HERN & HERON:

Lightweight AEAD and Hash Constructions based on Thin Sponge (v1)

Dingfeng Ye[1,2], Danping Shi[1,2], Yuan Ma[1,2], Peng Wang[1,2]

[1] Data Assurance and Communication Security Research Center,
Chinese Academy of Sciences

[2] State Key Laboratory of Information Security, Institute of Information Engineering,
Chinese Academy of Sciences

wp@is.ac.cn

March 29, 2019

Contents

Chapter 1

Introduction

Authentication encryption with associated data (AEAD) and hash function are widely used in modern information systems. Recently various competitions such as SHA-3, PHC and CAESAR are held to satisfy a wide range of application requirements of these cryptographic algorithms. On the other hand, due to the rapid development of lightweight devices, requirements in different resource-constrained scenarios are proposed, such as in IoT, embedded system, smart card and wireless sensor node. If these algorithms are implemented by different circuits, it will cause a waste of hardware resources. *We hope to design both AEAD and hash function which can be implemented in a single small circuit together.*

The most popular way to construct both AEAD and hash function is to use the sponge function which is a mode of operation with variable-length input and arbitrary-length output based on fixed-length permutation. The AEAD and hash function can share the same permutation. For example, AEAD schemes Keyak [1], Ketje [2] and Kravatte [3] all use the permutation of KECCAK-p. The advantage of sponge construction is that the security of AEAD and hash can be proved given that the underlying permutation is a publicly random one. So the permutation must be a strong component to support the theory of sponge.

The other way is to use blockcipher as shared component. There are a lot of AEAD modes including CCM[4], EAX[5], GCM[6], and OCB[7]. But the theory of blockcipher-based hash function is not practical. The security of blockcipher mode supposes that the blockcipher is a pseudorandom permutation (PRP) and blockcipher based hash function treats the blockcipher as an ideal cipher.

These two ways need strong components, increasing the area and latency of hardware implementation. Furthermore, these constructions also suffer security losses due to their simple structures, such as birthday attacks. Plenty of research is devoted to improving the security bound of sponge function [8, 9, 10, 11, 12] or constructing beyond-birthday-bound schemes [13, 14].

The third way is to adopt stream cipher style construction, such as AEAD scheme ACORN [15]). Like the sponge function, it also applies a permutation to operate on finite state variables, interleaved with the entry of input or the retrieval of output. But here the permutation does not need to be a strong one and can be implemented in a small scale circuit. We call it *thin sponge* function in this document. The security of thin sponge function is guaranteed by its resistance to specific attacks including differential attack, cube attack, etc. *However, the construction of hash function based on thin sponge has not been addressed in the literature.*

In this document, we give both AEAD and hash function constructions, called HERN and HERON respectively, by thin sponge function.

The permutation in HERN consists of four linear feedback shift registers (LFSRs) concatenated in a circle combined with some nonlinear feedback operations. As the state of HERN is too small, we use a buffer to expand the state of HERON. The buffer is a pipe used to temporarily store a segment of key stream. The basic operations and other state variables are the same as in HERN, leading to reduced implementation costs both in hardware and software.

The document is organized as follows. Chapter 2 gives the specifications of HERN and HERON. Chapter 3 gives the design rationale. Chapter 4 gives the security goals. Chapter 5 gives the security analysis. Chapter 6 gives the software and hardware performances. Chapter 7 concludes the advantages and limitations of HERN and HERON.

Chapter 2

Specifications

2.1 Overview

HERN and HERON are lightweight encryption authentication with associated data (AEAD) scheme and hash function respectively. HERN and HERON are bit-based constructions and process 1-bit message in each step. 32 steps can be computed in parallel.

HERN consists of two deterministic algorithms: an encryption algorithm HERN.Enc and a decryption algorithm HERN.Dec. The encryption algorithm takes as input a 128-bit key K, a 128-bit initialization vector IV, associated data A and a plaintext P and outputs a ciphertext (C, T) where T is a 128-bit authentication tag. We write it as HERN.$Enc_K(IV, A, P) = (C, T)$. The decryption algorithm HERN.Dec takes in a tuple (K, IV, A, C, T) and outputs P or a special symbol $\perp$ indicating that the ciphertext is invalid. We require that HERN.$Dec_K(IV, A, C, T) = P$ if HERN.$Enc_K(IV, A, P) = (C, T)$. HERON takes as input a message M, and outputs a 256-bit digest D. We write it as HERON$(M) = D$.

In HERN the associated data length and the plaintext length are less than 2^{64} bits. In HERON the message length is less than 2^{64} bits.

The parameters of HERN and HERON are listed in the following:

- HERN: 128-bit key, 128-bit IV, 128-bit tag.
- HERON: 256-bit digest.

2.2 Operations and round functions

2.2.1 Symbols and operations

The following symbols are used in HERN and HERON:

- $\oplus$: bit-wise exclusive OR (XOR).
- $\cdot$: bit-wise AND.
- $\|$: concatenation.
- $x << 1$: shift x to the left by 1 bit.
- l_M: bit length of M.

- $A^{(m)}$: the string formed by repeating the string A for m times. For example $0^{(4)} = 0000$ and $01^{(2)} = 0101$.
- ϵ: empty string.
- $0x$: hexadecimal string representation of binary string with leftmost being the highest bit, such as $0x2 = 0010$.

The state variables of HERN consists of two components. Firstly, four 64-bit variables

$$S^i = s_0^i s_1^i \cdots s_{63}^i, i = 0, 1, 2, 3$$

implement a concatenated linear feedback shift register (LFSR). Secondly, two 1-bit variables a and b, maintaining nonlinear feedback bits, are used to process input bit and generate output bit respectively.

As the state of HERN is too small for hash function construction, in addition to S^0, S^1, S^2, S^3, a and b, HERON uses another 512-bit buffer state variables:

$$S^4 = s_0^4 s_1^4 \cdots s_{511}^4.$$

The state variables are always preserved and updated during the computation of HERN and HERON. Three basic operations on state variables S^0, S^1, S^2, S^3, a and b are defined i.e. H_core_step, Adda and Addb, which are shared by both HERN and HERON. Based on these operations the round functions that process a bit in HERN and HERON are defined. We show all these variables, operations and functions in Fig.2.1.

