M.-J. O. Saarinen, StriBob: authenticated encryption from GOST R 34.11-2012 LPS permutation, *Mat. Vopr. Kriptogr.*, 2015, Volume 6, Issue 2, 67–78

МАТЕМАТИЧЕСКИЕ ВОПРОСЫ КРИПТОГРАФИИ
2015 Т. 6 № 2 С. 67–78

УДК: 519.719.2

StriBob: authenticated encryption from GOST R 34.11-2012 LPS permutation

M.-J. O. Saarinen

ECIT, Queens University Belfast, Belfast, UK

Получено 16.IX.2014

Authenticated encryption algorithms protect both the confidentiality and integrity of messages in a single processing pass. We show how to utilize the $L \circ P \circ S$ transform of the Russian GOST R 34.11-2012 standard hash «Streebog» to construct an efficient, lightweight algorithm for Authenticated Encryption with Associated Data (AEAD) via the Sponge scheme. The proposed algorithm "StriBob" has attractive security properties, is faster than the Streebog hash alone, twice as fast as the GOST 28147-89 encryption algorithm, and requires only a modest amount of running-time memory. StriBob is a Round 1 candidate in the CAESAR competition.

Key words: StriBob, Authenticated Encryption, GOST R 34.11-2012, Streebog, Sponge Construction, DuplexWrap, MonkeyDuplex, CAESAR

StriBob: аутентифицированное шифрование с помощью подстановки LPS из ГОСТ Р 34.11-2012

М.-Ю. О. Сааринен

Институт электроники, связи и информационных технологий, Королевский Университет Белфаста, Белфаст, Великобритания

Аннотация. Аутентифицированное шифрование обеспечивает как конфиденциальность, так и целостность данных за один проход. В работе показано, что LPS-преобразование в российском стандарте хэширования «Стрибог» в сочетании с конструкцией Sponge можно использовать для построения эффективно реализуемого низкоресурсного алгоритма аутентифицированного шифрования с ассоциированными данными. Предлагаемый алгоритм «StriBob» имеет хорошие криптографические характеристики, работает быстрее хэш-функции «Стрибог», в два раза быстрее алгоритма шифрования ГОСТ 28147-89 и использует небольшой объем оперативной памяти. Алгоритм «StriBob» является участником 1 этапа конкурса CAESAR.

Ключевые слова: StriBob, аутентифицированное шифрование, ГОСТ Р 34.11-2012, Стрибог, конструкция Sponge, DuplexWrap, MonkeyDuplex, CAESAR

Citation: *Mathematical Aspects of Cryptography*, 2015, vol. 6, no. 2, pp. 67–78 (Russian).

This work was carried out during the tenure of an ERCIM "Alain Bensoussan"Fellowship Programme at Norwegian University of Science and Technology.

1. Introduction

Since January 1, 2013, the Russian Federation has mandated the use of new GOST R 34.11-2012 hash algorithm in digital signatures [12, 15]. This hash was designed apparently in response to cryptographic weaknesses reported in the previous hash standard GOST R 34.11-94 [14, 22]. The 2012 standard, dubbed Streebog, has superficial similarities to the old 1994 standard but also features clearly AES-inspired design elements [10, 16, 24].

In contrast to the Russian approach, the U.S. NIST selected a novel Sponge-based design, Keccak, as the basis of future SHA-3 hash function standard [4, 9]. Sponge hashes diverge from more traditional Davies–Mayer [21] (SHA) and derived HAIFA [7] (Streebog) constructions in that they are based on a single keyless permutation π rather than on a keyed permutation which can seen as a special-purpose block cipher.

Furthermore, Sponge permutations may be used to achieve Authenticated Encryption in straightforward manner (see Figure 1) [3, 5]. Here both the confidentiality and integrity of a message can be guaranteed with a single processing pass, without the use of a separate encryption algorithm such as GOST 28147-89 [13] and a hash-based Message Authentication Code such as HMAC-Streebog [25]. This has clear advantages for performance and implementation footprint, which are especially useful in limited-resource applications. Even full-featured secure communications suites may be constructed from a single permutation [27].

