LHash: A Lightweight Hash Function (Full Version)*

Wenling Wu, Shuang Wu, Lei Zhang, Jian Zou, and Le Dong

Institute of Software, Chinese Academy of Sciences, Beijing 100190, P.R. China
{wwl,zhanglei}@tca.iscas.ac.cn

Abstract. In this paper, we propose a new lightweight hash function supporting three different digest sizes: 80, 96 and 128 bits, providing preimage security from 64 to 120 bits, second preimage and collision security from 40 to 60 bits. LHash requires about 817 GE and 1028 GE with a serialized implementation. In faster implementations based on function T, LHash requires 989 GE and 1200 GE with 54 and 72 cycles per block, respectively. Furthermore, its energy consumption evaluated by energy per bit is also remarkable. LHash allows to make trade-offs among security, speed, energy consumption and implementation costs by adjusting parameters. The design of LHash employs a kind of Feistel-PG structure in the internal permutation, and this structure can utilize permutation layers on nibbles to improve the diffusion speed. The adaptability of LHash in different environments is good, since different versions of LHash share the same basic computing module. The low-area implementation comes from the hardware-friendly S-box and linear diffusion layer. We evaluate the resistance of LHash against known attacks and confirm that LHash provides a good security margin.

Keywords: lightweight, hash function, sponge function, Feistel, security, performance.

1 Introduction

RFID products have been widely implemented and deployed in many aspects in our daily life, *e.g.* automated production, access control, electronic toll collection, parking management, identification and cargo tracking. The need for security in RFID and sensor networks is dramatically increasing, which requires secure yet efficiently implementable cryptographic primitives including secret-key ciphers and hash functions. In such constrained environments, the area and power consumption of a primitive usually comes to the fore, and standard algorithms are often prohibitively expensive to implement. Hence, lightweight cryptography has become a hot topic. A number of lightweight cryptographic algorithms are proposed, such as stream cipher Trivium [11] and Grain [16], block cipher PRESENT [5], HIGHT [18], LBlock [35], LED [15], Piccolo [31] and PRINCE [7]. Recently, some significant works on lightweight hash functions have also been performed. In [6], the proposed lightweight hash function is constructed from block cipher PRESENT in Hirosei's double-block mode [17]. The ARMADILLO [2] hash function proposed at CHES 2010 was found to be insecure. Then a new version of ARMADILLO (version 3) [33] was proposed at CARDIS 2012. QUARK [1] uses sponge structure [3] and internal permutation similar to feedback shift registers used in Grain. PHOTON [14] proposed at CRYPTO 2011 and Spongent [4] proposed at CHES 2011 also use sponge structure, but different internal permutations, which are based on AES-like and PRESENT-like structures, respectively. Moreover, Kavun et al [20] presented a lightweight implementation of Keccak at RFIDSec 2010.

In this paper, we propose a new lightweight hash function LHash with digest sizes from 80 to 128 bits. LHash is based on extended sponge functions framework, which allows trade-offs among security, speed, energy consumption and implementation cost by

* Full version of the extended abstract published in the proceedings of Inscrypt 2013.

adjusting parameters. The internal permutation is designed using a structure, named as Feistel-PG, which is an extended variant of improved generalized Feistel. Feistel-PG has faster diffusion, shorter impossible differential paths and integral distinguishers than similar structures. The S-box and MDS linear layer used in the internal permutation are designed to be hardware-friendly. Both of them have very compact hardware implementation. The MDS linear layer has an iterated implementation, which is similar to and even more compact than the linear layer used in PHOTON. We present that LHash achieves remarkably compact implementation in hardware. In our smallest implementation, the area requirements are 817 and 1028 GE with 666 and 882 cycles per block, respectively. Meanwhile, its efficiency on energy consumption evaluated by the metric of energy per bit proposed in [31] is the smallest class among current lightweight hash functions in literature. Especially, for the competitors with similar preimage and collision resistance levels, it also compete well in terms of area and throughput tradeoff as shown in Figure 7. Comparative results regarding the hardware efficiency for lightweight hash functions are summarized in Table 1. Regarding security, the internal permutation of LHash provides a good security margin against all kinds of attacks, including differential attack, impossible differential attack, zero-sum distinguisher, rebound attack etc. Since LHash is built on the internal permutation using extended sponge structure, we believe that the security bounds claimed can be reached.

Table 1. Comparison of LHash with existing lightweight hash functions

Algorithm	Parameters					Bounds			Area [GE]	Cycle [clks]	Throughput [kbps]		FOM [nb/clk/GE^2]		Energy/bit[*]
	n	b	c	r	r'	Pre	2nd Pre	Col			long	96-bit	long	96-bit	
LHash	80	96	80	16	16	64	40	40	817	666	2.40	1.44	35.96	21.59	34008
									989	54	29.63	17.78	302.9	181.75	3338
	96	96	80	16	16	80	40	40	817	666	2.40	1.31	35.96	19.63	34008
									989	54	29.63	16.16	302.9	165.2	3338
	128	128	112	16	32	96	56	56	1028	882	1.81	1.21	17.13	11.44	56669
									1200	72	22.22	14.81	154.3	102.89	5400
	128	128	120	8	8	120	60	60	1028	882	0.91	0.40	8.61	3.81	113337
									1200	72	11.1	4.94	77.15	34.29	10800
PHOTON	80	100	80	20	16	64	40	40	865	708	2.82	1.51	37.73	20.12	30621
									1168	132	15.15	8.08	111.13	59.27	7709
	128	144	128	16	16	112	64	64	1122	996	1.61	0.69	12.78	5.48	69845
									1708	156	10.26	4.40	35.15	15.06	16653
Spongent	80	88	80	8	8	80	40	40	738	990	0.81	0.42	14.84	7.74	91328
									1127	45	17.78	9.28	139.97	73.03	6339
	128	136	128	8	8	120	64	64	1060	2380	0.34	0.14	2.99	1.28	315350
									1687	70	11.43	14.90	40.16	17.21	14761
U-Quark	128	136	128	8	8	120	64	64	1379	544	1.47	0.61	7.73	3.20	93772
									2392	68	11.76	4.87	20.56	8.51	20332
H-PRESENT-128	128	-	128	64	-	128	64	64	2330	559	11.45	5.72	21.09	10.54	20351
									4256	32	200	100	110.41	55.21	2128
Keccak-f[100][+]	80	100	80	20	20	60	40	40	1250	800	2.50	1.50	16.00	9.60	50000
Keccak-f[200]	128	200	128	72	72	64	64	64	2520	900	8.00	3.56	12.6	5.60	31500
Keccak-f[400]	128	400	256	144	144	128	128	64	5090	1000	14.40	9.60	5.56	3.71	35347

[*]: Energy/bit = (Area[GE] $\times$ required cycles for one block process)/block size[bit]. [31]
[+]: Implementation data is estimated based on the same serialized architecture in [20].

This paper is organized as follows. Specification of LHash is given in Section 2. Section 3 describes the design rationale. Sections 4 and 5 provide results on security and implementation evaluations, respectively. Finally, we conclude in Section 6.