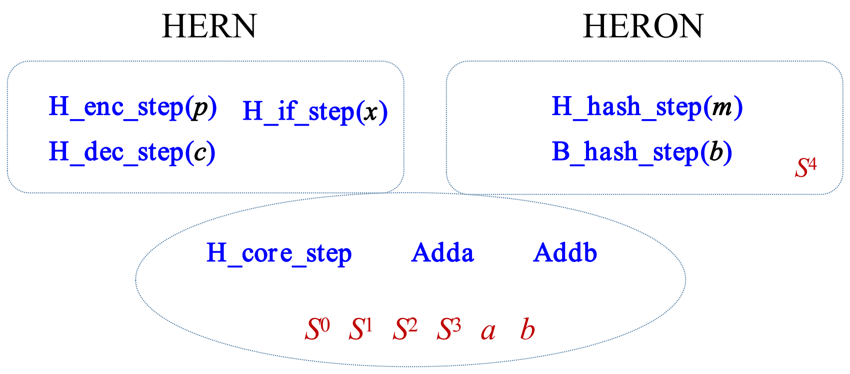

Figure 2.1: Basic operations and functions in HERN and HERON.

The operation H_core_step defined by Algorithm 1, stores the nonlinear feedback bits in a and b, and updates the state using LFSR as illustrated in Fig.2.2.

Two nonlinear functions from 8 bits to 1 bit are used in H_core_step: SB and SB'.

- $SB(x_0, y_0, x_1, y_1, x_2, y_2, x_3, y_3) = 1 \oplus x_0 \cdot y_0 \oplus x_1 \cdot y_1 \oplus x_2 \cdot y_2 \oplus x_3 \cdot y_3$,
- $SB'(x_0, y_0, x_1, y_1, x_2, y_2, x_3, y_3) = x_0 \cdot y_2 \oplus y_0 \cdot y_3 \oplus x_1 \cdot x_3 \oplus y_1 \cdot x_2$.

The operation Adda defined by Algorithm 2, XORs the value of a into s_{63}^0 and s_{63}^2.

The operation Addb defined by Algorithm 3, XORs the value of b into s_{63}^1 and s_{63}^3.

The operations H_core_step, together with Adda and Addb, form a simple permutation on state variables of S^0, S^1, S^2, S^3.

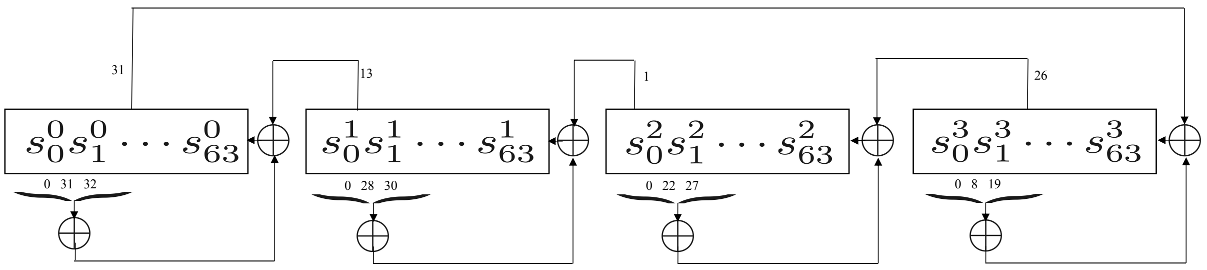
31
13
1
26
$s^0_0 s^0_1 \cdots s^0_{63}$
$s^1_0 s^1_1 \cdots s^1_{63}$
$s^2_0 s^2_1 \cdots s^2_{63}$
$s^3_0 s^3_1 \cdots s^3_{63}$
0 31 32
0 28 30
0 22 27
0 8 19

Figure 2.2: The LFSR in HERN and HERON is four LFSRs concatenated in a circle where one bit of one register affects its previous register.

Algorithm 1: H_core_step

```
1  /* Compute the nonlinear feedback bits*/
2  $a \leftarrow SB(s^0_{30}, s^0_{29}, s^1_{32}, s^1_{24}, s^2_{31}, s^2_4, s^3_{15}, s^3_{14})$
3  $b \leftarrow SB'(s^0_{30}, s^0_{29}, s^1_{32}, s^1_{24}, s^2_{31}, s^2_4, s^3_{15}, s^3_{14}) \oplus s^0_{32}$
4  /* Compute the linear feedback bits*/
5  $f^0 \leftarrow s^0_0 \oplus s^0_{31} \oplus s^0_{32} \oplus s^1_{13}$
6  $f^1 \leftarrow s^1_0 \oplus s^1_{28} \oplus s^1_{30} \oplus s^2_1$
7  $f^2 \leftarrow s^2_0 \oplus s^2_{22} \oplus s^2_{27} \oplus s^3_{26}$
8  $f^3 \leftarrow s^3_0 \oplus s^3_8 \oplus s^3_{19} \oplus s^0_{31}$
9  /* Update state*/
10 $S^i \leftarrow S^i << 1$, for $i = 0, 1, 2, 3$
11 $s^i_{63} \leftarrow f^i$, for $i = 0, 1, 2, 3$
```

Algorithm 2: Adda

```
1 $s^0_{63} \leftarrow s^0_{63} \oplus a$
2 $s^2_{63} \leftarrow s^2_{63} \oplus a$
```

Algorithm 3: Addb

```
1 $s^1_{63} \leftarrow s^1_{63} \oplus b$
2 $s^3_{63} \leftarrow s^3_{63} \oplus b$
```

2.2.2 Round functions

The round functions process the data bit-wisely in HERN and HERON. We list them in the following.

- H_if_step(x) defined by Algorithm 4, is used to process the bit of IV, associated data or generate authentication tag in HERN at initialization or finalization stage.
- H_enc_step(p) defined by Algorithm 5, is used in HERN to process the bit of plaintext to generate

the bit of ciphertext.

