Lightweight AE and HASH in a Single Round Function

Dingfeng Ye[1,2], Danping Shi[1,2], and Peng Wang[1,2]

[1]State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, China
[2]Data Assurance and Communication Security Research Center, Chinese Academy of Sciences, China
{ydf, dpshi, wp}@is.ac.cn

Abstract. To deal with message streams, which is required by many symmetric cryptographic functionalities (MAC, AE, HASH), we propose a lightweight round function called Thin Sponge. We give a framework to construct all these functionalities (MAC, AE, and HASH) using the same Thin Sponge round function. Besides the common security assumptions behind traditional symmetric algorithms, the security of our schemes depends on the hardness of problems to find collisions of some states. We give a class of constructions of Thin Sponge, which is improvement of the round function of Trivium and ACORN. We give simple criteria for determining parameters. According to these criteria, we give an example, which achieves all functionalities in a single round function and hence can be realized by the same hardware. Our algorithm is also efficient in software.

Keywords: Thin Sponge, lightweight, MAC, AE, HASH

1 Introduction

Authentication encryption (AE), message authentication code (MAC) and hash function are widely used in modern society. Recently various competitions such as eSTREAM, SHA-3, PHC and CAESAR are held to satisfy the need of more and more applications. Due to the rapid development of lightweight devices, requirements in different application scenarios are proposed, such as IoT, embedded system, smart card and wireless sensor node. In resource-constrained environment, if confidentiality, integrity and identity verification are required, and if all these functionalities are implemented by different circuits, it would violate the resource limits. *We hope to find a solution which implements all the functionalities in a single small circuit.*

There are mainly three approaches to construct MAC or AE scheme. Some use block cipher algorithms, such as cipher block chaining message authentication code(CBC-MAC) [1], One-key MAC(OMAC) [2], Parallelizable MAC(PMAC) [3], CCM[4], EAX[5], GCM[6], and OCB[7]. Some utilize hash functions, such as HMAC [8] and JHAE[9]. The last approach (adopted by ACORN [10]) achieves stream cipher, MAC and AE with a single lightweight round function, similar functions are called Thin Sponge in this paper. However, construction of hash function is not addressed.

There are mainly two common constructions for hash functions. One is based on Merkle-Damgård (MD)-structure, such as SHA-1[11], SHA-2[12] and MD5[13]. And the other is based on a sponge or sponge-like functions, such as Keccak[14], PHOTON[15], Quark[16], SPONGENT[17] and GLUON[18]. However, the modular addition operation used in the MD-structure is not suitable to hardware, and the implementation of sponge functions relies on large hardware, which is very slow in lightweight hardware.

In this paper, we give a heuristic construction of hash function using the Thin Sponge function of an AE. For security, we give an analysis which leads to criteria for deciding parameters for a specific security level. Our analysis suggests that the security strength depends on 2 parameters which are not affected by Thin Sponge function. So our method gives a solution whose security level and assurance can be adjusted at application layer without affecting hardware implementation. In addition, we give an improved Thin Sponge construction over ACORN and give a concrete example.

Thin Sponge function is a lightweight function for processing bit stream as follows:

$$F : \mathbb{S} \times \{0,1\} \to \mathbb{S}$$

where $\mathbb{S}$ is the state space. Here Thin means F can be realized by a small number (tens) of bit operations.

We present our constructions in the following framework. The first thing for setting all functionalities is to construct a MAC using Thin Sponge. Then AE can be heuristically obtained from MAC as in ACORN. To achieve our goal, we only need to construct HASH from AE.

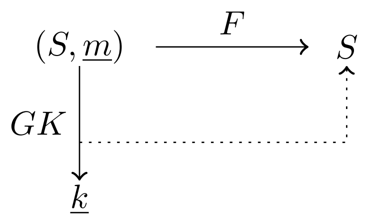
First, we recall the method to construct MAC using Thin Sponge.

- $(k, IV) \xrightarrow{Init} S_0$: use secret key k and IV to get initial state S_0.
- $(S_i, m_i) \xrightarrow{F} S_{i+1}$: update state by Thin Sponge round function F with message stream.
- $RO(S^m) \to Tag$: use a random oracle RO to produce output of MAC, where(and in the follows) S^m denotes state obtained from S by processing a message stream m.