In this note we show how to construct a modern lightweight AEAD algorithm from the core of the GOST R 34.11-2012 Streebog hash. Our proposal, "StriBob" is faster than the Streebog hash alone, has good security arguments, and runs on low-resource platforms. The proposal is a first round candidate in the U.S. NIST - funded CAESAR Competition [26, 28].

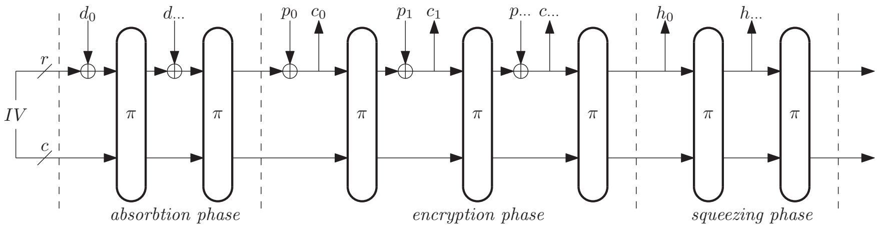

Fig. 1. A simplified view of a Sponge-based AEAD. First the padded Secret Key, Nonce, and Associated Authenticated Data - all represented by d_u words — are "absorbed" or mixed into the Sponge state. The π permutation is then used to also encrypt data p_i into ciphertext c_i (or vice versa) and finally to "squeeze" out a Message Authentication Code h_i.

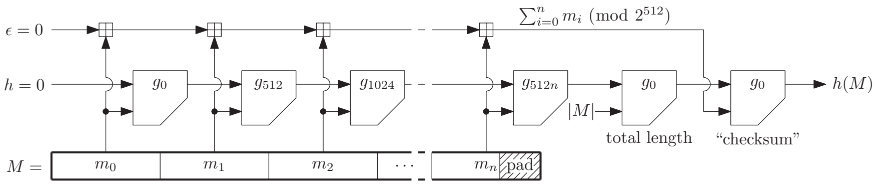

Fig. 2. Operation of STREEBOG with 512-bit output. For 256-bit hashes, the initial h value is changed to `0x010101..01` and the output $h(M)$ is truncated to 256 bits.

2. Structure of GOST R 34.11-2012

STREEBOG produces either 256-bit or 512-bit hash from a bit string of arbitrary size using the Merkle–Damgård [11, 23] iterative method (without randomization). Figure 2 gives an overview of the hashing process.

Padded message M is processed in 512-bit blocks $M = m_0 \mid m_1 \mid \cdots \mid m_n$ by a compression function $h' = g_N(h, m_i)$. The chaining variable h also has 512 bits and N denotes the index bit offset of the input block. After the last message block, there are finalization steps involving two invocations of the compression function, first on the total bit length of input, and then on checksum ϵ, which is computed over all input blocks $\bmod\ 2^{512}$.

2.1. STREEBOG Compression Function $g_N(h, m)$

The compression function $h' = g_N(h, m)$ takes as inputs a chaining variable h, message block m, a position index variable N, and produces a new chaining value h'. The compression function is constructed from a keyless 512-bit nonlinear permutation LPS and 512-bit vector XOR operations. The compression function has 12 rounds and performs a total of 25 invocations of LPS :

$$\begin{aligned}
[K_1, X_1] &= [\ \mathsf{LPS}(h \oplus N),\ m\ ],\\
[K_{i+1}, X_{i+1}] &= [\ \mathsf{LPS}(K_i \oplus C_i), \mathsf{LPS}(X_i \oplus K_i)\ ] \quad \text{for} \quad 1 \le i \le 12,\\
g_N(h, m) &= K_{13} \oplus X_{13} \oplus h \oplus m.
\end{aligned}$$

Figure 3 shows the structure of g. We can view it as a two-track substitution-permutation network where input value $h \oplus N$ and a set of 12 round constants C_i are used to key (via K_i) another substitution-permutation network operating on h. The outputs of the two tracks are finally XOR'ed together with original values of h and m. We note that h together with offset N uniquely defines all K_i subkey values for each invocation of g.