- H_dec_step(c) defined by Algorithm 6, is used in HERN to process the bit of ciphertext.
- B_hash_step(b) defined by Algorithm 7, is used in HERON to update the buffer state.
- H_hash_step(m) defined by Algorithm 8, is used in HERON to process the bit of message.

Algorithm 4: Initialization and finalization round function H_if_step(x)

```
Input: 1-bit $x$
1 H_core_step
2 $a \leftarrow a \oplus x$
3 Adda
4 Addb
```

Algorithm 5: Encryption round function H_enc_step(p)

```
Input: 1-bit plaintext $p$
Output: 1-bit ciphertext $c$
1 H_core_step
2 $a \leftarrow a \oplus p$
3 Adda
4 $c \leftarrow b \oplus p$
5 return $c$
```

Algorithm 6: Decryption round function H_dec_step(c)

```
Input: 1-bit ciphertext $c$
Output: 1-bit plaintext $p$.
1 H_core_step
2 $p \leftarrow b \oplus c$
3 $a \leftarrow a \oplus p$
4 Adda
5 return $p$
```

2.3 AEAD construction HERN

2.3.1 The encryption HERN.Enc

The encryption is divided into the following 3 stages.

Algorithm 7: Buffer state update function B_hash_step(b)

```
    Input: 1-bit $b$
    Output: 1-bit $d$
1 $d \leftarrow s^4_0$
2 $S^4 \leftarrow S^4 << 1$
3 $h \leftarrow d \oplus b$
4 $s^4_{511} \leftarrow h$
5 return $d$
```

Algorithm 8: Hash round function H_hash_step(m)

```
    Input: 1-bit message $m$
1 H_core_step
2 $a \leftarrow a \oplus m$
3 $b \leftarrow$ B_hash_step($b$)
4 Adda
5 Addb
```

1) Initialization. The initialization consists of loading the key K and constant CT into state variables and processing the initialization vector IV, associated data A and running H_if_step for 512 steps with zero input.

- Load the state variables S^i $(i = 0, 1, 2, 3)$ with constant $CT = ct_0ct_1 \cdots ct_{127} = 0x$ 14 f1 c2 72 32 79 c4 19 4b 8e a4 1d 0c c8 08 63 and the key $K = k_0k_1 \cdots k_{127}$ as follows.

  Set $s^0_i = ct_i$, for $i = 0$ to 7,
  Set $s^0_{i+8} = k_i$, for $i = 0$ to 31,
  Set $s^0_{i+40} = ct_{i+8}$, for $i = 0$ to 23,
  Set $s^1_i = ct_{i+32}$, for $i = 0$ to 7,
  Set $s^1_{i+8} = k_{i+32}$, for $i = 0$ to 31,
  Set $s^1_{i+40} = ct_{i+40}$, for $i = 0$ to 23,
  Set $s^2_i = ct_{i+64}$, for $i = 0$ to 7,
  Set $s^2_{i+8} = k_{i+64}$, for $i = 0$ to 31,
  Set $s^2_{i+40} = ct_{i+72}$, for $i = 0$ to 23,
  Set $s^3_i = ct_{i+96}$, for $i = 0$ to 7,
  Set $s^3_{i+8} = k_{i+96}$, for $i = 0$ to 31,
  Set $s^3_{i+40} = ct_{i+104}$, for $i = 0$ to 23,
  Set $a = b = 0$.

- Process $IV = iv_0iv_1 \cdots iv_{127}$. At each step, one bit of IV is used to update the state.

  - $\mathsf{H\_if\_step}(iv_i)$ for $i = 0$ to 127.

- Process $A = ad_0ad_1 \cdots ad_{u-1}$. At each step, one bit of A is processed to update the state.
  - $\mathsf{H\_if\_step}(ad_i)$, for $i = 0$ to $u - 1$.
- Run the $\mathsf{H\_if\_step}$ for 512 steps with zero-stream.
  - $\mathsf{H\_if\_step}(0)$, for $i = 0$ to 511.

2) Processing plaintext. The plaintext $P = p_0p_1 \cdots p_{v-1}$ is used to update the state bit-by-bit and the corresponding ciphertext bit is generated.

- $C \leftarrow \epsilon$.
- $c_i \leftarrow \mathsf{H\_enc\_step}(p_i)$, $C \leftarrow C \| c_i$, for $i = 0$ to $v - 1$.

3) Finalization. After processing all the plaintext bits, the finalization consists of running $\mathsf{H\_if\_step}$ for 512 steps with zero-stream input, and generating the tag.

- $\mathsf{H\_if\_step}(0)$, for $i = 0$ to 511.
- $T \leftarrow \epsilon$.
- $t \leftarrow \mathsf{H\_enc\_step}(0)$, $T \leftarrow T \| t$, for $i = 0$ to 127.
- return (C, T).

2.3.2 The decryption $\mathsf{HERN}.Dec$

The decryption is also divided into 3 stages.

1) Initialization. The initialization is the same as in the encryption.

2) Processing ciphertext. The plaintext $C = c_0c_1 \cdots c_{v-1}$ is used to update the state bit-by-bit and the corresponding plaintext bit is generated.

- $P \leftarrow \epsilon$.
- $p_i \leftarrow \mathsf{H\_dec\_step}(c_i)$, $P \leftarrow P \| p_i$, for $i = 0$ to $v - 1$.

3) Finalization. After processing all the ciphertext bits, the finalization consists of running $\mathsf{H\_if\_step}$ for 512 steps with zero-stream input, and generating the tag.

- $\mathsf{H\_if\_step}(0)$, for $i = 0$ to 511.
- $T' \leftarrow \epsilon$.

- $t \leftarrow \mathsf{H\_enc\_step}(0)$, $T' \leftarrow T' \| t$, for $i = 0$ to 127.
- return P if $T' = T$; $\perp$ otherwise.

2.4 Hash function construction HERON

HERON is divided into the following 3 stages.