The security strength of this MAC is determined by $\max\limits_{m \neq m'} Pr[S^{\underline{m}} = S^{\underline{m}'}]$ for unknown random S.

AE can be constructed by adding a non-linear filter function GK on MAC state to generate key stream as Fig.(1). At the initialization step, S_0 is obtained as $S_0 \leftarrow S_0^{\underline{0}_l}$, where $\underline{0}_l$ is the all zero bit stream of length l, and l is the number of rounds to repeat a sightly modified round function. The key stream bit $\underline{k}$ is fed back to the state.

HASH is constructed from AE using the same Thin Sponge as in Fig.2. As the AE state is too small, we use a buffer to expand the state of HASH. The



$$\begin{tikzcd}(S,\underline{m}) \arrow[r, "F"] \arrow[d, "GK"'] & S \\ \underline{k} \arrow[ur, dotted] & \end{tikzcd}$$

Fig. 1: AE round function T_{AE}

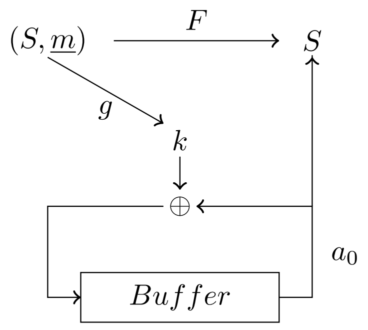


Fig. 2: HASH round function T_{HASH}

buffer is a pipe used to temporarily store a segment of re-encrypted key stream, and its size is denoted by l_B.

The actual message stream inputted to HASH round function is $\widetilde{m}$, which is obtained by dividing original message into fixed-length (such as 32-bit) blocks and inserting some fixed-length constant blocks. The length of a valid message block and a constant block are respectively denoted by l_m and l_c, and the rate of valid messages is denoted as $\rho = \frac{l_m}{l_m+l_c}$.

Heuristically, we guess that the security strength depends on l_B and ρ: bigger l_B and smaller ρ means stronger security.

The process of HASH is as follows:

- $\overline{S}_0$: use fixed random constant bits to fill the initial state $\overline{S}_0$.
- $(\overline{S_i}, \widetilde{m}_i) \xrightarrow{T_{HASH}} \overline{S_i}$: update state by HASH round function T_{HASH} with packed message stream. In this stage, the input of buffer is $k \oplus d$, the output is d, and the bit string in buffer is shifted to the right by 1 bit. At the same time, the output bit of buffer is fed back to AE state as in the AE initialization stage.

– $RO(\overline{S}_0^{\widetilde{m}}, mlength) \to Dig$: RO is a random oracle, $\overline{S}_0^{\widetilde{m}}$ is the final state, and $mlength$ is the length of message $\underline{m}$. RO can be realized by iterating the round function T_{HASH} for enough number of rounds. In each round the message bit is replaced by $e \oplus v$, where e is the middle bit of the buffer and v is a corresponding bit of the stream $\underline{h}$ obtained by repeating $mlength$: $\underline{h} = ||^*[mlength]$, where $[mlength]$ is seen as a bit stream of length 64.

The security of this HASH is determined by the hardness of following problem: to find $m \neq m'$ of equal length satisfying $\overline{S}_0^{\widetilde{m}} = \overline{S}_0^{\widetilde{m'}}$. Generally, the problem is defined by systems of equations. This kind of systems of equations are too complex for existing solvers. The only known method for solving this type of equations is differential cryptanalysis.

We consider a special kind of Thin Sponge functions where the security of MAC is easy to estimate. Denote

$$\begin{aligned} \mathbb{G} \times \{0,1\} &\xrightarrow{H} \{0,1\} \\ H(G,m) &= SB(G) + m, \\ \mathbb{G} &\subseteq \mathbb{S} \end{aligned}$$

$$\begin{aligned} \mathbb{S} &\xrightarrow{A} \mathbb{S} \\ S &\mapsto AS. \end{aligned}$$

Our Thin Sponge is defined as $F(S,m) = AS + b\beta$, where A is a linear transformation over $\mathbb{S}$(considered as a linear space), SB is the S-box function taking several state bits to one output bit, where $\mathbb{G}$ denotes the subspace consisting of the input bits of the S-box, and β is the constant state determined by the positions where $b = H(G,m)$ embedded in.