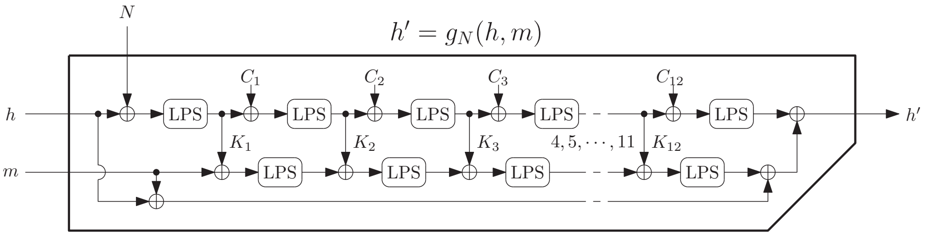

Fig. 3. Streebog compression function. All data paths, inputs, and outputs are 512-bit vectors. Here the $\oplus$ symbol denotes the XOR operation between two 512-bit vectors.

Computation of $g_N(h, m)$ requires at least 3×512 bits or 192 bytes of temporary storage, which may be preventive for ultra light-weight applications. Furthermore the $\mathrm{mod}\ 2^{512}$ summation for ϵ must be performed concurrently to the compression function.

2.2. The LPS Transform

The LPS transform is a 512-bit keyless permutation, and forms the cryptographic core of Streebog and StriBob. It is depicted in Figure 4 and consists of three stages. We abbreviate the composite function $L(P(S(x))) = (L \circ P \circ S)(x)$ as LPS . The components are:

S Nonlinear substitution. An 8×8-bit S-Box is first applied to each of the 64 bytes of data.

P Permutation. A byte transpose where the 8×8 byte matrix is reflected over its main diagonal (rows written as columns or columns written as rows).

L Linear transform. Finally the eight 64-bit words are individually subjected to a vector-matrix multiplication with a 64×64-bit matrix in $\mathbb{F}_2$.

LPS is closely related to the WhirlPool [2], which also uses 512-bit permutations. If we use the original AES-style notation, S is equivalent to `SubBytes`, P corresponds to `ShiftColumns`, and L to `MixRows`.

2.3. Security of LPS

Streebog gets all of its nonlinearity from 8-bit S-box S, which seems to have been designed to offer resistance against classical methods of cryptanalysis. Its differential bound [8] is $P = \frac{8}{256}$ and best linear approximation [20] holds with $P = \frac{28}{128}$.

The linear transform L is not randomly constructed even though it is expressed without explanation as a 64×64 binary matrix in [15]. In fact L has a byte-oriented structure as an MDS matrix with $\mathbb{F}_{2^8}$ arithmetic in a similar fashion as AES, even though this is not mentioned in the standard specification [16, 24]. Transforms S and L are effective in mixing bits of the eight 64-bit rows. Permutation P swaps rows and columns and after two rounds each input bit affects each output bit of the 512-bit state. Transform LPS has similar per-round avalanche to AES and similar resistance to Square attacks [17].

We note that structurally similar Whirlpool has been attacked with Rebound Distinguishers [18, 19]. However, these attacks may only barely reach 10 rounds of Whirlpool, and not 12 rounds used in the present construction.