1) Initialization. At the initialization of HERON, constants are loaded into the state.

- Load the state variables S^i $(i = 0, 1, 2, 3, 4)$ with constant $CT_0 = 0x$ 14 f1 c2 72, $CT_1 = 0x$ 32 79 c4 19, $CT_2 = 0x$ 4b 8e a4 1d, $CT_3 = 0x$ 0c c8 08 63 and $CT_4 = 0x$ d2 80 62 e1 e7 1d 3d da e3 c4 d1 58 a7 f0 67 ac 94 93 50 56 8e e5 c6 3d f5 a0 ce c3 d3 3d a5 a7 7d e8 92 ac e8 fd 9b 12 fb 62 5a 84 f1 5a 53 23 d9 3d 39 95 9a 48 5a 71 da b8 ec d1 9d 9b 3e 2e as follows.
  Set $S^0 = S^2 = CT_0 \| CT_1$,
  Set $S^1 = S^3 = CT_2 \| CT_3$,
  Set $S^4 = CT_4$.

2) Processing message. The message M is padded with minimal 0s so that its bit-length is the multiple of 32, then it is encoded by inserting 32-bit zero blocks.

- Padding M to $\overline{M}$.
  - $\overline{M} = M \| 0^{(w)}$

  where $w = (32 \cdot \lceil \frac{l_M}{32} \rceil - l_M)$.
- Encode $\overline{M}$ into MH.
  - $\overline{M} = \overline{M}_0 \| \overline{M}_1 \| \cdots \| \overline{M}_{l_{\overline{M}}/32-1}$
  - $MH = \overline{M}_0 \| 0^{(32)} \| \overline{M}_1 \| 0^{(32)} \| \cdots \| \overline{M}_{l_{\overline{M}}/32-1} \| 0^{(32)} = mh_0 mh_1 \cdots mh_{l_{MH}-1}$

  where $\overline{M}_0, \overline{M}_1, \cdots$ are 32-bit message blocks.
- Run $\mathsf{H\_hash\_step}$ using MH.
  - $\mathsf{H\_hash\_step}(mh_i)$, for $i = 0$ to $l_{MH} - 1$.

3) Finalization. After processing the encoded message stream, a 1024-bit string $U = bmlen^{(16)} \oplus CT4 \| 0^{(512)} = u_0 u_1 \cdots u_{1023}$ and the mid bit s^4_{256} in buffer are used to update the state of HERON and generate the digest D, where $bmlen$ is the binary representation of the length of M in 64-bit string.

- $\mathsf{H\_hash\_step}(u_i \oplus s^4_{256})$, for $i = 0$ to 1023.
- $D \leftarrow s^4_{256} s^4_{257} \cdots s^4_{511}$.

Chapter 3

Design rationale

Objective. We aim to design both AEAD and hash function that can be implemented in a single small circuit. HERN and HERON are also designed to be efficient in hardware and software.

HERN and HERON in common. The state variables S^i, $i = 0, 1, 2, 3$, a, b and the basic operations H_core_step, Adda and Addb are shared by HERN and HERON. All the round functions in HERN and HERON are based on these basic operations.

The LFSR in HERN and HERON is formed by 64-bit registers concatenated in a circle. 64-bit register is suitable for software implementation. a and b are used to store the nonlinear feedback bits which are XORed to the register by operations Adda and Addb.

SB and SB' are the only two nonlinear functions in HERN and HERON. SB and SB' are suitable for lightweight implementations, because their degree are only 2.

Although the specifications of HERN and HERON process the data bit-wisely, up to 32 steps can be computed in parallel. The feature benefits the performance on a wide rang of 8-bit, 16-bit and 32-bit microcontroller architectures.

The design of HERN. The initialization stage processes the IV and associated data by the same function H_if_step(x) and then runs H_if_step(0) 512 times. The encryption or decryption stage uses H_enc_step(p) and H_dec_step(c) to process the bit of plaintext or ciphertext. The finalization stage runs H_if_step(0) 512 times and uses H_enc_step(0) to generate the tag. We notice that H_enc_step(p) and H_dec_step(c) do not include the operation of Addb, but H_if_step(x) does. So that different stages are separated by using different operations.

The design of HERON. In addition to state variables and operations in HERN, HERON uses another 512-bit buffer state variable, in order to enlarge the state of HERN. The buffer state is connected to the state of HERN by the function B_hash_step(b) which uses only 1-bit XOR and shift operations, so that it can be carried out outside hardware circuit. Therefore HERON uses almost the same circuit in HERN.

All state variables are initiated by constants. The function H_hash_step(m) is used to process the bit of message and update state variables in the finalization stage. The security of HERON relies on two assumptions. First, it is hard to find two different messages that lead to a same state value. Second, the finalization stage should behave like a random oracle to the input of state value and length of message.

Choice of constants. The constants CT, CT_i, $i = 0, 1, 2, 3, 4$ in HERN and HERON are key-streams generated by stream cipher ZUC [16] with key $0x$ 3d 4c 4b e9 6a 82 fd ae b5 8f 64 1d b1 7b 45 5b and IV $0x$ 84 31 9a a8 de 69 15 ca 1f 6b da 6b fb d8 c7 66. Furthermore, some of the constants are reused i.e. $CT = CT_0 \| CT_1 \| CT_2 \| CT_3$.

Chapter 4

Security goals

The security goals of HERN are given in Table 4.1. HERN is designed to have confidentiality of the plaintexts under adaptive chosen-plaintext attacks and the integrity of the ciphertexts under adaptive forgery attacks. Note the IV should be use as nonce in HERN.*Enc* that never repeat and plaintext shall not be returned by HERN.*Dec* if the verification fails.

Table 4.1: Security Goals of HERN

	Confidentiality	Integrity
HERN	128-bit	128-bit

The security goals of HERON are given in Table 4.2.

Table 4.2: Security Goals of HERON

	Preimage	Collision
HERON	256-bit	128-bit

Chapter 5

Security analysis

The security of HERN should consider how many steps are needed to make sure that the initialization and finalization will behave like a random oracle. The most related cryptanalysis are differential cryptanalysis and cube attack. The key stream should be generated in a secure and random way.