2 Specification of LHash

2.1 Notations

In the specification of LHash, we use the following notations:

– M:	The original message
– n:	The digest size
– b:	The block size of internal permutation
– F_{96}, F_{128}:	The 96(128)-bit internal permutation
– C_i:	The i-th round constant
– P_b:	Nibble permutation on $b/2$ bits state
– s:	4×4 S-box
– S:	Concatenation of four S-boxes
– T:	Non-linear function on 16-bit word
– G_b:	Concatenation of $b/32$ function T
– A:	4×4 MDS linear transformation on 16-bit word
– $\bigoplus$:	Bitwise exclusive-OR operation
– $\times 2, \times 4$:	Constant multiplications on finite field $\mathbb{F}_2[x]/x^4 + x + 1$

2.2 Domain Extender

In LHash, we choose the extended sponge function [3] as illustrated in Figure 1. The message is padded and split into r-bit blocks, each of which is XORed to part of the state and enter the permutation. After the message blocks are all processed, output r' bits of the state as a digest block and continue iterating the permutation until the output digest size is reached.

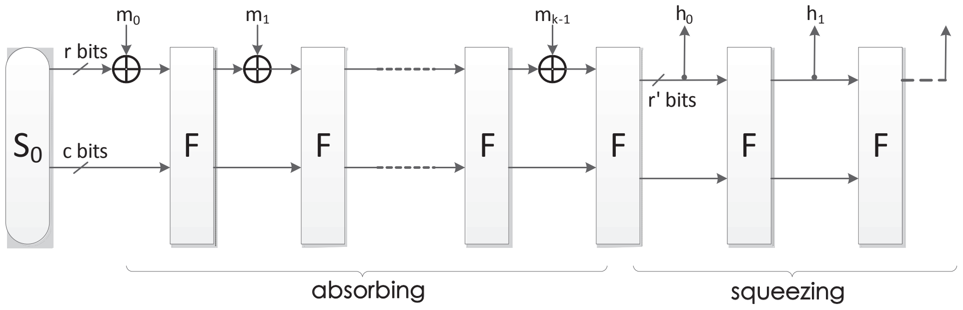

Fig. 1. Extended sponge function

In Figure 1, m_i is the i-th message block split from the padded message and h_i is the i-th digest block. F stands for a fixed internal permutation. r is the length of input message blocks and c is the size of the capacity. Then $b = r + c$ is the size of the fixed permutation and r' is the output size for each output digest block. S_0 is the initial value for the iteration. For different versions, the initial values are different. We set initial values as the concatenation of 8-bit binary expressions of the four parameters n, b, r and r' and fill zeros in the higher bits if the size is not enough, *i.e.* $S_0 = 0||...||0||n||b||r||r'$.

The padding works as follows. If the length of the original message is len, then the padding rule is to append one bit of "1" and x bits of "0", where x is the smallest non-negative integer such that $x + 1 + len \equiv 0 \; mod \; r$.

As shown in Table 2, four versions of LHash are constructed based on two permutations F_{96} and F_{128} with sizes of 96 and 128 bits. The parameters and security bounds can be found in Table 2. We refer to its various parameterizations as LHash-$n/b/r/r'$ for different digest sizes n, block sizes b, absorbing sizes r and squeezing sizes r'.

Table 2. Suggested parameters and security bounds of LHash

n	b	c	r	r'	collision	2nd preimage	preimage
80	96	80	16	16	2^{40}	2^{40}	2^{64}
96	96	80	16	16	2^{40}	2^{40}	2^{80}
128	128	112	16	32	2^{56}	2^{56}	2^{96}
128	128	120	8	8	2^{60}	2^{60}	2^{120}

2.3 Internal Permutation

The internal permutations F_{96} and F_{128} are constructed using an 18-round Feistel structure. The round transformations are shown in Figure 2. The permutation works as follows.

First split the b-bit input (b=96 or 128) into two halves $X_1||X_0$.

Then for $i = 2, 3, ..., 19$, calculate

$$X_i = G_b(P_b(X_{i-1} \oplus C_{i-1})) \oplus X_{i-2}$$

At last, $X_{19}||X_{18}$ is the output of the permutation.

Here the transformation G_b is the concatenation of $b/32$ function T which is the non-linear transformation on 16-bit word. The details of function T will be introduced in the following paragraphs. P_b is a simple permutation on $b/8$ nibbles, as defined in Table 3.

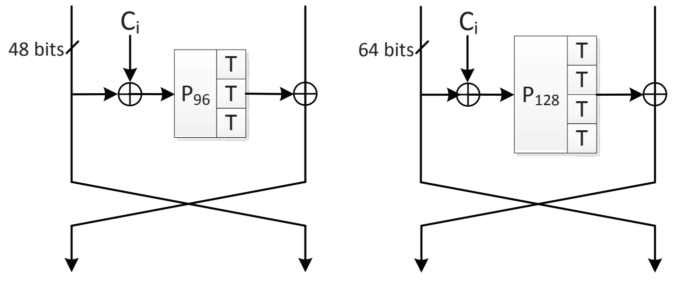

Fig. 2. Round transformations for internal permutation F_{96} and F_{128}

Table 3. Nibble permutation P_{96} and P_{128}

i	0	1	2	3	4	5	6	7	8	9	10	11
$P_{96}(i)$	6	0	9	11	1	4	10	3	5	7	2	8

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$P_{128}(i)$	3	6	9	12	7	10	13	0	11	14	1	4	15	8	5	2

Table 4. S-box used in LHash

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$s(x)$	14	9	15	0	13	4	10	11	1	2	8	3	7	6	12	5

The function T is defined as

$$T(x_3, x_2, x_1, x_0) = A(S(x_3, x_2, x_1, x_0))$$

where S is the concatenation of four S-boxes:

$$S(x_3, x_2, x_1, x_0) = s(x_3)||s(x_2)||s(x_1)||s(x_0).$$

The definition of the 4-bit S-box is shown in Table 4.

The linear layer A is an 4×4 MDS transformation on 16-bit word, and it is calculated as four iterations of the linear transformation B as shown below, *i.e.* $A = B^4$. In this figure $\times 2$ and $\times 4$ are constant multiplications on finite field $\mathbb{F}_2[x]/x^4 + x + 1$.

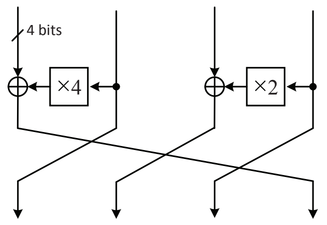

Fig. 3. Linear transformation B

The round constants C_i are used in both F_{96} and F_{128}. In each round, C_i is XORed to the most significant 16 bits of the state. The round constants are generated by a 5-bit LFSR. The initial state is zero: $x_4 = x_3 = x_2 = x_1 = x_0 = 0$, for $i > 4$, $x_i = x_{i-3} \oplus x_{i-5} \oplus 1$. Let $a_i = x_i||x_{i+1}||x_{i+2}||x_{i+3}$, $a'_i = x_i||x_{i+1}||x_{i+2}||\overline{x_{i+3}}$, $b_i = x_{i+1}||x_{i+2}||x_{i+3}||x_{i+4}$ and $b'_i = x_{i+1}||x_{i+2}||\overline{x_{i+3}}||x_{i+4}$. Then $C_i = a_i||b_i||a'_i||b'_i$. $\overline{x_{i+3}}$ stands for the complement of the bit x_{i+3}. The values of C_i are listed in Table 5.

