CryptoLUX > Lightweight Hash Functions

Lightweight hash function are lightweight cryptographic primitives.

The NIST provides figures for hardware implementation of the SHA-3 finalists aimed at optimizing the area[1]. For a 0.09 μm technology, the best they can achieve is 9,200 GE for Grøstl[2]; Keccak[3] (the winner of the competition) requiring at least 15,200 GE. These are way too much for, say, RFID tags. That is why lightweight hash functions have been proposed. A first batch of these is made of MAME[4] and Lesamnta-LW[5] but these were not really light: the compression function alone, intended to be used in a Merkle-Damgård construction, requires 8,200 GE in both cases. Therefore, we shall not treat these two here.

Hash Function Design

Desirable Properties

Here, "impossible" means "impossible for an adversary having some reasonable computing power" where "reasonable" is actually a complex concept. Consider a hash function H; we want it to be (at least):

1. Collision resistant: it is "impossible" to find x and y such that H(x)=H(y).
2. Preimage resistant: given a digest d, it is "impossible" to find x such that H(x)=d.
3. Second preimage resistant: given y, it is "impossible" to find x≠y such that H(x)=H(y).

Merkle-Damgård

Basic idea

Consider a compression function h mapping $\{0,1\}^n x \{0,1\}^k$ to $\{0,1\}^n$, a fixed and public initialization vector IV of $\{0,1\}^n$ and a message $(m_0, m_1, ..., m_{L-1})$ where each m_i is a block of k bits. Then we can build a hashfunction H where:

- $c_0 = IV$
- $c_{i+1} = h(c_i, m_i)$
- $H(m) = d = c_L$

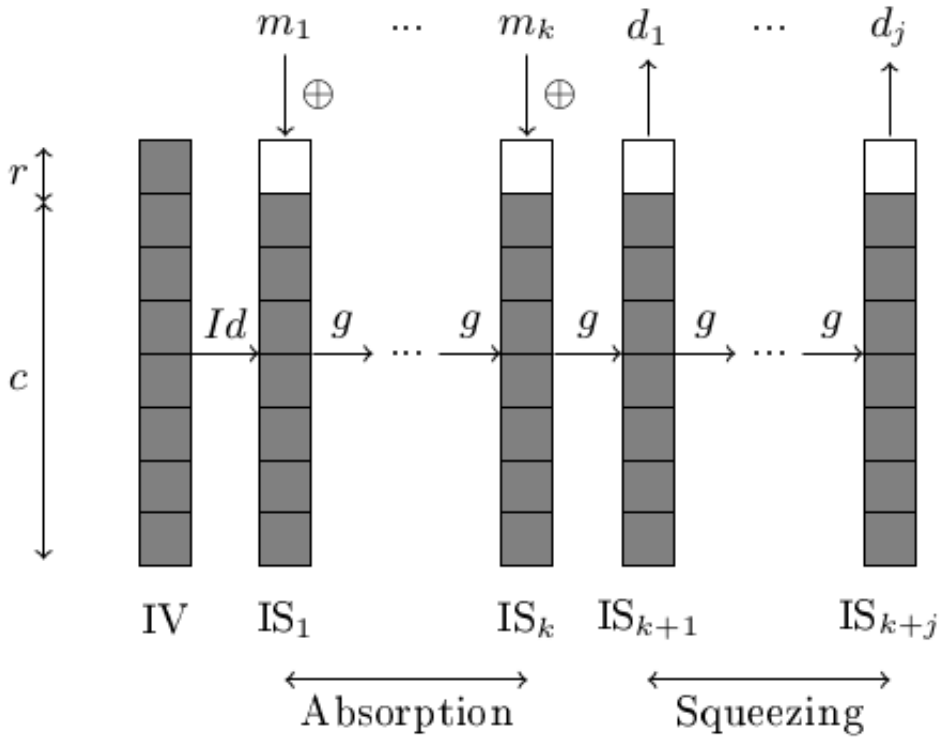
One of the main interest of this construction is that such a compression function h can easily be built out of a block cipher using an appropriate mode, for instance Davies-Meyer. However, if it is used in this simple way, it is vulnerable to, for example, length extension attacks.

Strengthened Merkle-Damgård

To prevent attacks based on the extension of the input message, a stronger construction should be used. The idea is to append the length of the input message to the end of the message itself combined

with a padding so as to have a padded message with a length which is a multiple of k. That way, length-extension-based attacks are prevented. This construction is the *strengthened Merkle-Damgård* construction.