In HERON, the state is consisted of two parts, the register part and the buffer part. We consider the differential trails which has 2 stages.

5.1 Differential analysis of initialization

We consider the differential propagation in the state. In our analysis, we introduce difference into IV, then compute the number of active S-box and differential probability. The SB and SB' can be seen as a 8-2-bit S-box. The S-box is active if and only if the input difference is non-zero. The numbers of active S-box are 27 after 128 steps, and the differential probability is nearly 2^{-47} because of the dependency of the S-boxes. As the limited computing resource, the smallest numbers of active S-box we found are 72, 156 and 216 after 192, 256 and 320 steps respectively. The differential probability is nearly 2^{-129}, 2^{-304} and 2^{-416} after 192, 256 and 320 steps.

From the above experiments, we think that the initialization has enough security margin against differential cryptanalysis.

5.2 Cube analysis

Cube attack is a common analysis method against stream ciphers, stream cipher based constructions and hash function. It is effective against the ciphers with low algebraic degree, or against the ciphers with high algebraic degree but the system of nonlinear equations of the cipher is very sparse.

5.2.1 Cube analysis on HERN

In the HERN initialization, there are 4 linear feedback shift registers, and there are 8 taps being used in the feedback, we expect that the system of equations of HERN would be dense.

We performed experiments to estimate the algebraic degrees of key stream bits in terms of the K and IV. In particular, we analyze whether the term $IV_{i_0} \cdot IV_{i_1} \cdot IV_{i_2} \cdots IV_{i_{n-1}}$ affects the key stream. In our experiments, we focus on the term $IV_{128-n} \cdot IV_{128-n+1} \cdots IV_{127}$ which is the product of the last n bits of IV. Note that in the initialization, the ith IV bit is XORed to the state at the ith step, so the last n bits of IV bits are expected to have the least effect on the key stream. In the experiments, we set the rest of IV bits to zero, and use 16 random keys for confirmation.

Table 5.1 shows the minimum number of steps that are needed so that all the key stream bits are affected by the term $IV_{128-n} \cdot IV_{128-n+1} \cdots IV_{127}$, which is the product of the last n bits of IV. For example $n = 1$, the last IV bit IV_{127} is XORed to the state at the 128th step, then it takes at least 92 steps for this bit being shifted through the state to effect the key stream bit after 219th step. In total, it takes at least 220 steps so that the last bit IV_{127} affects all the key stream bits.

Table 5.1: The minimum number of steps after which key stream bit is always affected by $IV_{128-n} \cdot IV_{128-n+1} \cdots IV_{127}$.

n	1	2	3	4	5	6	7	8
steps	220	251	251	253	259	260	260	270
n	9	10	11	12	13	14	15	16
steps	274	284	284	284	287	287	287	289
n	17	18	19	20	21	22	23	24
steps	290	291	291	293	300	304	307	307
n	25	26	27	28	29	30	31	32
steps	307	308	309	309	310	311	311	311

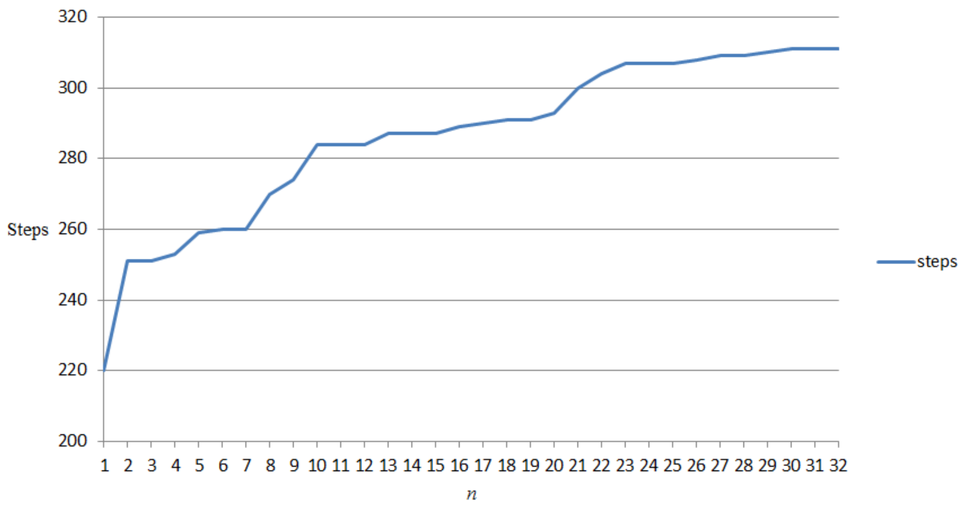

Figure 5.1: The increasing of minimum number of steps as n increases.

Fig.5.1 shows the increasing of the minimum number of steps as the cube size increases. From Table 5.1 and Fig.5.1, we could observe that as the cube size increases, the minimum number of steps increases at a almost decelerated pace. After computation, we could find that steps($2n$)$-$steps(n) ≤ 48, so steps(128) $\leq$ steps(64) $+ 48 \leq$ step(32) $+ 96 = 407$, which is less than $128 + 512 = 640$. We thus expect HERN has large security margin against the cube attack.

5.2.2 Cube analysis on HERON

In addition to state variables and operations in HERN, HERON uses another 512-bit buffer state variable, in order to enlarge the state of HERN, so we also expect that the system of equations of HERON would be dense.

We perform experiments to estimate the algebraic degrees of the nonlinear feedback bit to buffer (referred as b below) in terms of the processed message. In particular, we analysis whether the term $mh_{95-n} \cdot mh_{95-n+1} \cdot mh_{95}$ (encoded from the second message block $\overline{M}_1$) affects b. In the experiments, we set $MH = \overline{M}_0||0^{(32)}||\overline{M}_1||0^{(32)}$, the rest of $\overline{M}_1$ bits to zero and use 16 random $\overline{M}_0$ for confirmation.

