main steps in the Panama hash function. In the padding step, the message M is converted into a string $M^{'}$ whose length is a multiple of 256 by appending a single 1 followed by a number r of 0-bits with $0 \leq r < 256$. In the iteration step, $M^{'}$ is divided into 256-bits length message blocks and is denoted by $M^{'} = M_1, M_2, M_3 \ldots M_n$. Firstly, all bits in the state and the buffer are 0-bit, for each message block M_k $(0 \leq k \leq n)$ the state is updated by applying the 3 specific transformations ρ, then transformation σ is applied. The message block m^k and b^{16} are XORed a^s and a^t respectively and a_0 is flipped.

$$\sigma(a_0, a^s, a^t) = (a_0 \oplus 1, a^s \oplus M_k, a^t \oplus b^{16})$$

The figures of push and pull mode of Panama are given in Figure 1a and 1b.



(a) Push Mode of Panama



(b) Pull Mode of Panama

Then, the buffer takes block M_k as an input and the LFSR is stepped once. The buffer uses an updating function $\alpha(b) = \beta$

$$\begin{aligned} \beta^j &= b^{j-1} \; if \; j \notin \{0, 25\} \\ \beta^0 &= b^{31} \oplus M_k \\ \beta_i^{25} &= b_i^{25} \oplus b_{i+2 \; mod \; 8}^{31} \text{ for } 0 \leq i \leq 7 \end{aligned}$$

After all input blocks have been used, 33 pull iteration are performed. The outputs of the first 32 iterations of the pull mode are discarded to diffuse the buffer and state. The output of the last iteration which is the state part a^t is taken for the hash result.

IV. RadioGatún

The security of Panama is analyzed in [12] and an attack with complexity 2^{82} and negligible amount of memory to find collisions faster than birthday attack is proposed. Considering the weakness of Panama, its design is modified and

RadioGatún [13], a belt-and-mill hash function, is proposed. The main differences between Panama and RadioGatún are; (i) addition of feedforward mechanism from mill to belt, (ii) smaller input block size and belt width and (iii) enlargement of the mill from 17 to 19 words. Additionally, in the RadioGatún hash function an alternating-input iterative mangling function (IMF) with the belt-and-mill structure is used which may be a valuable alternative for MD4-like hash functions.

IMF takes a variable-length input and returns an infinite output stream. It can be simply derived to a hash function by truncating the output to the first n bits. The variable-length input for IMF is constructed by an alternating-input mechanism. The construction algorithm is specified in Algorithm IV.1.

The belt-and-mill structure is the generalization of the round function of Panama. It consists of four operations in parallel; (i) *Mill function*, an invertible non-linear function applied to the mill, *Belt function*, an invertible simple linear function applied to the belt, *Milt feedforward*, some bits of the mill are fed to the belt in a linear way and *Bell feedforward*, some bits of the belt are fed to the mill in a linear way. The algorithm of belt-and-mill function is specified in Algorithm IV.2.

Algorithm IV.1: INPUT CONSTRUCTION($p_0, \ldots, p_{n_p-1}$)

$S \leftarrow 0$ /* State initialization */
for $i \leftarrow 0$ **to** n_{p-1}
$\{T = S \oplus F_i(p_i) S = R(T)$
for $i \leftarrow 0$ **to** n_{b-1}
$\{S = R(S)$
for $i \leftarrow 0$ **to** n_{z-1}
$\begin{cases} S = R(S) \\ z_i = F_0(S) \text{ /* } F_0\text{: output mapping */} \end{cases}$
return $(z_0, \ldots, z_{n_z-1})$

Algorithm IV.2: A BELT-MILL ROUND FUNCTION($-$)

$(A, B) = R(a, b)$ where
$A = MILL(a) \oplus BELL(b)$
$B = BELT(b) \oplus MILT(a)$

The generic structure of the RadioGatún is as follows:

$$z = \text{RadioGatún}[l_w](x)$$

where x is the input block, l_w is the word length which can take any value between 1 and 64 and z is the infinite output stream. RadioGatún function is an alternating-input IMF with the belt-and-mill structure. The mill a has 19 words $a[i]$, the belt b of 13 stages $b[i]$ of 3 words $b[i, j]$ each. An input block p consists of 3 words $p[i]$, an output block z has 2 words $z[i]$. The round function, the mill function and input/output mappings are given by Algorithm IV.3 , Algorithm IV.5, Algorithm IV.4 and Algorithm IV.6, respectively.

The function works as follows:

- Padding to x by appending a single bit 1 and 0's until the length of the result is a multiple of the input block length and decomposing it into input blocks p_0 to $p_{n_{p-1}}$.
- Then execute Algorithm IV.1 with Algorithm IV.3 until determined output bits are generated. The number of blank rounds is n_b=16.

The figure of RadioGatún is given in Figure 1.