Sponge



Principle of a sponge function.

The sponge construction is a method to build a hash function from a publicly known unkeyed permutation (P-Sponge) or random function (T-Sponge). It was introduced by Bertoni et al., the *Keccak team*, in 2007 [6]. Sponges can also be used to provided stream ciphers or even authenticated encryption, see the duplex construction.

General Idea

The principle of a sponge is fairly simple and is described on the Figure on the right. The message is first padded (more on that later) and sliced into blocks of r bits. Then, an internal state of (r+c) bits is initialized so as to contain only zeroes. r is called *(bit)rate* and c is called *capacity*. The digest is obtained by 'absorbing' the padded message and then 'squeezing' the internal state. The absorption phase consists in:

1. Replacing the first r bits of the internal by the xor of these bits and one r-bits block with (or possibly by the r-bits block directly)
2. Replacing the internal state by its image by the update function.

Then, the squeezing is made of d/r steps where d is the size of the digest in bits. Each step consists in:

1. Storing the first r bits of the internal state

2. Replacing the internal state by its image by the update function.

The r-bits block thus obtained are concatenated to obtain the digest.

It has been proven that the truncated output of a sponge can be distinguished from that of a random oracle with a probability which is essentially $N/2^{c/2}$ where c is the capacity and N is the number of calls to the update function. This is the so-called *flat sponge claim*.[7]

The padding used is usually very simple. Although the designers of the sponge do not suggest one in particular in their seminal paper[3], the one used in practice consists simply in adding one bit equal to 1 at the end of the message and then fill it with zeroes until it reaches a length which is a multiple of r. It is, as we can see, much simpler than the one used in a strengthened Merkle-Damgård construction.

P-Sponge and T-Sponge

P(ermutation)-Sponge and T(ransformative)-Sponge are sponge models using respectively a random permutation and a random function to update their internal state. In the paper in which the sponge functions were introduced[6], it was shown that sponges with capacity c, rate r and digest size n absorbing messages of length $m<2^{c/2}$ are such that different attacks require the following number of calls to the update function.

Case	Preimage	Second Pre-image	Collision	Cycle finding
T-Sponge	min(n, c+r)	min(n, c-$\log_2$(m))	min(n, c)/2	(c+r)/2
P-Sponge	c-1	min(n, c/2)		c+r

JH-like construction

What we call JH-like sponge is a structure similar to that of the hash function JH[8], finalist of the SHA-3 competition. It differs from the regular sponge by having the message blocks injected twice: first, like in a sponge, before the call to the updating function and then on the opposite side of the internal state after this call. Both times, the same message block is injected.

The JH hash function uses a strengthened Merkle-Damgård-style padding consisting in appending the length of the message to the last block, unlike the simple padding of the sponge. Note however that functions based on the same construction may use different paddings. For instance, SipHash only appends the length of the message modulo 256.

Notations

We call *internal state size* the size in bits of the complete internal state of a sponge (rate plus capacity) and the size of the chaining value in the case of a Merkle-Damgård construction. The *rate* is the size of the message block treated at each round.

Summary

Summary of the main characteristic								
Presentation			**Cryptographic Prop**					
name	designers	reference (design)	digest size	rate	internal state size	structure	preimage	sec pre
ARMADILLO2	Badel et al.	CHES 10[9]	80	48	256	Merkle-Damgård	2^{80}	2^{80}
			128	64	384		2^{128}	2^{12}
			160	80	480		2^{160}	2^{16}
			192	96	576		2^{192}	2^{19}
			256	128	768		2^{256}	2^{25}
DM-PRESENT	Poschmann	PhD Thesis (09)[11]	64	80 128	64	Merkle-Damgård (Davies-Meyer)	64	Nor
GLUON	Berger et al.	AFRICACRYPT 12[13]	128	8	136	T-Sponge	2^{128}	2^{64}
			160	16	176		2^{160}	2^{80}
			224	32	256		2^{224}	2^{11}
PHOTON	Guo et al.	CRYPTO 11[15]	80	20 / 16	100	P-Sponge	2^{64}	2^{40}
			128	16	144		2^{112}	2^{64}
			160	36	196		2^{124}	2^{80}