Table 5.2 shows the minimum number of steps that are needed so that b of each steps are all affected by the term $mh_{95-n} \cdot mh_{95-n+1} \cdot mh_{95}$, which is the product of the last n bits of $\overline{M}_0$. For example $n = 1$, the last bit mh_{95} is processed at the 96th step, then $0^{(32)}$ are processed, and in finalization phase, it takes at least 91 steps for this bit to effect every b.

Table 5.2: The minimum number of steps after which the nonlinear feedback bit to buffer is always affected by $mh_{95-n} \cdot mh_{95-n+1} \cdot mh_{95}$.

n	1	2	3	4	5	6	7	8
steps	91	124	124	124	124	146	149	149
n	9	10	11	12	13	14	15	16
steps	149	150	151	151	151	151	153	153
n	17	18	19	20	21	22	23	24
steps	153	154	155	155	156	158	169	172
n	25	26	27	28	29	30	31	32
steps	172	172	177	179	179	179	179	180

Fig.5.2 shows the increasing of the minimum number of steps as the cube size increases. From Table 5.2 and Fig.5.2, we could observe that as the cube size increases, the minimum number of steps increases at a almost decelerated pace. After computation, we could find that steps($2n$)$-$steps(n) ≤ 36, so steps(256) $\leq$ steps(128) $+ 36 \leq$ step(64) $+ 72 \leq$ step (32) $+ 108 = 288$, which is far less than 1024. We thus expect HERON has large security margin against the cube attack.

5.3 Security analysis of HERON

The security of HERON is reduced to the problem of finding collisions: to find two messages $M \neq M'$ so that $S^M = S^{M'}$, where S^M is the state $(S^0, S^1, S^2, S^3, S^4)$ after processing all message M. In general, the problem of finding collisions will be transformed into solving systems of equations. This kind of systems of

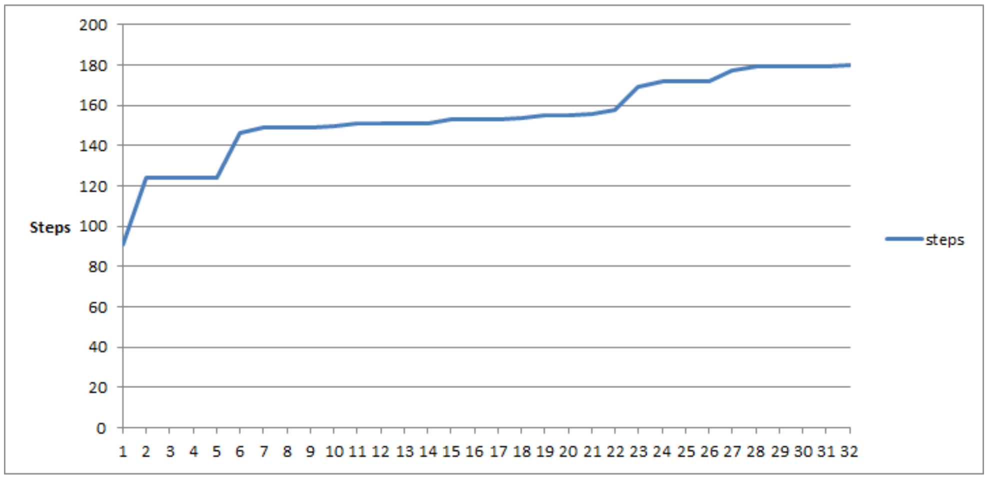

Figure 5.2: The increasing of minimum number of steps as n increases.

equations are too complex for existing solvers. The only known method for solving this type of equations is differential cryptanalysis.

Differential cryptanalysis is a method to find a differential trail which has the smallest cost. We denote a as the bit a input to Adda, and $\underline{a}$ is the sequence consisting of a for all message process steps. A differential trail is a sequence pair $(\underline{a}, \underline{a}')$, which is the differential sequence. Consider the differential cryptanalysis in the processing message process. A full valid trail is the one that makes the whole state from 0 to 0. Note that $\underline{a}'$ is decided by $\underline{a}$ and the output of $SB^{'}$(which must be 0 for inactive S-boxes). At each step, a is either the output of SB or arbitrary, which we say the S-box is at constant position or message position respectively. The cost of a trail denoted the work factor to realize this trail, when the cost is q, we mean the work factor is 2^q.

We can count the cost of a trail as follows: each active S-box at constant position increases the cost by two and each non-active S-box at message position decreases the cost by one. The cost of a differential trail is the maximum of costs of its all sub-trails.

In HERON, the state is consisted of two parts, the register part and the buffer part. We consider the following trail which has 2 stages.

In the first stage, stream $\underline{a}$ ($\underline{a}'$=0) is chosen as a small multiple of minimal polynomial of $\mathcal{A}$, where $\mathcal{A}$ is the following linear matrix corresponding to the linear update functions of the register part. The difference of buffer part has no effects on the register part in the first state.

$$\mathcal{A} = \begin{pmatrix} \mathcal{A}_0 & \mathcal{Z}_0 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathcal{A}_1 & \mathcal{Z}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathcal{A}_2 & \mathcal{Z}_2 \\ \mathcal{Z}_3 & \mathbf{0} & \mathbf{0} & \mathcal{A}_3 \end{pmatrix} \tag{5.1}$$