The probability $\max\limits_{m \neq m'} Pr[S_0^{\underline{m}} = S_0^{\underline{m}'}]$ is determined by the number of active S-boxes of all valid differential trails $\underline{b} = \{b_i\}_{i \in \{0,1,2,\dots\}}$ such that $0^{\underline{b}} = \underline{0}$, where round function is just a linear function A, S-box is active if and only if its input G is non-zero. Generally, the number of active S-boxes is smallest when $\{b_i\}_{i \in \{0,1,2,\dots\}}$ is the coefficients of the minimal polynomial of A. In this case, state difference sequence is called the characteristic sequence and we denote $AG = |\{i : G_i \neq 0\}|$, where G_i is input difference of S-box in the i-th round. To give a MAC of λ-bit security, we just make $AG > \lambda$.

To give the GK function of the AE, we need another independent S-box $SB' : \mathbb{G} \longrightarrow \{0,1\}$, and define $GK(S) = SB'(G) + s$, where s is some bit of S.

A Concrete Construction

We give a MAC round function with 256-bit state for 128-bit security. The linear transformation A is defined by four 64-bit LFSRs, which can run at least 32 steps in parallel. The leftmost bit of every LFSR is simply updated by xoring three bits of its right half 32 bits and one bit from another register at each step so that 32 steps can be done in parallel. The four LFSRs are concatenated in a circle where one bit of one register affects its next register.

$\mathbb{G} = \{s^j_{j_0}, s^j_{j_1}\}_{j\in\{0,1,\dots,3\}} \subseteq \mathbb{S}$ and $SB(G) = 1 \oplus \bigoplus_{j=0}^{3} s^j_{j_0} \cdot s^j_{j_1}$, where $s^j_{j_0}$,$s^j_{j_1}$ are two bits from the j-th register. $SB' = \bigoplus_{i_0 \neq i_1 i_0=0}^{3} s^{i_0}_{t_0} \cdot s^{i_1}_{t_1}, t_i \in \{j_0, j_1\}$, it is independent to SB except at 3 input difference values.

The main imporvements over ACORN are the follows. The shift operation in software is easier on 64-bit registers. The concatenation of the registers has more choices to make the characteristic polynomial of A is irreducible. The S-boxes we used are lighter.

An example is given by replacing the indexes with specific numbers in above description, where the hardness of collision problems for MAC and HASH are supposed to be 148 and 192 bits respectively, and the efficiency of AE is slightly better than ACORN. HASH runs in half speed of AE.

To summarize, our main contributions are as follows:

– We give an improved construction of AE over ACORN.
– We give the construction from AE to Hash and heuristic criteria for security strength.
– According to the security criteria, we give an example, which achieves all functionalities with a single small hardware.

2 Notations

We use small letters for bits, underlined $'\underline{small\ \ letters}'$ for vector streams, sequences of bits etc. $|\cdot|$ for the number of elements in a set, $+$ for Xor, $\cdot$ for And.

For any round function of the form $T : \mathbb{S} \times \{0,1\} \longrightarrow \mathbb{S}$, we denote $S^{\underline{m}}$ to be the state obtained as follows:

$$S \longleftarrow T(S, m_i), \text{ for } i = 0, 1, \dots,$$

where T is omitted if the context is clear.

3 MAC and AE

In this section, we give explicit construction of our Thin Sponge function, and its security analysis.

3.1 Description

Suppose the security level is 128-bit, our Thin Sponge function acts on a 256-bit state $\mathbb{S}$.

– $\mathbb{S}$ consists of four 64-bit LFSR registers $\{\mathbb{S}^j\}_{j=0,1,2,3}$ as Fig.3 shows. Each register uses a linear feedback function $F_j, j \in \{0, 1, 2, 3\}$. The shift operation is easier on 64-bit register. We write $S^j = \dots, s^j_{-1}, s^j_{-2}, \dots, s^j_{-64}$.