3 Design Rational

3.1 Extended Sponge Function and the Choice of Paramenters

In most of the known RFID protocols, a hash function is required for privacy. In such cases, only the first preimage resistance is needed. Thus the second preimage bound can be sacrificed to have a more compact hardware implementation. LHash is based on extended sponge functions framework, which allows trade-offs among security, speed, energy consumption and implementation costs by adjusting parameters. Compared to the traditional construction based on block ciphers, the advantages are:

Table 5. Round constants

round	0	1	2	3	4	5	6	7	8
C_i	0012	0113	1301	3725	7E6C	ECFE	C9DB	9280	2436
round	9	10	11	12	13	14	15	16	17
C_i	485A	8193	1200	2537	5A48	A5B7	5B49	B7A5	7F6D

- Sponge function is based on fixed permutation. The cost of key expansion in block ciphers can be avoided. For encryption, the same secret key are used multiple times. As a result, the key expansion is done once and the subkeys can be stored and used again and again. But for block cipher based hash, we need to do the key expansion for every new message block.
- Avoiding feeding forward operation and saving storage. The cost of storage is high in lightweight computing environment. Thus for lightweight hash, we need to use as less storage as possible. In block cipher based hash, the feeding forward mode is necessary, thus the plaintext/key needs to be stored after the encryption is finished. If we need optimal resistance of second preimage attack, the capacity in sponge function needs to be of twice the size as the digest length and the storage requirement is similar to the block cipher based hash. But for sponge function, we can sacrifice the second preimage resistance by adjusting the parameters and reduce the storage requirement to about the same as the digest size.
- The security reduction of Sponge function has been proved. Thus we only need to analyze the security of the internal permutation. If the internal permutation doesn't have any flaw, we can have confidence in the security of the hash based on it in sponge mode.

3.2 Internal Permutation

Structure. The structure of the internal permutation is as shown in Figure 5 in Appendix II. P_n is a vector permutation whose unit size is the same as the size of sbox and G is the concatenation of several function $T's$. This structure is a kind of Feistel-type and combines the advantage of generalized Feistel structure. Its basic non-linear module T is small and parallelable, which make LHash can be implemented efficiently in both software and hardware.

Compared to traditional Feistel structure, the diffusion effect of generalized Feistel structure is slower and hence more rounds are needed to achieve the desired security level. In order to overcome the disadvantage, we propose an extended variant of improved generalized Feistel structure which represented as Feistel-PG. We utilize a permutation layer P_n on nibbles to improve the diffusion effect. The unit size of P_n is the same as the size of the S-box and requires no extra hardware area cost. However, the choice of P_n has impact on the security. After a lot of attempts and tests, we decide to choose current permutations since they are the best ones we found. Assuming the same block length and size of non-linear module T, compared to traditional generalized Feistel structure such as Type-2 GFS and improved GFS [32], Feistel-PG can achieve full diffusion [32] in less rounds which means its diffusion speed is faster, and the attackable round number of impossible differential and integral path is fewer. The comparison between different structures are listed in Table 6.

Moreover, compared to the usually utilized SP structure in the design of internal permutation of hash functions, Feistel-type structure has completely different proper-

Table 6. Comparison of structures

Structure	Size	Full Diffusion	Impossible Differential	Integral Path
Type-2 generalized Feistel	96	7	13	12
Improved GFS	96	5	9	10
Feistel-PG	96	4	8	8
Type-2 generalized Feistel	128	9	17	16
Improved GFS	128	6	10	11
Feistel-PG	128	4	8	8

ties. Therefore, most of the hash function attack techniques suitable for the property of SP-type structure, such as the most famous rebound attack, super-sbox techniques etc, will be less effective. Therefore, we believe that Feistel-type internal permutation can achieve good immunity against known attacks, and later in Section 4 we will evaluate the security of LHash against known attacks in detail.

S-box. On the pursuit of hardware efficiency, we use 4×4 S-boxes $s : F_2^4 \rightarrow F_2^4$ in LHash. Compared with the regular 8×8 S-box, small S-box has much more advantage in hardware implementation. For example, to implement the S-box of AES in hardware more than 200 GE are needed. On the other hand, the S-box used in LHash requires two AND operations, two ORs, one NOT and six XORs. The area costs for AND, OR, NOT and XOR are 1.33 GE, 1.33 GE, 0.5 GE and 2.67 GE. Thus the S-box costs 21.84 GE in total. Furthermore, in the aspect of security, the S-box used in LHash is complete, has no fixed points, optimal differential and linear characteristics probability of 2^{-2} and algebraic degree of 3.

Linear Diffusion Layer. The diffusion of the internal permutation is achieved by both the nibble permutation P_n and the linear layer A used in function T. Their combination results in good security. The lower bound for the active S-boxes for 17 out of 18 rounds of F_{96} and F_{128} are 48 and 64, respectively. P_n is the nibble permutation with no hardware cost. A is a linear transformation on 16 bits, the branch number of A regarding 4 nibbles is 5, which is optimal.

A follows the 4-branch generalized Feistel structure, the round function uses constant multiplications $\times 2$ and $\times 4$ on $\mathbb{F}_2[x]/x^4 + x + 1$. After four iterations, the branch number of 5 can be reached. The linear transformation A has two advantages:

- Easy to invert. The reason why we consider the inversion is for compact hardware implementation, which will be explained in the following sections. Since generalized Feistel structure is used to implement A, we only need to change the permutation of the round function to invert it and the round function can be reused.
- Low area cost for hardware implementation. In the generalized Feistel structure, there are two XOR operations between 4-bit branches, which requires 8 bits of XORs. The multiplication by 2 and 4 can be implemented using 1 and 2 bits of XORs. The total area cost of A is 11 bits of XORs and is less than the iterated MDS layer used in PHOTON, which requires 15 bits of XORs.

The circuits for multiplications by 2 and 4 in $\mathbb{F}_2[x]/x^4 + x + 1$ can be found in Figure 6.

4 Security Evaluation

Since extended sponge function is used, the desired security of the internal permutation is that the permutation is indistinguishable with a random permutation with no more than $2^{c/2}$ queries. The parameter r is small for all versions of LHash. Thus for the adversary, the controllable freedom degree is small (no more than 16 bits for each permutation). It is difficult for the adversaries to take advantage of the vulnerability of the internal permutation and turn it into an attack on the hash function. Based on the reasoning above, we only propose analysis of the internal permutation. All analyses in this section are based on the stronger assumption that the input can be completely controlled by the adversary.

4.1 Generic Security Bounds

With the assumption that the internal permutation is ideal, the security bounds for extended sponge function are as follows [14].

- Collision bound: $min\{2^{n/2}, 2^{c/2}\}$.
- Second preimage bound: $min\{2^{n}, 2^{c/2}\}$.
- Preimage bound: $min\{2^{min\{n,b\}}, max\{2^{min\{n,b\}-r'}, 2^{c/2}\}\}$.

According to our analysis on the internal permutations F_{96} and F_{128}, we believe the bounds shown in Table 2 can be reached.

4.2 Differential Analysis

For differential analysis of LHash, it is highly dependent on the maximum differential characteristic probability of the internal permutations F_{96} and F_{128}. Considering the internal permutations are built based on block cipher structure, we can adopt the regular method of searching least number of active sboxes to evaluate the upper bound of differential characteristic probability for F_{96} and F_{128}. This method is widely used in security evaluation of Feistel cipher against differential analysis such as Camellia and CLEFIA.