Denote e_j be 1×64 unit vector with j-th element be 1. $\mathbf{0}$ in $\mathcal{A}$ is 64×64 zero matrix. According to H_core_step, $\mathcal{A}_i, \mathcal{Z}_i$ are 64×64 matrix on $\mathbf{F}_2$. j-th$(j = 0, 1, \ldots, 62)$ row of all $\mathcal{A}_i, i = 0, \ldots, 3$ is e_{j+1}, 63-th row of $\mathcal{A}_0(\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$is $e_0 \oplus e_{31} \oplus e_{32}(e_0 \oplus e_{28} \oplus e_{30},\ e_0 \oplus e_{22} \oplus e_{27}, e_0 \oplus e_8 \oplus e_{19})$. Each of the first 62 rows of $\mathcal{Z}_i$ is all zero vector, and 63-th of $\mathcal{Z}_0(\mathcal{Z}_1, \mathcal{Z}_2, \mathcal{Z}_3)$ is $e_{13}(e_1, e_{26}, e_{31})$. Then the state update in H_core_step can be expressed in the following, where S^T is the transposed 64×1 vector of S:

$$S^T \leftarrow \mathcal{A} \cdot S^T \tag{5.2}$$

The small multiple makes almost all the S-boxes active so that this sub-trail is valid. This stream makes register state zero, and makes buffer state containing a segment $\underline{M'}$, where $\underline{M'}$ is determined later. The number of active S-boxes in this sub-trail is about $(l_S - c_1)$, where $l_S = 256$ is the size of the register state here, and c_1 is a small constant. According to previous subsection, the cost of this sub-trail is about

$$2 \cdot (l_S - c_1) \cdot (1 - \rho).$$

$\rho = \frac{|\overline{M_0}|}{|\overline{M_0}|+32} = \frac{1}{2}$, where $|\overline{M_0}|$ is the length of first coded message block.

Let M, M' be sequences of suitable length which transform the register state from 0 to 0. This suitable length is about half of l_S, so the cost of this sub-trail is

$$2 \cdot (\frac{l_S}{2}) \cdot (1 - \rho).$$

After the first stage and a segment of zero input,we can make the input to the register is ($\underline{\mathrm{a}} = \underline{M}, \underline{\mathrm{a}}' = \underline{M'}$), which makes it zero and at the same time clear-off the buffer contents $\underline{M'}$.

The cost of the whole trail is the maximum of the two, *i.e.*

$$2 \cdot (l_S - c_1) \cdot (1 - \rho).$$

This gives an upper bound of security strength of HERON.

We give an argument that it is also the lower bound. Consider trails which begin with zero state and end before the buffer output affects S-boxes, i.e. $\underline{\mathrm{a}}' = \underline{0}$. The length of these trails are $l_B + c_2$, where c_2 is a constant (in our example, $c_2 > 64$), and l_B is the length of buffer. In any valid trail the inactive S-boxes' input forms a linear space, which is a subspace of the space generated by variables in $\underline{\mathrm{a}}$, where each bit is considered a variable. Each S-box gives a subspace of dimension 8, some of which overlap with other S-boxes, so we pretend that each S-box gives an independent subspace of dimension 4. The total space has dimension less than $l_B + c_2$, the number of inactive S-boxes is less than $\frac{l_B + c_2}{4}$ and the number of active S-boxes is more than $\frac{3}{4}(l_B + c_2)$. As a result, the cost of the trail would be larger than $2(\frac{3}{4} - \rho)(l_B + c_2)$. This suggests that our upper bound above is also a lower bound if l_B is large enough.

If the length of buffer is large enough, the security strength will not improve as the length of buffer adds. Thus $l_B = 512$ is enough.

5.4 Length extension attack

Typically, length extension attacks use the hash value of one message and its length to the hash value of the other message that takes the previous message as prefix. Hash functions that use MD structure are susceptible to this kind of attack such as MD-5, SHA-1, SHA-2. The sponge function based hash function SHA-3 is not susceptible.

HERON adopts the thin sponge function construction which is divided into two stages. In the message processing stage, it is hard to find two different messages that lead to a same state value. The finalization stage behaves like a random oracle to the input of state value and length of message. As to the strength of the finalization stage, even the attacker knows the hash value and the message length, it is difficult to get the hash value of the extended message.

Chapter 6

Performance

6.1 Hardware implementations

Target device is XILINX Artix-7 AC701 Evaluation Platform (xc7a200tfbg676-2). To minimize the area, the datapath can be 1 bit.

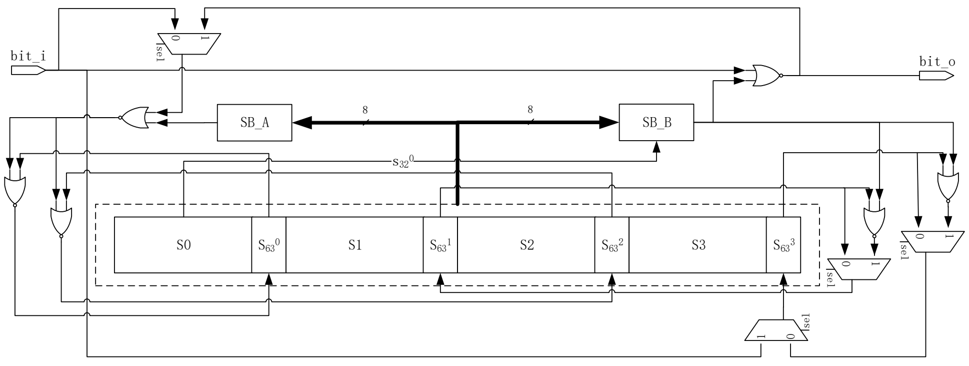

Figure 6.1: The 1-bit datapath HERN core

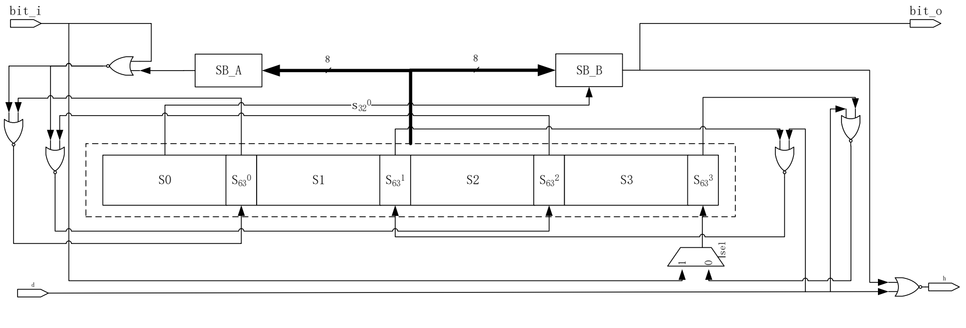

Figure 6.2: The 1-bit datapath HERON core

Fig.6.1 shows the basic structure of the HERN implementation on FPGA. Fig.6.2 shows the HERON implementation. The most expensive module is state register (S0-S3), other modules are basic combinational circuits. The modules absorb 1 bit from the input port and output the result bit by bit. The circuits have no complex structure. We further integrate the HERN and HERON into the circuit.