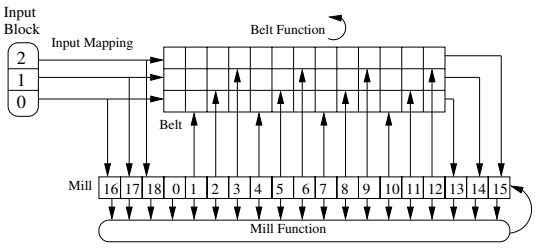


Fig. 1: The structure of RadioGatún

Algorithm IV.3: THE ROUND FUNCTION R($-$)

$(A, B) = R(a, b)$
$B[i] = b[i + 1 \ mod \ 13]$ for $i = 0, \ldots, 12$
$B[i + 1, i \ mod 3] = B[i + 1, i \ mod \ 3] \oplus a[i + 1]$
$A = MILL(a)$ /* Mill function */
$A[i + 13] = A[i + 13] \oplus b[12, i]$ for $i = 1, 2$

Algorithm IV.4: THE INPUT MAPPING($p[i]$)

$(a, b) \leftarrow 0$
for $i \leftarrow 0$ **to** 2
$\begin{cases} b[0, i] = p[i] \\ a[i + 16] = p[i] \end{cases}$
return (a, b)

Algorithm IV.5: THE MILL FUNCTION(a)

$A = MILL(a)$
/* γ: non-linearity */
$A[i] = a[i] \oplus \overline{a[i + 1]} a[i + 2]$ for $i = 0, \ldots, 18$
/* π: intra-word and inter-word dispersion */
$a[i] = A[7i] \ggg i(i + 1)/2$for $i = 0, \ldots, 18$
/* θ: diffusion */
$A[i] = a[i] \oplus a[i + 1] \oplus a[i + 4]$ for $i = 0, \ldots, 18$
$A[0] = A[0] \oplus 1$ /* ι: asymmetry */

Algorithm IV.6: THE OUTPUT MAPPING($-$)

$z[0] = a[1]$
$z[1] = a[2]$
return (z)

V. RC4

RC4 is a synchronous stream cipher designed by Ron Rivest in 1987 and the details of the algorithm were kept as a trade secret, however in 1994 the details leaked out using reverse engineering. It is widely used in SSL and WEP applications.

The state of the cipher is defined to be a 2^8-word table that consists of a permutation of integers between 0 and 255. Firstly, using the secret key, the permutation table is updated, then they keystream is generated using swap operations.

Various papers on the security analysis of the stream cipher RC4 were published. Most of these papers concentrate on the weaknesses of the keystream generation function [14], [15], [16].

A. RC4 Hash

In [10], a new hash function family RCH_l, $16 \leq l \leq 64$ based on the stream cipher RC4 is proposed. Firstly, the padding rule

$$pad(M) = bin_8(l)||M||1||0^k||bin_{64}(|M|)$$

is applied to the message M where $bin_{64}(|M|)$ is the binary representation of number of bits of M, and k is the least non-negative integer such that $8+|M|+1+k+64 \equiv 0 \bmod 512$. Then, $pad(M) = M_1||\ldots||M_t$ such that each M_i consists of 512 bits. The maximum possible message length is $2^{64}-1$.

As the second step, the iterations

$$(S_0, j_0) \overset{M_1}{\to} (S_1, j_1) \overset{M_2}{\to} \ldots (S_{t-1}, j_{t-1}) \overset{M_t}{\to} (S_t, j_t) = C^+(M).$$

are followed where $(S, j) \overset{X}{\to} (S^*, j^*)$ is defined as $C((S, j), X) = (S^*, j^*)$. The initial (S_0, j_0) is taken as $(S^{IV}, 0)$. The pseudo code of the compression function C is given in Algoritm V.1 and the details of the function r is available in [10].

Finally, after obtaining (S_t, j_t), S_{t+1} is computed as $S_0 \circ S_t$ and j_{t+1} as j_t. The hash value of the message M, $HBG_l(OWT(S_{t+1}, j_{t+1}))$. The pseudo-codes of the functions HBG and OWT functions are given in Algorithm V.3 and V.2, respectively.

Algorithm V.1: C$((S, j), X)$

for $i \leftarrow 0$ **to** 255
$\begin{cases} j = j + S[i] + X[r(i)]; \\ \text{swap}(S[i], S[j]); \end{cases}$
return (S, j)

Algorithm V.2: OWT(S, j)

$Temp1 = S$;
for $i \leftarrow 0$ **to** 511
$\begin{cases} j = j + S[i]; \\ \text{swap}(S[i], S[j]); \end{cases}$
$Temp2 = S$;
$S = Temp1 \circ Temp2 \circ Temp1$;
return (S, j)

Algorithm V.3: $HBG_l(S, j)$

for $i \leftarrow 0$ **to** l
$\begin{cases} j = j + S[i]; \\ \text{swap}(S[i], S[j]); \\ Out = S[S[i] + S[j]]; \end{cases}$

VI. Dragon-Hash, A New Hash Function Based on Dragon

In this section, we propose a new example hash function design Dragon-Hash based on the stream cipher Dragon [11]. The purpose of the design is to demonstrate that given a stream cipher, it is possible to design a hash function. Very limited research on the security of Dragon-Hash is done, it should be noted this design is not suitable for any use other than education purposes.