			224	32	256		2^{192}	2^{112}
			256	32	288		2^{224}	2^{128}
QUARK	Aumasson et al.	CHES 10[16]	136	8	136	P-Sponge	2^{128}	2^{64}
			176	16	176		2^{160}	2^{80}
			256	32	256		2^{224}	2^{112}
SipHash-2-4	Aumasson et al.	INDOCRYPT 12[17]	64	64	256	JH-style T-Sponge	2^{64}	2^{64}
SPN-Hash	Choy et al.	AFRICACRYPT 12[18]	128	256	128	JH-style P-Sponge	2^{128}	$2^{??}$
			256	512	256		2^{256}	$2^{??}$
SPONGENT	Bogdanov et al.	CHES 11[19]	80	8	88	P-Sponge	2^{80}	2^{40}
			128	8	136		2^{120}	2^{64}
			160	16	176		2^{144}	2^{80}
			224	16	240		2^{208}	2^{112}
			256	16	272		2^{240}	2^{128}

Descriptions

ARMADILLO2

- Article: *ARMADILLO: a Multi-Purpose Cryptographic Primitive Dedicated to Hardware*, CHES 10[9]
- Authors: Stephane Badel, Nilay Dagtekin, Jorge Nakahara Jr, Khaled Ouafi, Nicolas Reffe, Pouyan

Sepehrdad, Petr Susil, Serge Vaudenay

ARMADILLO2 is a multi-puprose primitive intended to be used as a FIL-MAC (application I), for hashing and digital signatures (application II) and as a PRNG and PRF (application III). It has been broken by Naya-Plasencia and Peyrin[10] who managed to find collisions when it is used as a hash function in very small time (few seconds on a regular PC).

This primitive was in the process of being patented by oridao when the article was published.

DM-PRESENT

- Article: *Lightweight Cryptography: Cryptographic Engineering for a Pervasive World*, Poschmann's PhD thesis (09)[11]
- Author: Poschmann, Alex

DM-PRESENT is simply a Merkle-Damgård scheme where the compression function is the block cipher PRESENT in Davies-Meyer mode. DM-PRESENT-80 is based on PRESENT-80 and DM-PRESENT-128 on PRESENT-128. No security claims are made about collision or second preimages as we can read in Section 6.5.1 of Poschmann's PhD Thesis[11]:

> Such hash functions will only be of use in applications that require the one-way property and 64-bit security.

GLUON

- Article: *The GLUON Family: A Lightweight Hash Function Family Based on FCSRs*, AFRICACRYPT 12[13]
- Authors: Thierry P. Berger, Joffrey D'Hayer, Kevin Marquet, Marine Minier, and Gael Thomas

GLUON is a T-sponge, meaning that it is a sponge with a non-injective update function. The said function is based on the software oriented stream-ciphers X-FCSR-v2 and F-FCSR-H-v3: the internal state of the sponge is padded and loaded into a so-called FCSR (Feedback with Carry Shift Register) which is clocked a fixed amount of time. Then, some cells of the FCSR are xored to form the first word of the next internal state, the FCSR is clocked, the same words are xored to form the second word of the next internal state, etc.

The update function of GLUON-64 is many to one and has a behaviour which is very different from that of a random function.

PHOTON

- Article: *The PHOTON Family of Lightweight Hash Functions*, CRYPTO 11[15]
- Authors: Jian Guo, Thomas Peyrin, and Axel Poschmann

Photon is a P-Sponge based on an AES-like permutation. For the smallest security parameter (PHOTON-80/20/16), the bitrate during absorption is 20 but it is equal to 16 during the squeezing

phase. The throughput figures given correspond to throughput when outputting long messages as these are the ones usually given. However, the figures for shorter messages are smaller (i.e. better) for PHOTON.