3. Authenticated Encryption Algorithm StriBob

In a sponge function only a single keyless permutation π is required. We utilize the LPS transform and twelve round constants C_i of GOST R 34.11-2012 in our new design. For some vector of twelve 512-bit subkeys C_i we define a 512-bit permutation $\pi_C(X_1) = X_{13}$ with iteration

$$x_{i+1} = \mathsf{LPS}(X_i \oplus C_i) \quad \text{for} \quad 1 \le i \le 12.$$

We assume that π_C and π_K are equally strong since both C and K consist of an essentially random sets of subkeys. There is a straightforward intuitive security relation between π_K and a single instance of the full compression function g. We note that for the very first message block m, the subkeys K_i are always constant

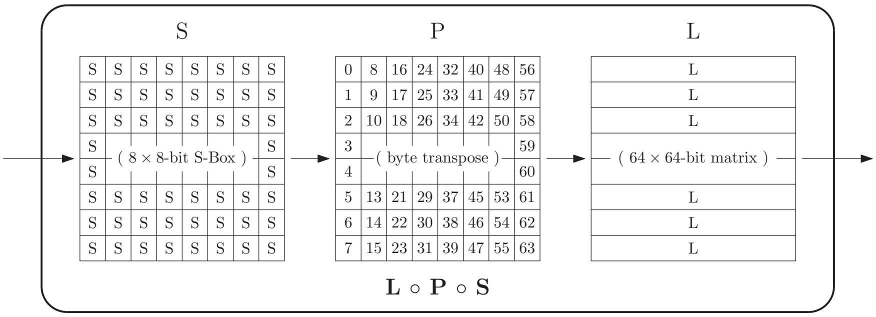

Fig. 4. LPS consists of a byte substitution layer S, byte transpose P, and a linear layer L. L may be alternatively expressed as a 8×8-byte matrix in $\mathsf{GF}(2^8)$.

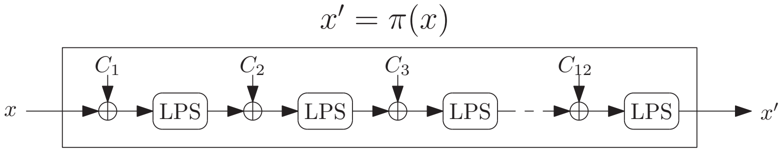

Fig. 5. The 512-bit permutation π used by STRIBOB.

as they depend on the initial constant $h = 0$ alone. We can therefore write for the first block:

$$h' = g_0(0, m) = \pi_K(m) \oplus m.$$

The output truncation after the last invocation of g of STREEBOG-256 indicates that collision resistance is expected of half of the output as well, which is exactly what we need in an $r = 256$ Sponge mode (Section 3.1.).

After careful analysis, we conjecture that the π_C permutation offers *no structural distinguishers* that are not based on some trivial property such as a priori knowledge of output value of $\pi_C(x)$ for some particular x. We use π_C alone in our final construction.

3.1. Sponge Mode and Security Parameters

The sponge function π is operated in a MULTIPLEX-like mode in order to achieve maximum flexibility [27]. This way STRIBOB may be used for plain hashing, PRNG generation and in two-party protocols in addition to Authenticated Encryption with Associated Data (AEAD).

Theorem 1. *The* DUPLEXWRAP *and* MULTIPLEX *authenticated encryption modes satisfy the following privacy and authentication security bounds:*

$$\mathrm{Adv}^{\mathsf{priv}}_{\mathsf{sbob}}(\mathcal{A}) < \frac{D+T}{2^k} + \frac{D^2+4DT}{2^{c+1}},$$
$$\mathrm{Adv}^{\mathsf{auth}}_{\mathsf{sbob}}(\mathcal{A}) < \frac{D+T}{2^k} + \frac{D^2+4DT}{2^{c+1}} + \frac{D}{2^t}$$

against any single adversary $\mathcal{A}$ if $K \overset{\$}{\leftarrow} \{0,1\}^k$, tags of t bits are used, π is a randomly chosen permutation, D is the data complexity (number of queries to target), and T is the offline attack time complexity.

Proof. See Theorem 4 of [6] and related work [1, 3].

Since $b = 512$, we choose a Sponge rate of $r = 256$ bits, which leaves capacity $c = b - r = 256$. We choose key size $k = 192$ and limit $N < 2^{60}$. As our capacity is $c \approx 254$ (couple of effective capacity bits are lost due to domain separation [27]), a 192-bit security level is comfortably reached.