Dragon is one of the candidates eSTREAM project that aims to identify new stream ciphers that might become suitable for widespread adoption. Dragon consists of a word based nonlinear feedback shift register of size 1024 and a nonlinear filter with memory. The NFSR is updated using the function F that consists of two 8×32-bit s-boxes, and inputs 192 bits and outputs 192 bit. The 64 bits of the F's output is given as output after each iteration. For the details of F see [11].

A. Dragon-hash

Given any message of length less than 2^{64}, Dragon-Hash produces the hash of size 256 bits. There is no constraint on the value of IV, and can be chosen randomly. Firstly, a simple padding rule is applied to the message M. The string $100\ldots0$ and binary representation of the message length are appended to M and the number of 0's in the string $100\ldots0$ is chosen so that the message length is a multiple of 256. Secondly, the message is divided into blocks of size 256, in the form $M_1, M_2, \ldots, M_t$. Then, each message block is loaded to the function as given in Algorithm VI.1. Dragon-Hash has a Merkle-Damgård structure given in the following equations;

$$\begin{aligned} h_0 &= IV, \\ h_i &= DragonHash(M_i, h_{i-1}),\ i = 1, 2, \ldots, t \\ H(x) &= h_t. \end{aligned}$$

The security of Dragon-Hash depends on the security of stream cipher Dragon and there exists no attack against Dragon better than exhaustive key search. For the stream cipher Dragon, it is not possible to obtain the secret key using the first 256 keystream bits, this is equivalent to the pre-image resistance of Dragon-Hash. However, it should be noted that there are slight changes in the keystream generation of the stream cipher Dragon and Dragon-Hash. Obtaining a collision using birthday attack requires about 2^{128} complexity, since the hash size is 256 bits.

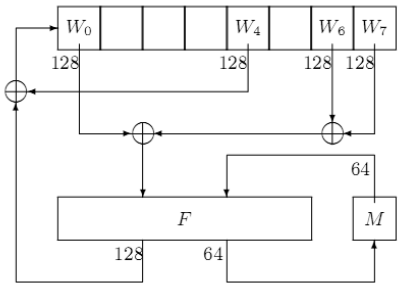


Fig. 2: The structure of Dragon[11]

Algorithm VI.1: DRAGON HASH(M_i, h_{i-1})

$W_0||W_2||\dots||W_7 = M_i||(M_i \oplus IV)||\overline{(M_i \oplus IV)}||IV$
$Memory = 0x0000447261676F6E$
for $i \leftarrow 1$ **to** 16
$\begin{cases} a||b||c||d = (W_0 \oplus W_6 \oplus W_7) \\ e||f = Memory \\ \{a', b', c', d', e', f'\} = F(a, b, c, d, e, f) \\ W_0 = (a'||b'||c'||d') \oplus W_4 \\ W_j = W_{j-1} \text{ for } j = 7, \dots 1; \\ Memory = e'||f' \end{cases}$
for $i \leftarrow 1$ **to** 4
$\begin{cases} a||b||c||d = (W_0 \oplus W_6 \oplus W_7) \\ e||f = Memory \\ \{a', b', c', d', e', f'\} = F(a, b, c, d, e, f) \\ W_0 = (a'||b'||c'||d') \oplus W_4 \\ W_j = W_{j-1} \text{ for } j = 7, \dots 1; \\ Memory = e'||f' \\ Output_i = a'||e' \end{cases}$
$Hash = (Output_1||Output_2||Output_3||Output_4) \oplus h_{i-1}$
return $(Hash)$

Hash Function	Hash Size	State Size	Block size	Max. Message Length	Word Size	Collision
PANAMA	256	8736	256	-	32	With flaws
RadioGatún	Arbitrary	58 words	3 words	-	1-64	No
RC4-Hash$_l$	128-512	2048	512	$2^{64}-1$	8	-
Dragon-Hash	256	1088	256	$2^{64}-1$	128	-
MD2	128	384	128	-	8	Almost
MD4	128	128	512	$2^{64}-1$	32	Yes
MD5	128	128	512	$2^{64}-1$	32	Yes
RIPEMD	128	128	512	$2^{64}-1$	32	Yes
RIPEMD-128/256	128/256	128/256	512	$2^{64}-1$	32	No
SHA-0	160	160	512	$2^{64}-1$	32	Yes
SHA-1	160	160	512	$2^{64}-1$	32	With flaws
SHA-256/224	256/224	256	512	$2^{64}-1$	32	No
SHA-512/384	512/384	512	1024	$2^{128}-1$	64	No
Tiger-2 [17]	192/160/128	192	512	$2^{64}-1$	64	No