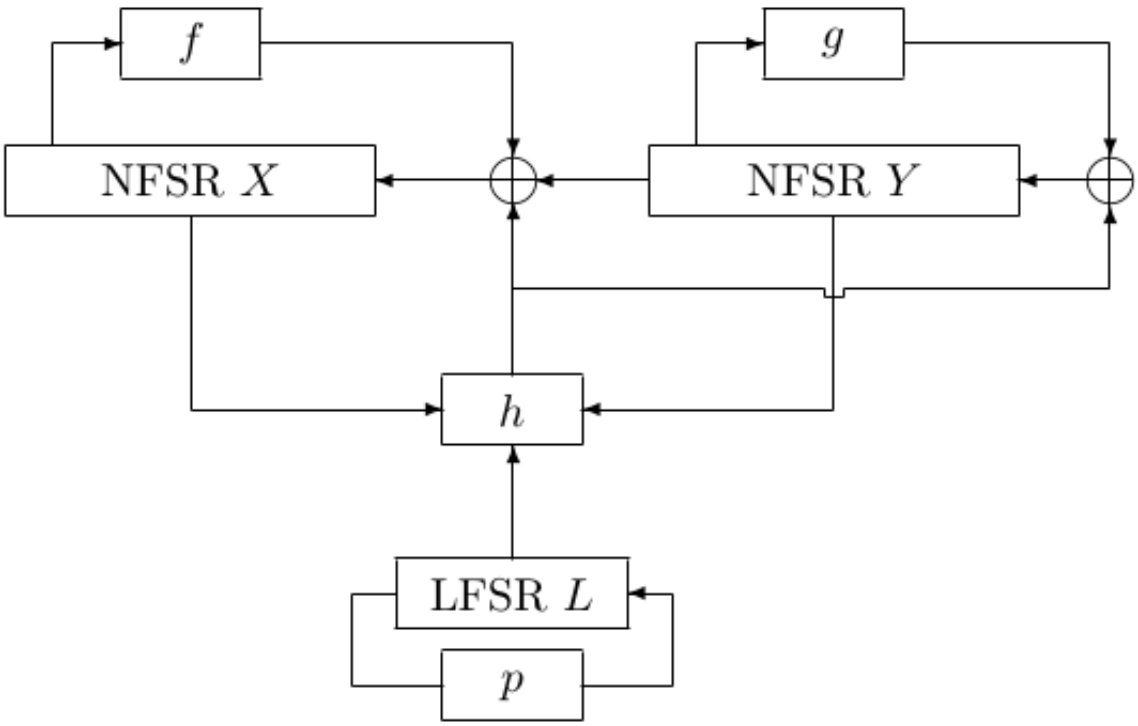
The permutation consists in 12 iterations (for every security parameters) of the following sequence of transformations performed over a square of dxd nibbles of 4-bits (8-bits for the largest version). As we can see, the AES cipher has obviously been a great influence for the design of the permutation.

1. AddConstant: a round dependent constant is xored into every nibble and then a cell dependent constant is xored as well.
2. SubCells: A S-box is applied on every nibbles. If the nibbles are 4-bits long then the PRESENT S-box is used, otherwise it is the AES's.
3. ShitRows: Identical to the AES's.
4. MixColumnsSerial: The nibbles are considered like elements of GF(2^4) (or GF(2^8) for the largest security parameter) and each column is multiplied by a MDS matrix designed so as to be efficiently implemented in hardware.

The design of the permutation used to update the sponge is close to the LED cipher which was designed later by the same people. As said above, the AES was also a great influence.

More information on this primitive (including a reference implementation) can be found on this googlesite set up by its authors.

QUARK



The QUARK permutation

- Article: *Quark: a lightweight hash*, CHES 10[16]

- Authors: Jean-Philippe Aumasson, Luca Henzen, Willi Meier, Maria Naya-Plasencia

QUARK is a P-Sponge with a hardware oriented permutation inspired by the lightweight block ciphers KTANTAN and KATAN and the hardware oriented stream cipher Grain. The smallest version (136 bits long digest) is called U-QUARK, the middle one (176 bits long digest) D-QUARK and the longest (256 bits long digest) S-QUARK.

The update function maps an element of $\{0,1\}^b$ to $\{0,1\}^b$ by loading each half in a distinct NSFR of length b/2 and then clocking these 4xb times. The NSFR's are connected to each other and to a small LFSR of length log(4b), see Figure on the side. The functions f,g and h are boolean functions chosen for their non-linearity, resilience, algebraic degree and density. f and g are identical for all versions and borrowed from Grain-v1 while h depends on the instance.

The authors claim these possible uses:

> The following primitives can then be realized:
> - Message authentication code (MAC);
> - Pseudorandom generator;
> - Stream cipher;
> - Random-access stream cipher;
> - Key derivation function.
>
> Furthermore, the Quark instances can easily be modified to operate in the duplex construction [...] to allow the realization of functionalities as authenticated encryption or reseedable pseudorandom generators.

A modified version of the sponge used is used in C-QUARK, an authenticated encryption scheme.

SipHash

- Article: *SipHash: a fast short-input PRF*, INDOCRYPT 12[17]
- Authors: Aumasson, J. P., & Bernstein, D. J.