The search program is usually a truncated differential path search with Viterbi algorithm. Considering that the sbox is a bijective and deterministic nonlinear function, its input and output differences can be truncated to 1-bit. Namely if its input and output differences are not zero, then we call it an active sbox and denote it as "1" in the truncated differential path. Otherwise, if the input and output differences are both zero, then we call it a passive sbox and denote it as "0". Notice that for a passive sbox its differential probability $DP_S = 1$, and for an active sbox $DP_S = p < 1$. Therefore, by counting the minimum number of active sboxes, we can evaluate the upper bound of differential characteristic probability. In the truncated differential path search, we start from input state $\Delta D^{(0)}$, and transit toward output state $\Delta D^{(r)}$ round by round so as to minimize the number of active sboxes at every round as follows.

For every possible truncated differential path $(\Delta D^{(i)} \rightarrow \Delta D^{(i+1)})$, assign the right-hand value to left-hand if the inequation is satisfied where $z(\Delta D^{(i+1)})$ is a temporal variable of the minimum active sbox number.

$$z(\Delta D^{(i+1)}) > AS_{min}(\Delta D^{(i)}) + AS(\Delta D^{(i)} \rightarrow \Delta D^{(i+1)})$$

Table 7. The guaranteed minimum number of active sboxes for F_{96} and F_{128}

round	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
F_{96}	1	2	6	10	14	18	22	24	26	29	32	36	40	42	46	48	51	54	58
F_{128}	1	2	6	10	16	22	27	31	35	37	41	45	51	54	61	64	67	71	75

where $AS_{min}(\varDelta D^{(i)})$ is the total minimum active sbox number of the truncated path from the first round to $\varDelta D^{(i)}$, and $AS(\varDelta D^{(i)} \to \varDelta D^{(i+1)})$ denotes the active sbox number of truncated path $(\varDelta D^{(i)} \to \varDelta D^{(i+1)})$. Then the temporal variable $z(\varDelta D^{(i+1)})$ after finishing the above steps becomes $AS_{min}(\varDelta D^{(i+1)})$. Finally, the minimum result of $AS_{min}(\varDelta D^{(r)})$ is the guaranteed minimum number of active sboxes for r-round.

After searching the guaranteed minimum number of active sboxes for internal permutations F_{96} and F_{128} by computer program, the results are listed in Table 7. Since the maximal probability for differential distribution of the sbox is 2^{-2}, 17 rounds of F_{96} and F_{128} cannot be distinguished from a random permutation by using differential paths. Thus we believe LHash is secure regarding differential attack.

4.3 Rebound Attack

Rebound attack was proposed in 2009 [26], which was very effective against AES-like structures. Till now, lots of works have been done to improve it [27, 19, 23–25, 34, 30, 28]. The original rebound attack works on AES structure itself, which cannot be directly applied to LHash. Sasaki tried to analyze the resistance of Feistel-SP structure against rebound attack [29]. Here we propose the preliminary rebound attack on LHash.

11-round rebound distinguishers on the internal permutations. In Figure 8, 5-round inbound paths for both F_{96} and F_{128} are given. Then the full 11-round path can be obtained by extending the inbound rounds 3 rounds backward and 3 rounds forward as shown in Figure 10. The attack steps are similar for F_{96} and F_{128}. Therefore we only choose the attack on F_{128} for the demonstration in Figure 9. The attack steps are:

Step 1. Determine the difference of gray nibbles. Noticed that the differences in gray nibbles are related. Once we fix the difference of the gray nibbles after the S-boxes in the first round, the differences of all the remaining gray nibbles are fixed.

Step 2. The first inbound. Choose random difference for the blue nibbles such that they match at the S-boxes in the first and the second rounds. We can determine the value of the blue lines from the matches. The probability for the match is 2^{-16}. We expect to find 2^{16} solutions with 2^{16} complexity.

Step 3. The second inbound. The process is similar with the first inbound. Choose random difference for the yellow nibbles and match at the S-boxes in round 4 and round 5. 2^{16} solutions are expected with 2^{16} complexity.

Step 4. The outbound. XOR the values of the blue and the yellow lines at the third round to get the value of the red lines. Calculate forward and backward following the red lines. The conditions introduced by the active S-boxes can be fulfilled with probability of 2^{-16}.

We only need to do the inbound steps once since we can get 2^{32} solutions for the red lines and only 2^{16} of them are needed. The total complexity to find a solution to the 11-round path is 2^{16}. For a random permutation, finding a solution fulfilling the input and output truncated differences is a limited birthday problem [13]. Though the input and output are both full active, the difference of one half is generated by 16-bit difference and can be regarded as having 12 non-active nibbles. On the other hand, the complexity to solve this limited birthday problem for a random permutation is $max\{2^{48/2}, 2^{48/2}, 2^{48+48-128}\} = 2^{24}$, which is higher than the complexity using our differential path. That is how this distinguisher works. For F_{96}, it takes 2^{12} complexity to find one solution, which is lower than the generic case (2^{18}).

Remarks on Super-Sbox technique. Super-Sbox technique [13] exploits the independency between columns in 2 rounds of AES-like structure and improves the attackable rounds of rebound attack. In Feistel-PG structure, there is no independent structure like this. Therefore, we believe super-sbox technique doesn't work on LHash.

4.4 Zero-sum Distinguisher

For a given permutation F, zero-sum distinguisher aims to find a partition of the input values X such that $\bigoplus_{x \in X} x = \bigoplus_{x \in X} F(x) = 0$ with low complexity. Here we consider another kind of distinguisher called half zero-sum distinguisher. Suppose permutation F is $2n$ bits, we aim to find set X such that $\bigoplus_{x \in X} Trunc_n^1(x) = \bigoplus_{x \in X} Trunc_n^2(F(x)) = 0$, where $Trunc_n^1$ and $Trunc_n^2$ are truncation functions with half of the state size.

We have measured the algebraic degree of F_{96} and F_{128}, using the technique proposed by Boura *et al.* [8–10]. For F_{96}, half of the state doesn't reach maximal algebraic degree of 95 after 7 rounds. Then we can propose a 15-round half zero-sum distinguisher for F_{96}.

Choose 20 nibbles (except the first 4 nibbles of left branch) to be active, and then we can obtain 8 independent active nibbles after 3 rounds. When we choose one bit from the other part of the state and fix it, the 8 nibbles will go over all the 2^{32} values no matter which bit value we choose besides the 8 nibbles. Then we can deduce that the algebraic degree of half of the state after another 5 rounds is no more than 27 and the sum of these bits will be zero. Therefore, in the forward direction, we have an 8-round half zero-sum distinguisher. Similarly, we can deduce that it is a 7-round half zero-sum distinguisher in the backward direction when we select the same active nibbles. Combining the forward and backward paths, we get a 15-round half zero-sum distinguisher for F_{96}, and the data complexity is 2^{80}. Apply the same technique on F_{128}, we can find a 15-round half zero-sum distinguisher with 2^{96}, which can be improved using the algebraic bounding techniques. In the forward direction, we choose all the nibbles on the left side(64 bits) and 63 bits on the right side as active bits. In the forward direction, we can ensure that after 9 rounds, half of the state is balanced(with zero-sum). In the backward direction, we can only go back for 8 rounds. As a result, we have a 17-round half zero-sum distinguisher on F_{128} with 2^{127} data complexity.

4.5 Slide Attacks

Slide attacks are proposed for block ciphers, which take advantage of the self-similarity in the key expansion by constructing plaintext-ciphertext pairs fulfilling the slide conditions and recover the internal state or the secret key. Since hash function can be used to construct MACs, *e.g.* HMAC and NMAC, and sponge function itself can be used to construct MACs, we need to consider this type of attacks.