3.2. Padding Example

StriBob uses BLNK padding, which is a variant of [27]. The “payload rate bytes” are the first 32 bytes of the state and byte 32 is used as padding and domain indicator. Each element is padded with a `0x01` byte and zeros to full r-bit block length so that π is called between different domains. If the domain data length is an exact multiple of r, `BLNK_END` is set at the domain indicator byte. The domain flags used in the CAESAR implementation are:

Flag name	Value	Padding bit or Domain identifier
`BLNK_END`	`0x01`	Padding marker bit
`BLNK_FIN`	`0x02`	Data element final block marker bit
`BLNK_KEY`	`0x10`	Secret key (in)
`BLNK_NPUB`	`0x20`	Public sequence number (in)
`BLNK_NSEC`	`0x30`	Secret sequence number (in / out)
`BLNK_AAD`	`0x40`	Authenticated Associated Data (in)
`BLNK_MSG`	`0x50`	Confidential Message Payload (in/out)
`BLNK_MAC`	`0x60`	Message Authentication Code (out)

Example. To illustrate the operation with CAESAR parameters, we use the 192-bit secret key “`192-bit Secret Key value`” and public nonce “`Nonces Used Once`” (16 bytes) to authenticate Associated Data “`AAD Test Vector Exact Block 32 B`” (32 bytes) and to encrypt plaintext “`This is a Test Vector for stribob192r1`” (38 bytes).

S1 StriBob uses an all-zero initial state. The first input to π is the padded secret key value:
3139322D6269742053656372657420 4B65792076616C7565**01**00000000000000
1200···00

S2 Nonce is XORed into the state before second π:
4E6F6E63657320557365642 04F6E6365**01**000000000000000000000000000000
2200···00

S3 Associated data length equals rate (32 bytes) so padding is in domain separation byte:
41414420546573742056656374 6F722045786163742042 6C6F636B2033322042
4300···00

Info. The state before encryption is:

39E876FD1FA6DB05FC681ECAC803A2A48B6CB30E6B47D9FEC94FE1E8CB3E02D4

734803FB16F36A5653DEFEBC7012C28C949172CAEC1274E19A7C5132AFE58EAC

S4 Padded plaintext blocks for π invocations 4 and 5:
546869732069732061205465737420566563746F7220666F72207374726962626F

5000···00
623139327231**01**00000000000000000000000000000000000000000000000000

5200···00

Corresponding ciphertext bytes:

6D801F8E3FCFA8259D484AAFBB7782F2EE0FC7611967BF91BB6F929CB95760BB

A808DE292F8B

S5 Authentication tag extraction, 128 bits. No need for state after this with CAESAR.

165BD9D62B3C7B7D6DC423446BE76082

4. Implementation and Performance Notes

Since the new construct requires 12 invocations of LPS per 256 bits processed in comparison to 25 invocations per 512 bits with STREEBOG, we see that the new construct is faster. Furthermore, the operational memory requirement is shrunk to approximately 25% of the original.

Low-resource software platforms. For a software implementation on a low-resource 8 or 16-bit CPUs and SoCs (e.g. RFID, Smart Card, Sensor, Ubiquitous / IoT category systems) it is advantageous to realize the linear layer L as a matrix multiplication in $\mathbb{F}_{2^8}$. Multiplication in a small finite field may be implemented via discrete logarithm and exponentiation tables: $AB = \exp(\log A + \log B)$. Note that STREEBOG and therefore STRIBOB uses a special bit-inverted representation for field elements [16].

One can combine the S-Box lookup and discrete logarithm table into a single 8×8-bit lookup table $\log(S(x))$. The 8×8 matrix over $\mathbb{F}_{2^8}$ M (representing L) may be stored in log form. Required addition $x + y \ (\bmod\ 2^8 - 1)$ may be implemented by adding carry bit $\lfloor \frac{x+y}{2^8} \rfloor$ of addition $x + y \ (\mathrm{mod} 2^8)$ to the 8-bit sum itself: set $\exp(255) = \exp(0)$ in this case.