We further implement HERN and HERON in byte by byte modules. As shows in Table 6.1 HERN and HERON is quite lightweight.

Table 6.1: Performance on FPGA.

Core	Module	Slice	LUT	FF	Cycles	Frequency/MHz
HERN	bit	10	30	25	1536+u+v	146
HERON	bit	9	27	25	1280+w	176
HERN+HERON	bit	11	32	25	-	141
HERN	byte	39	127	128	192+u/8+v/8	134
HERON	byte	33	105	128	160+w/8	152

6.2 Software implementations

HERN and HERON are implemented in C code. We have the test run on Intel Core i7-6500U 2.5GHz processor and Windows 10 Pro version 1809. We use Visual Studio 2017 as the IDE and compile our code with the option "Maximum Optimization (Favor Speed) (/O2)". The associated data length in HERN is always zero. IV length and tag length are both 128 bits. In HERON, the digest length is 256 bits. We just use the change of message length to evaluate the speed of HERN and HERON. In a test with processing same message, we run algorithm 1000 times and use the Median as the running time.

Table 6.2: The speed (cpb) of HERN and HERON.

	64B	128B	256B	512B	1024B	2048B	4096B
HERN	54.16	33.14	20.71	16.02	13.56	12.39	11.80
HERON	74.84	49.86	38.46	32.82	30.03	28.59	27.90

Chapter 7

Advantages and limitations

In this document we design two cryptographic algorithms: an AEAD scheme HERN and a hash function HERON. The main advantages of HERN and HERON are:

- One stone, two birds. We construct both HERN and HERON that share the same state variables and basic operations leading to reduced implementation costs both in hardware and software.
- Lightweight implementation. The feedback functions are either linear or have only degree of 2 and one message bit is processed in each step. These features benefit lightweight implementation.
- Flexibility of implementation. The feature that up to 32 steps can be computed in parallel benefits implementations on a wide rang of 8-bit, 16-bit and 32-bit microcontroller architectures.
- Efficiency in hardware and software. In HERN and HERON, 32 steps can be computed in parallel, so its speed is reasonably fast.

Acknowledgment

We would like thank all that participate in our discussion, testing or writing. They are Gang Liu, Yingjie Zhang, Qiu Chen, Zihao Wei, Zhaohui Chen.

Bibliography

[1] Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. CAESAR submission: Keyak v2, 2016. `https://competitions.cr.yp.to/round3/keyakv22.pdf`.

[2] Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. CAESAR submission: Ketje v2, 2016. `https://competitions.cr.yp.to/round3/ketjev2.pdf`.

[3] Guido Bertoni, Joan Daemen, Seth Hoffert, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. Farfalle: parallel permutation-based cryptography. *IACR Trans. Symmetric Cryptol.*, 2017(4):1–38, 2017.

[4] Morris Dworkin. *SP 800-38C. Recommendation for Block Cipher Modes of Operation: the CCM Mode for Authentication and Confidentiality.* National Institute of Standards Technology, 2005.

[5] Mihir Bellare, Phillip Rogaway, and David A Wagner. The EAX mode of operation. 3017:389–407, 2004.

[6] David A. McGrew and John Viega. The security and performance of the Galois/Counter mode of operation (full version). *IACR Cryptology ePrint Archive*, 2004:193, 2004.

[7] Ted Krovetz and Phillip Rogaway. The software performance of authenticated-encryption modes. In *FSE 2011*, pages 306–327, 2011.

[8] Philipp Jovanovic, Atul Luykx, and Bart Mennink. Beyond $2^{c/2}$ security in sponge-based authenticated encryption modes. In *Advances in Cryptology - ASIACRYPT 2014. Proceedings, Part I*, pages 85–104, 2014.

[9] Bart Mennink, Reza Reyhanitabar, and Damian Vizár. Security of full-state keyed sponge and duplex: Applications to authenticated encryption. In *Advances in Cryptology - ASIACRYPT 2015, Proceedings, Part II*, pages 465–489, 2015.

[10] Elena Andreeva, Joan Daemen, Bart Mennink, and Gilles Van Assche. Security of keyed sponge constructions using a modular proof approach. In *FSE 2015*, pages 364–384, 2015.

[11] Yusuke Naito and Kan Yasuda. New bounds for keyed sponges with extendable output: Independence between capacity and message length. In *FSE 2016*, pages 3–22, 2016.

[12] Avik Chakraborti, Nilanjan Datta, Mridul Nandi, and Kan Yasuda. Beetle family of lightweight and secure authenticated encryption ciphers. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(2):218–241, 2018.

[13] Thomas Peyrin and Yannick Seurin. Counter-in-tweak: Authenticated encryption modes for tweakable block ciphers. In *Advances in Cryptology - CRYPTO 2016, Proceedings, Part I*, pages 33–63, 2016.

[14] Tetsu Iwata, Kazuhiko Minematsu, Thomas Peyrin, and Yannick Seurin. ZMAC: A fast tweakable block cipher mode for highly secure message authentication. In *Advances in Cryptology - CRYPTO 2017, Proceedings, Part III*, pages 34–65, 2017.

[15] Hongjun Wu. ACORN: A lightweight authenticated cipher (v3). 2016. `https://competitions.cr.yp.to/round3/acornv3.pdf`.

[16] ETSI/SAGE. Specification of the 3GPP confidentiality and integrity algorithms 128-EEA3 & 128-EIA3. document 2: ZUC specification, version: 1.6, 2011. `https://www.gsma.com/aboutus/workinggroups/wp-content/uploads/2014/12/eea3eia3zucv16.pdf`.