We have two different types of slide attacks: sliding on round transformations inside the internal permutation and sliding on iterations of the internal permutations, *i.e.* sliding message blocks. Firstly, slide inside the internal permutation is prevented by adding different round constants in each round. Secondly, slide attack between iterations of the internal permutations can be prevented if our padding rule ensures the last message block to be always non-zero. The padding rule of LHash fulfills this property. Thus we can conclude that slide attack doesn't work on LHash.

4.6 Other Attacks

Rotational distinguisher. Rotational distinguisher [21, 22] was proposed to analyze ARX structures. Calculate the output of a rotational pair and check if the rotational condition is still fulfilled. In the design based on S-boxes and MDS linear layer, rotational distinguisher doesn't work well. The using of S-boxes ensure that if the rotational amount is not multiple of the size of the S-box, the rotational relation will be destroyed. The only possible way to apply rotational distinguisher on LHash, is to use a rotational amount as multiple of 4. Furthermore, the rotational pair will be destroyed by the nibble permutation layers. Based on this reason, we conclude that LHash is immune to rotational distinguisher.

Self differential attack. In a self differential attack, the difference between different partitions of a single value is considered, instead of the difference between a pair of values. The best collision attack on Keccak is based on this kind of attack [12]. The self similarity property can be found in AES, if there is no constants, the similarity can be preserved forever. In LHash, the nibble permutation can destroy the self similarity and ensures LHash immune to this kind of attack.

5 Implementation

5.1 Hardware Implementation

We evaluate hardware implementation of LHash using the *Virtual Silicon* (VST) standard cell library based on *UMC L180 0.18μm 1P6M logic process* (*UMCL18G212T3*). We propose two different hardware implementations: 1) minimal area (serialized) implementation and 2) implementation based on function T. In the second implementation, the area cost is higher while the speed is significantly increased and the energy consumption is significantly reduced.

Serialized implementation. In hardware implementation of AES-like structures, such as PHOTON, the value before S-boxes don't need to be stored. The output values of the S-boxes can be stored in the same storage units of the inputs, i.e. the input values are overwritten. During the calculation of the iterated MDS layer, the intermediate values are also stored in the same place. For Feistel structure, the situation is different. Since the value of the left branch needs to be kept for the next round, we cannot just discard the original values before S-boxes. If the round function cannot be calculated in one cycle, we need extra storage to store the intermediate values for the following calculations, which is bad for compact implementations. In order to achieve compact serialized implementation, we introduce an equivalent expression of the round transformation. Fig. 4 shows the equivalent round transformation with one T module. It can be expressed as follows, and the equation ensures that we don't need to store the intermediate output value of sbox during the calculation.

$$A(S(P_n(X_{i-1} \oplus C_i))) \oplus X_{i-2} = A(S(P_n(X_{i-1} \oplus C_i)) \oplus A^{-1}(X_{i-2}))$$

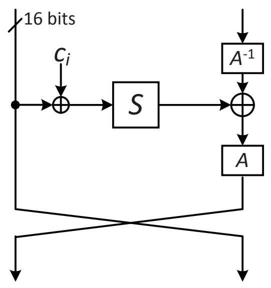

Fig. 4. Equivalent round transformation

First, we applied the inversion of the linear layer, *i.e.* A^{-1}, to the right branch of the state. The updated value is stored at the original storage unit. Then we calculate the constant addition, nibble permutation, and sbox operation of the left branch nibble by nibble and XOR the output of the sbox into the right branch storage. After all sboxes have been processed, the linear transformation A is applied to the right branch again and we get the value of X_i. During the calculation, no extra storage is required. After finishing all the nonlinear T modules in one round, another operation is needed to swap the left and right branches.

In the serialized implementation of LHash, we use a 4-bit width datapath and only one instance of sbox. A^{-1} and A need to be implemented respectively. First of all, state storage needs 96(128) bits flip-flop cells to store the data, and each bit flip-flop requires 6 GE. Therefore, for F_{96} and F_{128} this module requires $96 \times 6 = 576$ GE and $128 \times 6 = 768$ GE respectively, which occupies the majority of the total area required. Then for the round transformation, six kinds of operations need to be done, including constant addition, nibble permutation, sbox, 4-bit XOR operation, linear transformation A^{-1} and A. Notice that in the design of LHash, the constants only apply to the first 16-bit of left branch and for the other bits the constants equal to zero. Therefore, we only need four 4-bit XOR to implement the constant addition operation which requires about 42.72 GE. Moreover, another 32.75 GE is needed for the constant generator(5-bit storage, one XNOR and one NOT). Then the nibble permutation can be implemented by simple wiring and costs no area. The choice of data is controlled by the Controller module where a Finite State Machine is used to generate the control signals. As specified in Sect. 3.2, the modules of sbox, A^{-1} and A require 21.84 GE, 29.37 GE(11 bits XOR), and 29.37 GE respectively. Finally, to XOR the output of sbox into the right branch nibble, a 4-bit XOR is needed which costs 10.68 GE. In the end of the round transformation, the swap operation can be implemented by wiring and need no additional area. Furthermore, an overall Controller module is needed to generate

Table 8. Software performances in cycles per byte of the LHash variants

LHash-80/96/16/16	LHash-96/96/16/16	LHash-128/128/16/32	LHash-128/128/8/8
139c/B	139c/B	156c/B	312c/B

all the control signals and logic circuits. The Controller module is realized by a Finite State Machine and its gate varies depending on the size of internal permutation: about 74 GE for F_{96} and 93 GE for F_{128}.

In summary, for F_{96}, the round transformation contains three function T's. Each of them requires 4 cycles for A^{-1}, 4 cycles for the combination operation of constant addition, S-box and 4-bit XOR, and another 4 cycles for A. After the calculation of three function T, another 1 cycle is required to swap the left and right branches. Thus the round transformation requires $12\times3+1=37$ cycles and F_{96} requires $37\times18=666$ cycles in total. The area cost of F_{96} is about 817 GE, including 576 GE for 96-bit storage, 53.4 GE for five 4-bit XORs, 22 GE for the sbox, 29.37×2 GE for both A^{-1} and A, 32.75 GE for the constant generator(5-bit storage, one XNOR and one NOT) and about 74 GE for logic circuits. For F_{128}, the round transformation contains four T modules. It takes $(12\times4+1)\times18=882$ cycles to finish the calculation. The area cost of the serialized implementation of F_{128} is 1028 GE, including 768 GE register storage, 166.89 GE for the round transformation, and about 93 GE for logic circuits.

Function *T* based implementation. Since F_{96} and F_{128} share the same module T, thus we only need to implement it once. In order to finish the calculation of function T in one cycle, we need eight 4-bit registers, constant generator and 16-bit XOR. The function T requires about 515.17 GE, including 192 GE for eight 4-bit storage, 32.75 GE for the constant generator, 88 GE for four S-boxes, $29.37\times4=117.84$ GE for A and $2.67\times32=85.44$ GE for 32-bit XOR. Both F_{96} and F_{128} can share the same function T and we only need extra storage for both of them. Therefore F_{96} requires $515.17+384=899.17$ GE, with additional 89.8 GE control logic circuits, and the total area cost is about 989 GE. It takes $3\times18=54$ cycles to finish the calculation of F_{96}. Similarly, F_{128} costs about $(515.17+576+108.8)\approx1200$ GE and 72 cycles.