As the transpose P may be coded into loops (switching the column and row indexes), the implementation of LPS requires a total of $256 + 256 + 8 \times 8 = = 576$ bytes for storage. Unfortunately C_i round constants still require $12 \times 64 = = 768$ bytes. One may consider a variant that uses a fast pseudorandom generator such as some Fibonacci-based sequence or linear congruential generator instead of a truly random C to further compress the implementation.

Medium- to high-resource software platforms. A software implementation on system with a medium- or high-performance CPUs (e.g. server, desktop, laptop, or tablet category systems) can utilize $8 \times 8 \times 64$-bit lookup tables that combine S and L, requiring a total of 16 kB and 768 B for round constants. The compression function code itself is very compact.

Results of wall-clock throughput measurements on a typical desktop system[1]:

Algorithm	**Throughput**
AES - 128 / 192 / 256	109.2 / 90.9 / 77.9 MB/s
SHA - 256 / 512	212.7 / 328.3 MB/s
GOST 28147-89	53.3 MB/s
GOST R 34.11-1994	20.8 MB/s
GOST R 34.11-2012	109.4 MB/s
StriBob	115.7 MB/s

Hardware. Use of AES instruction set significantly boosts AES performance, but so would similar hardware optimizations for Streebog and StriBob. Since the rate of StriBob is twice that of AES and there are 12 rounds (indicating roughly equivalent critical path), we may expect StriBob hardware implementations to be significantly faster than AES.

5. Conclusions

We propose StriBob, an Authenticated Encryption with Associated Data (AEAD) algorithm based on the GOST R 34.11-2012 hash standard. The new algorithm is faster than the hash standard alone, twice as fast as the GOST 27147-89 encryption algorithm, and is competitive against AES. StriBob is a first round candidate in the CAESAR competition of the U.S. National Institute of Standardization and Technology [26, 28].

A strong security relation exists between StriBob's π function and the compression function g of GOST R 34.11-2012, giving us a significant level of confidence in its security. Furthermore the underlying Sponge mode of operation is provably secure. We feel that our proposal offers a viable alternative to present GOST standards.

[1] Measurements were made on a single core of an Intel Core i7 860 @ 2.80 GHz system running Ubuntu Linux 13.10 (amd64) with gcc 4.8.1. The AES, SHA, GOST 28147-89 and R 34.11-1994 timings with were measured with Ubuntu default OpenSSL (1.0.1e). A. Degtyarev's implementation (0.11) was used for the GOST 34.11-2012 benchmark. The StriBob reference implementation is by author.

References

[1] Andreeva E., Mennink B., and Preneel B., “Security reductions of the second round SHA-3 candidates.”, July 2010, IACR ePrint 2010/381, eprint.iacr.org/2010/381.

[2] Barreto P. S. L. M., Rijmen V., “The Whirlpool hashing function. NESSIE Algorithm Specification”, 2000, Revised May 2003, www.larc.usp.br/pbarreto/WhirlpoolPage.html.

[3] Bertoni G., Daemen J., Peeters M., Assche G. V., “Duplexing the sponge: Single-pass authenticated encryption and other applications.”, SAC 2011 *(2011)*, Lect. Notes Comput. Sci., **7118**, 320–337.

[4] Bertoni G., Daemen J., Peeters M., Assche G. V., “The Keccak reference, version 3.0”, NIST SHA3 Submission Document, January 2011.

[5] Bertoni G., Daemen J., Peeters M., Assche G. V., “Permutation-based encryption, authentication and authenticated encryption”, *DIAC 2012*, 2012, keccak.noekeon.org/KeccakDIAC2012.pdf.