SipHash has a ARX structure which was inspired by BLAKE and Skein. It provides a family of keyed functions mapping $\{0,1\}^*$ to $\{0,1\}^{64}$ and is aimed at a use as MAC or in hash tables. It has a structure similar to JH, like SPN-Hash and uses a padding taking the length of the message into account as well. However, it consists simply in adding a byte with the length of the message modulo 256.

It is not claimed to be collision resistant and obviously is not due to the small digest size.

SPN-Hash

- Article: *SPN-hash: improving the provable resistance against differential collision attacks*, AFRICACRYPT 12[18]
- Authors: Choy, J., Yap, H., Khoo, K., Guo, J., Peyrin, T., Poschmann, A., & Tan, C. H.

The main interest of this hash function is its provable security against differential collision attacks. It is a JH-like structure using, as its name indicates, a permutation based on SPN. The structure of the SPN is based on that of the AES: first, 8x8 S-boxes are applied on each byte of the internal state. The S-box used is exactly that of the AES. Then, a more complex mixing layer is applied; its design being the main focus of the article; optimal diffusion and lightweightness in hardware being both evaluated. At last, round constants are xored in the internal state in fashion similar to those of LED and PHOTON. These operations are iterated 10 times for all security parameters.

The authors also mention a 512 bits version of the design but it is not fully specified and hence hardware benchmarking was rendered impossible as we can see on page 10 of the original paper:[18]

> 512-bit SPN-Hash. The choice of matrix for the 16 × 16 MDS is left open to the reader.

The padding used is the same as in a strengthened Merkle-Damgård: the length of the message is appended to the last block.

SPONGENT

- Article: *SPONGENT: A lightweight hash function*, CHES 11[19]
- Authors: Bogdanov, A., Knežević, M., Leander, G., Toz, D., Varıcı, K., & Verbauwhede

SPONGENT can be seen as a P-Sponge where the permutation is a modified version of the block cipher PRESENT. These primitives actually have several designers in common.

The number of rounds of the PRESENT-like permutation ranges from 45 for SPONGENT-88 to 140 for SPONGENT-256. Each round consists in:

1. xoring of the content of a LFSR clocked at each round (can be seen as a round constant)
2. applying an S-box layer with a 4x4 S-box satisfying the same criteria as the PRESENT S-box
3. permuting the bits in a fashion similar to PRESENT.

There is no attack on SPONGENT to the best of our knowledge except for linear distinguishers for reduced-round versions[20].

Notes

1. ↑ 1.0 1.1 1.2 1.3 To the best of our knowledge.
2. ↑ The authors claim:
   > We comment that SipHash is not meant to be, and (obviously) is not, collision-resistant.
3. ↑ 3.0 3.1 The authors point out that the JH structure circumvents the MitM attack on P-Sponges causing second-preimage resistance to be of $2^{c/2}$ for P-Sponges. However, they do not make a clear security claim so it remains unclear whether they claim $2^{c/2}$ (as in a flat sponge claim) or something else (2^{c}?).