5.2 Software Implementation

We give in Table 8 our software implementation performances for the LHash variants. The processor used for the benchmarks is an Intel Core i7-3612QM @2.10GHz. We have also benchmarked other lightweight hash function designs. QUARK reference code [1], which is very likely to be optimized, runs at 8k, 30k and 22k cycles per byte respectively for U-QUARK, D-QUARK and S-QUARK. The speed for PHOTON-80/20/16 and PHOTON-128/16/16 [14] are 96 and 156 cycles per byte, respectively.

6 Conclusion

We proposed a new lightweight hash function LHash, supporting digest length of 80, 96 and 128 bits, providing from 64 to 120 bits of preimage security and from 40 to 60 bits of second preimage and collision security. The internal permutation is designed based on structure Feistel-PG, using nibble permutations to improve the resistance to different attacks of the structure. The component S-box and linear layer are designed to be secure and suitable for hardware implementations. Serialized implementation of the internal permutation in LHash requires 817 and 1028 GE. LHash has the lowest energy consumption among existing lightweight hash functions. We offer the trade-offs among security, speed, energy consumption and implementation cost by adjusting the parameters. We also evaluate the security of LHash and our cryptanalytic results show that LHash achieves enough security margin against known attacks. In the end, we strongly encourage the security analysis of LHash and helpful comments.

Acknowledgement. We wish to thank Florian Mendel for his kind communication and many useful suggestions, and also the anonymous reviewers for their helpful comments. This work is partly supported by the National Basic Research Program of China and the National Natural Science Foundation of China.

References

1. Aumasson, J.-P., Henzen, L., Meier, W., Plasencia, M.N.: Quark: A Lightweight Hash. *J. Cryptology*, 26(2):313-339 (2013)
2. Badel, S., Dagtekin, N., Nakahara, J., Ouafi, K., Reffé, N., Sepehrdad, P., Susil, P., Vaudenay, S.: ARMADILLO: A Multi-purpose Cryptographic Primitive Dedicated to Hardware. In Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 398-412. Springer, (2010)
3. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: Sponge Functions. Ecrypt Hash Worksop, (2007)
4. Bogdanov, A., Knezevic, M., Leander, G., Toz, D., Varici, K., Verbauwhede I.: spongent: A Lightweight Hash Function. In Preneel, Takagi (eds.) CHES 2011. LNCS, vol. 6917, pp. 312-325. Springer, (2011)
5. Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y., Vikkelsoe, C.: PRESENT: An Ultra-Lightweight Block Cipher. In Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 450-466. Sringer, (2007)
6. Bogdanov, A., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y.: Hash Functions and RFID Tags: Mind the Gap. In Oswald E., Rohatgi, P., (eds.) CHES 2008. LNCS, vol. 5154, pp. 283-299. Sringer, (2008)
7. Borghoff, J., Canteaut, A., Güneysu, T., Kavun, E.B., Knezevic, M., Knudsen, L.R., Leander, G., Nikov, V., Paar, C., Rechberger, C., Rombouts, P., Thomsen, S.S., Yalçin T.: Prince - A Low-latency Block Cipher for Pervasive Computing Applications (full version). `http://eprint.iacr.org/2012/529.pdf`, (2012)
8. Boura, C., Canteaut, A.: Zero-sum Distinguishers for Iterated Permutations and Application to Keccak-*f* and Hamsi-256. In Biryukov, A., Gong, G., Stinson, D.R. (eds.) SAC 2010. LNCS, vol. 6544, pp. 1-17. Sringer, (2010)
9. Boura, C., Canteaut, A.: On the Influence of the Algebraic Degree of f^{-1} on the Algebraic Degree of gf. *IEEE Transactions on Information Theory*, 59(1):691-702, (2013)
10. Boura, C., Canteaut, A., Cannière, C.D.: Higher-order Differential Properties of Keccak and *Luffa*. In Joux, A., (ed.) FSE 2011. LNCS, vol. 6733, pp. 252-269. Sringer, (2011)
11. Cannière, C.D., Preneel, B.: Trivium. In Robshaw, Billet (eds.) New Stream Cipher Designs - The eSTREAM Finalists. LNCS, vol. 4986, pp. 244-266. Springer, (2008)
12. Dinur, I., Dunkelman, O., Shamir, A.: Self-differential Cryptanalysis of Up to 5 Rounds of SHA-3. `http://eprint.iacr.org/2012/672.pdf`, (2012)
13. Gilbert H., Peyrin, T.: Super-sbox Cryptanalysis: Improved Attacks for AES-like Permutations. In Hong, Iwata (eds.) FSE 2010. LNCS, vol. 6147, pp. 365-383. Springer, (2010)
14. Guo, J., Peyrin, T., Poschmann, A.: The PHOTON Family of Lightweight Hash Functions. In Rogaway (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 222-239. Springer, (2011)
15. Guo, J., Peyrin, T., Poschmann, A., Robshaw, M.J.B.: The LED Block Cipher. In Preneel, Takagi (eds.) CHES 2011. LNCS, vol. 6917, pp. 326-341. Springer, (2011)
16. Hell, M., Johansson, T., Maximov, A., Meier, W.: The Grain Family of Stream Ciphers. In Robshaw, Billet (eds.) New Stream Cipher Designs - The eSTREAM Finalists. LNCS, vol. 4986, pp. 179-190. Springer, (2008)