[6] Bertoni G., Daemen J., Peeters M., Assche G. V., Keer R. V., CAESAR submission: Keyak, **1** (March 2014), competitions.cr.yp.to/round1/keyakv1.pdf.

[7] Biham E., and Dunkelman O., “A framework for iterative hash functions — HAIFA”, July 2007, IACR ePrint, eprint.iacr.org/2007/278.

[8] Biham E., Shamir A., *Differential cryptanalysis of the Data Encryption Standard*, Springer, 1993.

[9] Chang S., Perlner R., Burr W. E., Turan M. S., Kelsey J. M., Paul S., Bassham L. E., “Third-round report of the SHA-3 cryptographic hash algorithm competition”, Tech. Rep. NISTIR 7896, Nat. Inst. Stand. Technol., November, 2012.

[10] Daemen J., Rijmen V., The design of Rijndael: AES - the Advanced Encryption Standard, Springer, 2002.

[11] Damgård I., “A design principle for hash functions”, *Lect. Notes Comput. Sci.*, **435** (1989), 416–427.

[12] Dolmatov V., Degtyarev A., “GOST R 34.11-2012: Hash Function”, August 2013, ITEF RFC 6986.

[13] “GOST. Cryptographic protection for data processing system. GOST 28147-89”, 1989 (in Russian).

[14] “GOST. Cryptographic protection of information, hash function. GOST R 34.11-94”, 1994 (in Russian).

[15] “GOST. Information technology. Cryptographic protection of information, hash function. GOST R 34.11-2012”, 2012 (in Russian).

[16] Kazymyrov O., Kazymyrova V., “Algebraic aspects of the Russian hash standard GOST R 34.11-2012”, CTCrypt’13 (June 23-24, 2013, Ekaterinburg, Russia), 2013, IACR ePrint, eprint.iacr.org/2013/556.

[17] Knudsen L., Wagner D., “Integral cryptanalysis (extended abstract)”, FSE 2002, Lect. Notes Comput. Sci., **2365**, Springer, 2002, 112–127.

[18] Lamberger M., Mendel F., Rechberger C., Rijmen V., Schläffer M., “Rebound distinguishers: Results on the full whirlpool compression function”, ASIACRYPT ’09, Lect. Notes Comput. Sci., **5912**, ed. Matsui M., 2009, 126–143.

[19] Lamberger M., Mendel F., Schläffer M., Rechberger C., Rijmen V., “The rebound attack and subspace distinguishers: Application to Whirlpool”, J. Cryptology, **28**: 2 (2015), 257–296.

[20] Matsui, M., “Linear cryptoanalysis method for DES cipher”, EUROCRYPT ’93, Lect. Notes Comput. Sci., **765**, ed. Helleseth T., 1994, 386–397.

[21] Matyas S., Meyer C., Ossas J., “Generating strong one-way functions with cryptographic algorithm”, IBM Technical Disclosure Bulletin, **27** (1985), 5658–5659.

[22] Mendel F., Pramstaller N., Rechberger C., Kontak M., Szmidt J., “Cryptanalysis of the GOST hash function.”, CRYPTO 2008, Lect. Notes Comput. Sci., **5157**, 2008, 162–128.

[23] Merkle R., *Secrecy, Authenticatication, and public key systems*, PhD thesis, Stanford University, 1979.

[24] NIST, “Advanced Encryption Standard (AES)”, FIPS 197, 2001.

[25] NIST, “The keyed-hash message authentication code (HMAC)”, FIPS 198-1, July 2008.

[26] NIST, Bernstein D., “CAESAR call for submissions”, January 2014, competitions.cr.yp.to/caesar-call.html.

[27] Saarinen M.-J. O., “Beyond modes: Building a secure record protocol from a cryptographic sponge permutation”, CT-RSA 2014, Lect. Notes Comput. Sci., **8366**, 2014, 270–285.

[28] Saarinen M.-J. O., “The STRIBOBr1 authenticated encryption algorithm”, CAESAR, 1st Round, March 2014, www.stribob.com.