– The concatenation of the registers is as follows: the four registers are concatenated to a circle where one bit of each register is xored to the input of the next register, where the extracted position of register j is denoted by u^j.
– $\mathbb{G} = \{s^j_{j_0}, s^j_{j_1}\}_{j\in\{0,1,\ldots,3\}} \subseteq \mathbb{S}$, and $SB(G) = 1 \oplus \bigoplus_{j=0}^{3} s^j_{j_0} \cdot s^j_{j_1}$, where $s^j_{j_0}$,$s^j_{j_1}$ are two bits of the j-th register $\mathbb{S}^j$. $SB^{'} = \bigoplus\limits_{i_0\neq i_1, i_0=0}^{3} s^{i_0}_{t_0} \cdot s^{i_1}_{t_1}, t_i \in \{j_0, j_1\}$.

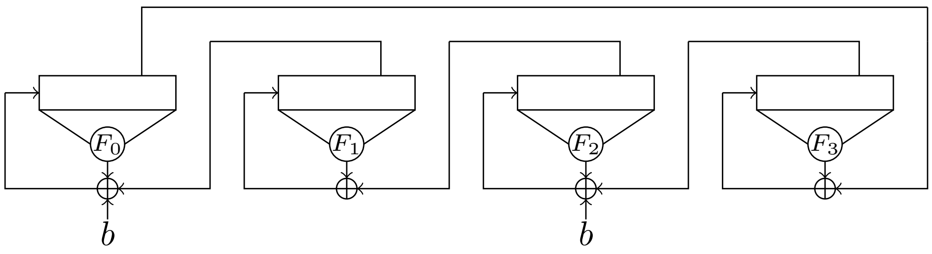


Fig. 3: T_{MAC}

In the following, we give the algebraic description of the algorithm. The state S can be seen as a vector of polynomials in t^{-1}, polynomials are seen as in the ring $\mathbb{F}_2[t^{-1}] \mod ((t^{-1})^{64})$ or written as $\mathbb{F}_2[t^{-1}]/((t^{-1})^{64})$. Each linear feedback function F_j has a connection polynomial, denoted by $f_j \in \mathbb{F}_2[t]$. For example, if $F_j(S^j_0) = s^j_{-32} \oplus s^j_{-47} \oplus s^j_{-64}$, then $f_j(t) = t^{32} + t^{47} + t^{64}$.

Then the linear transformation A is represented as a 4×4 matrix of polynomials in t as follows:

$$P = \begin{pmatrix} f_0 & t^{u_1} & 0 & 0 \\ 0 & f_1 & t^{u_2} & 0 \\ 0 & 0 & f_2 & t^{u_3} \\ t^{u_0} & 0 & 0 & f_3 \end{pmatrix}.$$

$$AS = t^{-1} \cdot S + [t^{-1} \cdot P \cdot S]_0. \tag{1}$$

where $[]_0$ denotes the constant term(of degree 0).

The MAC round function T_{MAC} is:

$$\begin{aligned} \mathbb{S} \times \{0,1\} &\longrightarrow \mathbb{S} \\ T_{MAC}(S,m) &= AS + (SB(G) + m)\beta, \end{aligned}$$

where

$$\beta = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}.$$

To do differential cryptanalysis, the content of j-th register is denoted as a power series

$$S^j = \sum_{i \geq -64} s_i^j \cdot t^i.$$

Denote S_i (S_i^j) be the i-th state (j-th register state), that is

$$S = \begin{pmatrix} S^0 \\ S^1 \\ S^2 \\ S^3 \end{pmatrix}, \tag{2}$$

and $S_i = [S]_{i-1,i-2,\ldots,i-64}$, i.e. terms of degree ranging from $i-1$ to $i-64$.

Input sequence is also denoted as

$$B = \sum_{i \geq 0} b_i \cdot t^i,$$

where $b_i = SB(G_i) + m_i, G_i = \{s_{i-j_1}^j, s_{i-j_2}^j\}_{j=0,1,2,3}$.

Then our MAC can be described as

$$P \cdot S + B \cdot \beta = S, \tag{3}$$

which is

$$(P + I) \cdot S = b \cdot \beta. \tag{4}$$

The round function in AE (initialization and finalization) is:

$$T_{AE} = AS + (SB(G) + m)\beta + GK(S)\beta^{'},$$

where

$$\beta^{'} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \end{pmatrix}.$$

3.2 Security Analysis

The security strength λ of this MAC is determined by $\delta = \max_{m \neq m^{'}} Pr[S_0^m = S_0^{m^{'}}]$. That is $\lambda = log_2(\frac{1}{\delta})$, which is supposed to be AG, the number of active S-boxes in the characteristic sequence.