17. Hirose, S.: Some Plausible Constructions of Double-Block-Length Hash Functions. In Robshaw, M. J. B. (ed.) FSE 2006. LNCS, vol. 4047, pp. 210-225. Springer, (2006)
18. Hong, D., Sung, J., Hong, S., Lim, J., Lee, S., Koo, B., Lee, C., Chang, D., Lee, J., Jeong, K., Kim, H., Kim, J., Chee, S.: HIGHT: A New Block Cipher Suitable for Low-Resource Device. In Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, pp. 46-59. Springer, (2006)
19. Jean, J., Naya-Plasencia, M., Peyrin, T.: Improved Rebound Attack on the Finalist Grøstl. In Canteaut, A. (ed.) FSE 2012. LNCS, vol. 7649, pp. 110-126. Springer, (2012)
20. Kavun, E.B., Yalcin, T.: A Lightweight Implementation of Keccak Hash Function for Radio-Frequency Identification Applications. In S.B. Ors Yalcin (ed.) RFIDSec 2010. LNCS, vol. 6370, pp. 258-269. Springer, (2010)
21. Khovratovich, D., Nikolic, I.: Rotational Cryptanalysis of ARX. In Hong, Iwata (eds.) FSE 2010. LNCS, vol. 6147, pp. 333-346. Springer, (2010)
22. Khovratovich, D., Nikolic, I., Rechberger, C.: Rotational Rebound Attacks on Reduced Skein. In Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 1-19. Springer, (2010)
23. Lamberger, M., Mendel, F., Rechberger, C., Rijmen, V., Schläffer, M.: Rebound Distinguishers: Results on the Full Whirlpool Compression Function. In Matsui (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 126-143. Springer, (2009)
24. Lamberger, M., Mendel, F., Rechberger, C., Rijmen, V., Schläffer, M.: The Rebound Attack and Subspace Distinguishers: Application to Whirlpool. `http://eprint.iacr.org/2010/198.pdf`. Accepted for publication in *J. Cryptology*.
25. Matusiewicz, K., Naya-Plasencia, M., Nikolic, I., Sasaki, Y., Schläffer, M.: Rebound Attack on the Full Lane Compression Function. In Matsui (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 106-125. Springer, (2009)
26. Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: The Rebound Attack: Cryptanalysis of Reduced Whirlpool and Grøstl. In Dunkelman, O. (ed.) FSE 2009. LNCS, vol. 5665, pp. 260-276. Springer, (2009)
27. Naya-Plasencia, M.: How to Improve Rebound Attacks. In Rogaway (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 188-205. Springer, (2011)
28. Rijmen, V., Toz, D., Varici, K.: Rebound Attack on Reduced-round Versions of JH. In Hong, Iwata (eds.) FSE 2010. LNCS, vol. 6147, pp. 286-303. Springer, (2010)
29. Sasaki, Y.: Double-sp is Weaker than Single-sp: Rebound Attacks on Feistel Ciphers with Several Rounds. In Galbraith, S.D., Nandi, M. (ed.) INDOCRYPT 2012. LNCS, vol. 7668, pp. 265-282. Springer, (2012)
30. Sasaki, Y., Wang, L., Wu, S., Wu, W.: Investigating Fundamental Security Requirements on Whirlpool: Improved Preimage and Collision Attacks. In Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 562-579. Springer, (2012)
31. Shibutani, K., Isobe, T., Hiwatari, H., Mitsuda, A., Akishita, T., Shirai, T.: Piccolo: An Ultra-Lightweight Blockcipher. In Preneel, Takagi (eds.) CHES 2011. LNCS, vol. 6917, pp. 342-357. Springer, (2011)
32. Suzaki, T., Minematsu, K.: Improving the Generalized Feistel. In Hong, Iwata (eds.) FSE 2010. LNCS, vol. 6147, pp. 19-39. Springer, (2010)
33. Susil, P., Vaudenay, S.: Multipurpose Cryptographic Primitive ARMADILLO3. In Mangard, S. (ed.) CARDIS 2012. LNCS, vol. 7771, pp. 203-218. Springer, (2013)
34. Wu, S., Feng, D., Wu, W.: Practical Rebound Attack on 12-Round Cheetah-256. In Lee, D., Hong, S. (eds.) ICISC 2009. LNCS, vol. 5984, pp. 300-314. Springer, (2009)
35. Wu, W., Zhang, L.: LBlock: A Lightweight Block Cipher. In Lopez, J., Tsudik, G. (eds.) ACNS 2011. LNCS, vol. 6715, pp. 327-344. Springer, (2011)

Appendix I: Test Vectors

Test vectors for LHash are shown in hexadecimal notation as follows.

Appendix II: Figures

Message	FF FE FD FC FB FA F9 F8 F7 F6 F5 F4 F3 F2 F1 F0 EF EE ED EC EB EA E9 E8 E7 E6 E5 E4 E3 E2 E1 E0
LHash-80/96/16/16	4A BD BA E1 44 7F C8 E4 5B 58
LHash-96/96/16/16	55 EC 4F FE 99 2A 32 94 F1 F7 90 61
LHash-128/128/16/32	38 E9 1A E1 8F 11 5A 0B 27 79 68 22 A9 0B 1C 5A
LHash-128/128/8/8	0A D6 35 B4 8F E3 BD 84 F9 58 7C 68 B0 CA DA E0