TABLE II: Comparison of some of the hash functions

VII. CONCLUSION

Due to the many attacks against hash functions having the construction similar to MD4, different hash function designs are needed. Using stream ciphers as a building block for hash function designs seem to be a very appropriate alternative due to the speed and efficiency of stream ciphers. We believe that new evaluation criteria and attack methods for hash functions are going to be proposed as a result of these designs based on stream ciphers.

In Table II, we give a comparison of some of the most commonly used hash functions.

REFERENCES

[1] Ronald L. Rivest. The MD5 message-digest algorithm, 1992. URL: http://theory.lcs.mit.edu/ rivest/rfc1321.txt. Note: Request For Comments 1321.
[2] *Secure hash standard*. National Institute of Standards and Technology, Washington, 2002. URL: http://csrc.nist.gov/publications/fips/. Note: Federal Information Processing Standard 180-2.
[3] Hans Dobbertin, Antoon Bosselaers, and Bart Preneel. Ripemd-160: A strengthened version of ripemd. In Gollmann [18], pages 71–82.
[4] Ronald L. Rivest. The MD4 message-digest algorithm, 1991.
[5] Xiaoyun Wang, Dengguo Feng, Xuejia Lai, and Hongbo Yu. Collisions for hash functions MD4, MD5, HAVAL–128 and RIPEMD, 2004. URL: http://eprint.iacr.org/2004/199/.
[6] Florent Chabaud and Antoine Joux. Differential collisions in sha-0. In *CRYPTO '98: Proceedings of the 18th Annual International Cryptology Conference on Advances in Cryptology*, pages 56–71, London, UK, 1998. Springer-Verlag.
[7] Xiaoyun Wang and Hongbo Yu. How to break md5 and other hash functions. In Cramer [19], pages 19–35.
[8] Eli Biham, Rafi Chen, Antoine Joux, Patrick Carribault, Christophe Lemuet, and William Jalby. Collisions of sha-0 and reduced sha-1. In Cramer [19], pages 36–57.
[9] Joan Daemen and Craig S. K. Clapp. Fast hashing and stream encryption with panama. In Serge Vaudenay, editor, *Fast Software Encryption*, volume 1372 of *Lecture Notes in Computer Science*, pages 60–74. Springer, 1998.
[10] Donghoon Chang, Kishan Chand Gupta, and Mridul Nandi. Rc4-hash: A new hash function based on rc4. In Rana Barua and Tanja Lange, editors, *INDOCRYPT*, volume 4329 of *Lecture Notes in Computer Science*, pages 80–94. Springer, 2006.
[11] E. Dawson, K. Chen, M. Henricksen, W. Millan, L. Simpson, and S. Moon H. Lee. Dragon: A Fast Word Based Stream Cipher. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/006, 2005. http://www.ecrypt.eu.org/stream.
[12] Vincent Rijmen, Bart Van Rompay, Bart Preneel, and Joos Vandewalle. Producing collisions for panama. In *FSE '01: Revised Papers from the 8th International Workshop on Fast Software Encryption*, pages 37–51, London, UK, 2002. Springer-Verlag.
[13] G. Bertoni, J. Daemen, G. VanAssche, and M. Peeters. Radiogatún, a belt and mill hash function, 2007.
[14] Scott R. Fluhrer and David A. McGrew. Statistical analysis of the alleged rc4 keystream generator. In *FSE '00: Proceedings of the 7th International Workshop on Fast Software Encryption*, pages 19–30, London, UK, 2001. Springer-Verlag.
[15] Jovan Dj. Golic. Linear statistical weakness of alleged rc4 keystream generator. In *EUROCRYPT*, pages 226–238, 1997.
[16] Bimal K. Roy and Willi Meier, editors. *Fast Software Encryption, 11th International Workshop, FSE 2004, Delhi, India, February 5-7, 2004, Revised Papers*, volume 3017 of *Lecture Notes in Computer Science*. Springer, 2004.
[17] Ross J. Anderson and Eli Biham. Tiger: A fast new hash function. In Gollmann [18], pages 89–97.
[18] Dieter Gollmann, editor. *Fast Software Encryption, Third International Workshop, Cambridge, UK, February 21-23, 1996, Proceedings*, volume 1039 of *Lecture Notes in Computer Science*. Springer, 1996.
[19] Ronald Cramer, editor. *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, volume 3494 of *Lecture Notes in Computer Science*. Springer, 2005.