References

1. ↑ Kavun, E. B., & Yalcin, T. (2012, March). On the suitability of SHA-3 finalists for lightweight applications. In ser. The Third SHA-3 Candidate Conference. pdf at nist.gov
2. ↑ Gauravaram, P., Knudsen, L. R., Matusiewicz, K., Mendel, F., Rechberger, C., Schläffer, M., & Thomsen, S. S. (2008). Grøstl–a SHA-3 candidate. Submission to NIST. pdf at uclouvain.be
3. ↑ 3.0 3.1 Bertoni, G., Daemen, J., Peeters, M., & Van Assche, G. (2008). Keccak specifications. Submission to NIST, 42. pdf at noekeon.org
4. ↑ Yoshida, H., Watanabe, D., Okeya, K., Kitahara, J., Wu, H., Küçük, Ö., & Preneel, B. (2007). MAME: A compression function with reduced hardware requirements. In Cryptographic Hardware and Embedded Systems-CHES 2007 (pp. 148-165). Springer Berlin Heidelberg. pdf at springer.
5. ↑ Hirose, S., Ideguchi, K., Kuwakado, H., Owada, T., Preneel, B., & Yoshida, H. (2011). A lightweight 256-bit hash function for hardware and low-end devices: lesamnta-LW. In Information Security and Cryptology-ICISC 2010 (pp. 151-168). Springer Berlin Heidelberg. pdf at springer
6. ↑ 6.0 6.1 Bertoni, G., Daemen, J., Peeters, M., & Van Assche, G. (2007, May). Sponge functions. In ECRYPT hash workshop (Vol. 2007). pdf at psu.edu
7. ↑ Bertoni, G., Daemen, J., Peeters, M., & Van Assche, G. (2008). On the indifferentiability of the sponge construction. In Advances in Cryptology–EUROCRYPT 2008 (pp. 181-197). Springer Berlin Heidelberg. pdf at eprint.iacr.org
8. ↑ Wu, H. (2011). The hash function JH. Submission to NIST (round 3). pdf at uclouvain.be
9. ↑ 9.0 9.1 9.2 Badel, S., Dağtekin, N., Nakahara Jr, J., Ouafi, K., Reffé, N., Sepehrdad, P., ... & Vaudenay, S. (2010). ARMADILLO: a multi-purpose cryptographic primitive dedicated to hardware. In Cryptographic Hardware and Embedded Systems, CHES 2010 (pp. 398-412). Springer Berlin Heidelberg. pdf at psu.edu
10. ↑ 10.0 10.1 Naya-Plasencia, M., & Peyrin, T. (2012, January). *Practical cryptanalysis of ARMADILLO2*. In Fast Software Encryption (pp. 146-162). Springer Berlin Heidelberg. pdf at christina-boura.info
11. ↑ 11.0 11.1 11.2 11.3 Poschmann, A. *Lightweight Cryptography: Cryptographic Engineering for a Pervasive World*. PhD Thesis from Faculty of Electrical Engineering and Information Technology Ruhr-University Bochum, Germany. pdf at eprint.iacr.org
12. ↑ Koyama, T., Sasaki, Y., & Kunihiro, N. (2013). Multi-differential cryptanalysis on reduced DM-PRESENT-80: collisions and other differential properties. In Information Security and Cryptology–ICISC 2012 (pp. 352-367). Springer Berlin Heidelberg. pdf at springer
13. ↑ 13.0 13.1 13.2 Berger, T. P., D'Hayer, J., Marquet, K., Minier, M., & Thomas, G. (2012). *The GLUON family: a lightweight Hash function family based on FCSRs*. In Progress in Cryptology-AFRICACRYPT 2012 (pp. 306-323). Springer Berlin Heidelberg. pdf at inria.fr
14. ↑ Perrin, L., & Khovratovich, D. (2014, March). *Collision Spectrum, Entropy Loss, T-Sponges, and Cryptanalysis of GLUON-64.* In Fast Software Encryption 2014 (to appear). pdf at eprint.iacr.org
15. ↑ 15.0 15.1 15.2 Guo, J., Peyrin, T., & Poschmann, A. (2011). *The PHOTON family of lightweight hash functions*. In Advances in Cryptology–CRYPTO 2011 (pp. 222-239). Springer Berlin Heidelberg. pdf at google.com
16. ↑ 16.0 16.1 16.2 Aumasson, J. P., Henzen, L., Meier, W., & Naya-Plasencia, M. (2013). Quark: A lightweight hash. Journal of cryptology, 26(2), 313-339. pdf at springer

17. ↑ 17.0 17.1 17.2 Aumasson, J. P., & Bernstein, D. J. (2012). SipHash: a fast short-input PRF. In Progress in Cryptology-INDOCRYPT 2012 (pp. 489-508). Springer Berlin Heidelberg. pdf at eprint.iacr.org
18. ↑ 18.0 18.1 18.2 18.3 Choy, J., Yap, H., Khoo, K., Guo, J., Peyrin, T., Poschmann, A., & Tan, C. H. (2012). SPN-hash: improving the provable resistance against differential collision attacks. In Progress in Cryptology-AFRICACRYPT 2012 (pp. 270-286). Springer Berlin Heidelberg. pdf at ntu.edu.sg
19. ↑ 19.0 19.1 19.2 Bogdanov, A., Knežević, M., Leander, G., Toz, D., Varıcı, K., & Verbauwhede, I. (2011). SPONGENT: A lightweight hash function. In Cryptographic Hardware and Embedded Systems–CHES 2011 (pp. 312-325). Springer Berlin Heidelberg.pdf at kuleuven.be
20. ↑ 20.0 20.1 Abdelraheem, M. A. (2013). *Estimating the probabilities of low-weight differential and linear approximations on PRESENT-like ciphers*. In Information Security and Cryptology–ICISC 2012 (pp. 368-382). Springer Berlin Heidelberg. pdf at springer