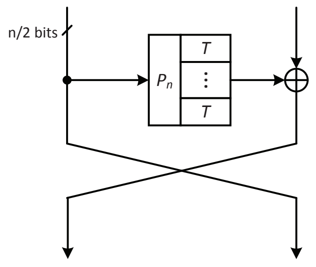

Fig. 5. Feistel-PG structure

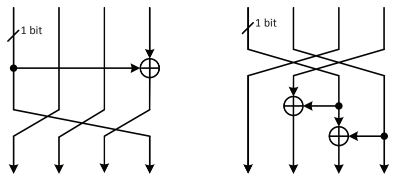

Fig. 6. Circuits for multiplications by 2 and 4 on $\mathbb{F}_2[x]/x^4 + x + 1$

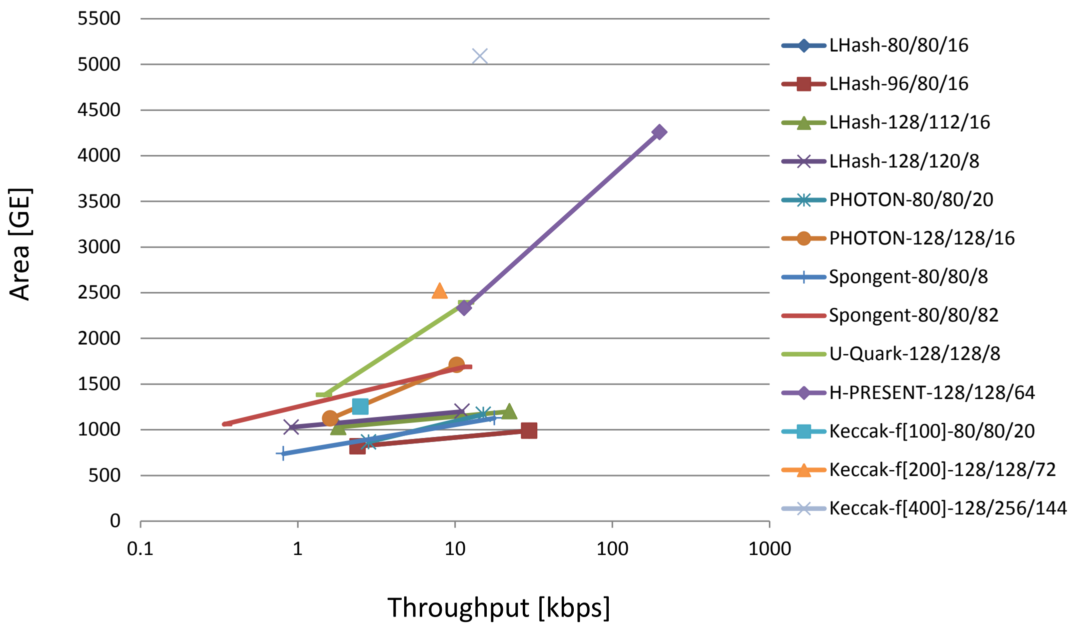

Fig. 7. Area versus throughput trade-off of lightweight hash functions

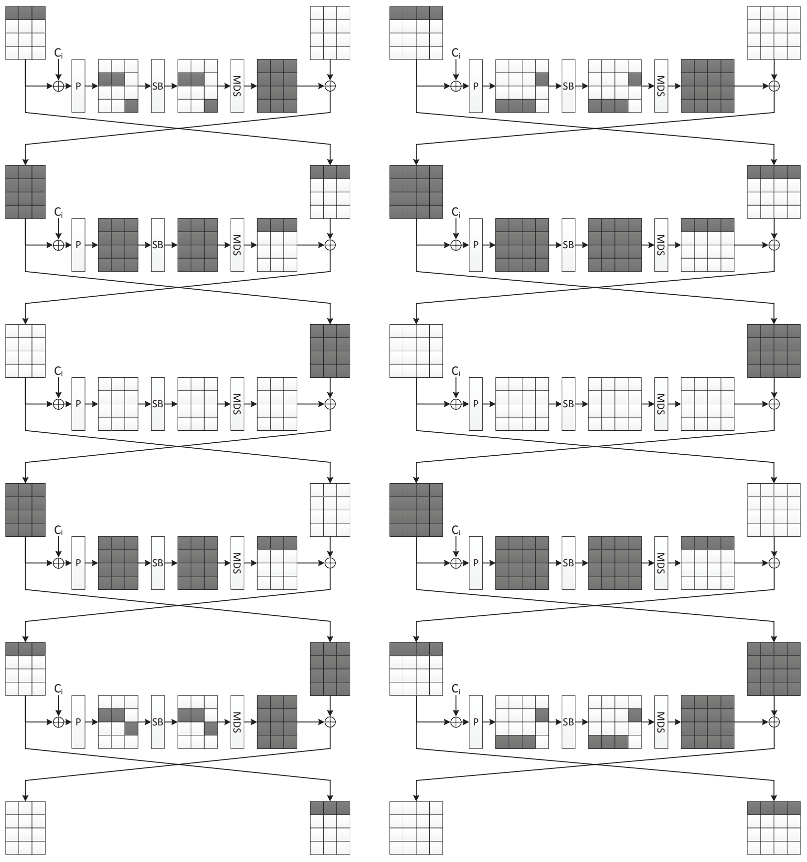

Fig. 8. 5-round inbound paths for F_{96} and F_{128}

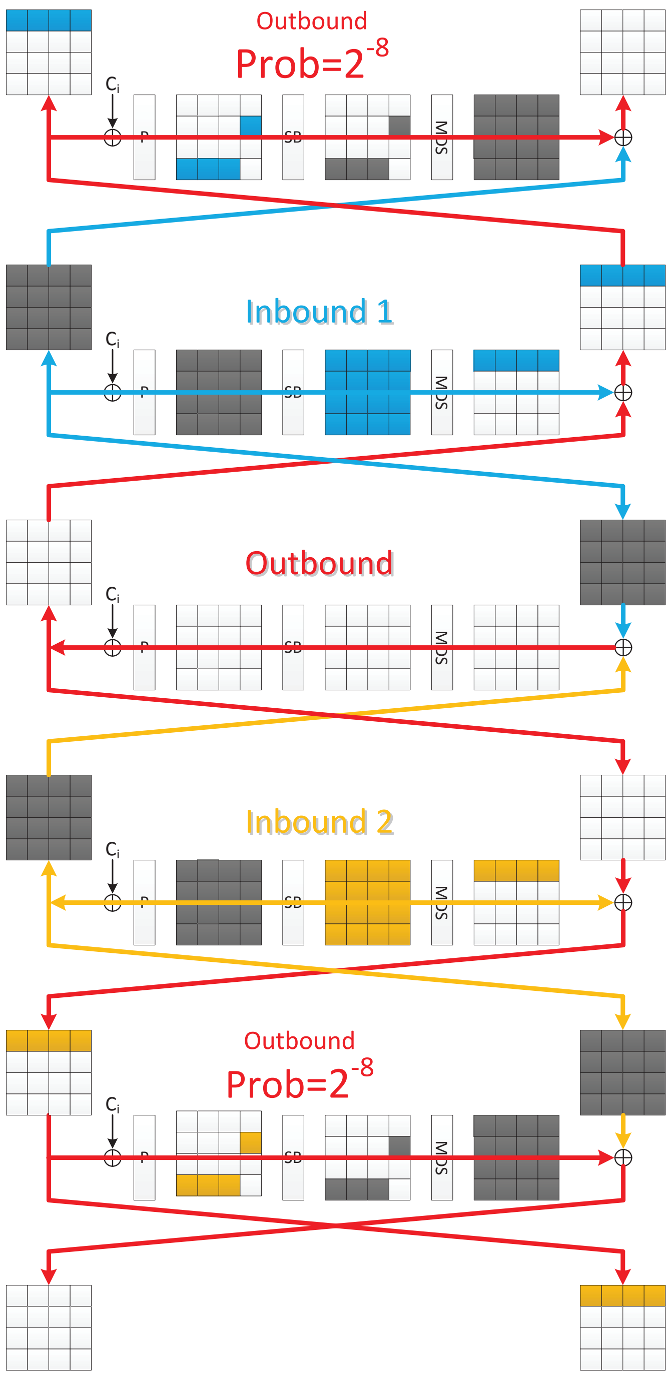

Fig. 9. 5-round inbound path for F_{128}

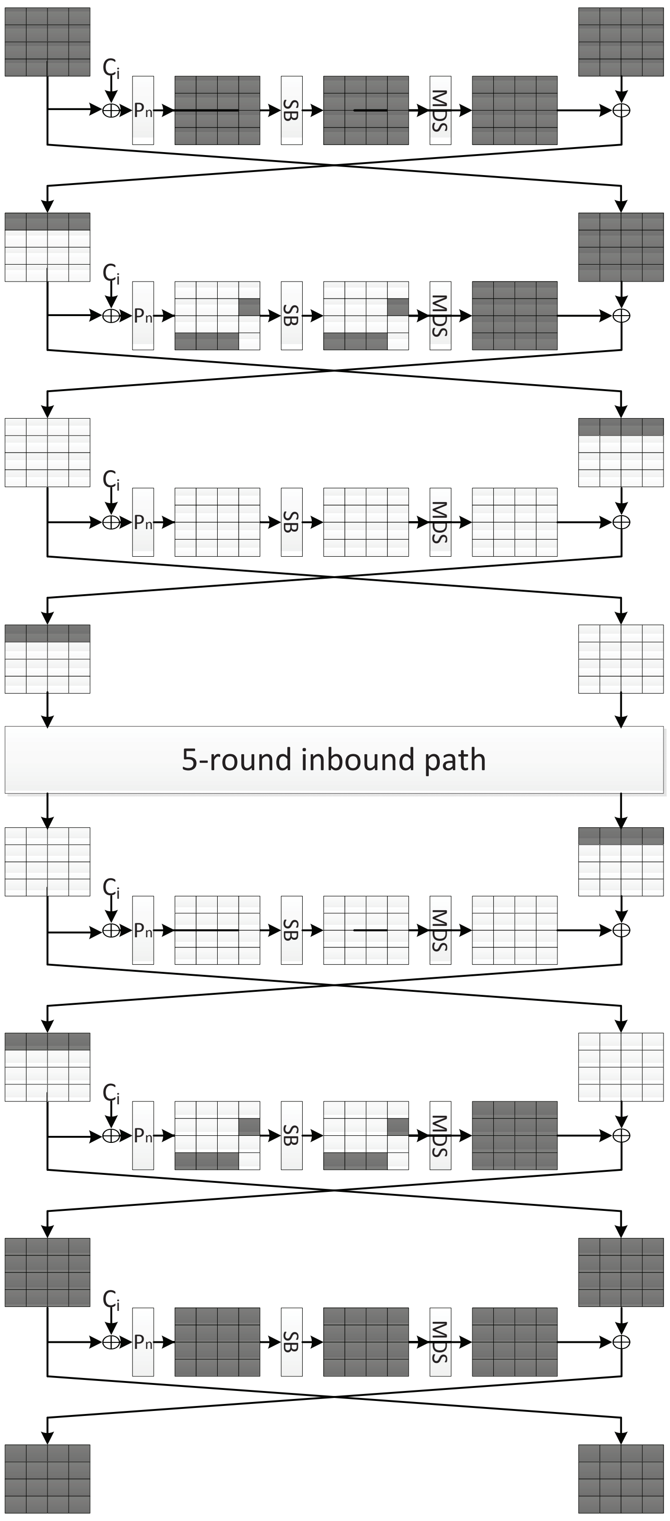

Fig. 10. Rebound attack for 11-round F_{128}