From Eq.(4), S can be expressed as follows:

$$S = \frac{(P + I)^* \cdot \beta \cdot b}{\det(P + I)}, \tag{5}$$

where $(P+I)^*$ is the adjoint matrix of $(P+I)$, and $\det(P+I) = \prod_{j=0}^{3}(1+f_j) + t^{\sum_{j=0}^{3} u_j}$ is the characteristic polynomial of A. When $b = \det(P+I)$, we get $S = (P+I)^* \cdot \beta$, and

$$AG = |\{i : G_i \neq 0\}|,$$

where $G_i = \{s^j_{i-j_0}, s^j_{i-j_1}\}_{j\in\{0,1,\ldots,3\}}$

3.3 Deciding Parameters

We let f_j have exactly 3 terms, i.e. $f_j = t^{\epsilon^j_0} + t^{\epsilon^j_1} + t^{\epsilon^j_2}$. We choose all $\epsilon^j_0, \epsilon^j_1, \epsilon^j_2$ to make that the weight of characteristic sequence S is as large as we can achieve, and also the characteristic polynomial irreducible. Weight of the sequence is defined as the number of 1.

Then we chose G to be random positions and to make AG as large as possible.

For Example, if we let

- $f_0 = t^{64}+t^{33}+t^{32}, f_1 = t^{64}+t^{36}+t^{34}, f_2 = t^{64}+t^{42}+t^{37}, f_3 = t^{64}+t^{56}+t^{45}$. $u_0 = 33, u_1 = 51, u_2 = 63, u_3 = 38$.
- $G = \{s^0_{-34}, s^0_{-35}, s^1_{-32}, s^1_{-40}, s^2_{-33}, s^2_{-60}, s^3_{-49}, s^3_{-50}\}$.
- then we get $AG = |\{i : G_i \neq 0\}| = 148$.

4 HASH

4.1 Definition

The round function of HASH is as follows:

$$\mathbb{S} \times \mathbb{B} \times \{0,1\} \to \mathbb{S} \times \mathbb{B}$$

$$T_{HASH}(S, \underline{b}', \widetilde{m}) = (AS + b\beta + b'\beta', b' + GK(S) + t^{-1}\underline{b}'),$$

where $\mathbb{S}$ and $\mathbb{B}$ denote respectively the state space of registers and buffer. $\mathbb{B} = \mathbb{F}_2[t^{-1}]/((t^{-1})^{l_B}), b' = \sum_{i=0}^{l_B-1} b'_i t^{-i} \in \mathbb{B}, b = SB(G) + \widetilde{m}, b' = b'_{l_B-1}$.

Note that the register part of T_{HASH} is the same as T_{AE}, except that $GK(S)$ is replaced by $b'_{Blength-1}$, and it goes to the buffer state as key stream for encrypting buffer output, where the latter is an operation of the AE process.

We get the output of HASH by applying a random oracle RO:

$$RO(\overline{S}^{\widetilde{m}}_0, mlength) \to Dig,$$

where $mlength$ is the length of message stream. RO is realized by the iterating the round function $\overline{S} \leftarrow \overline{S}^{\widetilde{h}}$. At this stage, $\widetilde{m}$ is replaced by $\widetilde{h} = b'_{\frac{Blength}{2}-1} + h$, where h is provided by the stream $\underline{h}$ of repeating the message length.

Finally we output the newest n bits of the buffer as the digest of HASH, where n is the required length of HASH output.

In the message encoding phase, the original message stream m is divided into fixed-length blocks $M_0, M_1, M_2, \ldots$, and inserted some fixed-length constant blocks $C_0, C_1, C_2 \ldots$ to get the input string $\widetilde{m}$, i.e.

if $m = M_0||M_1||M_2|| \ldots$,

then $\widetilde{m} = M_0||C_0||M_1||C_1|| \ldots$.

In our example, we chose $C_i = \underline{0}$ for all i.

4.2 Security Analysis

The security of HASH is reduced to the problem of finding collisions: to find $m \neq m'$ so that $\overline{S}_0^{\widetilde{m}} = \overline{S}_0^{\widetilde{m'}}$. In general, the problem of finding collisions will be transformed into solving systems of equations. This kind of systems of 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. A differential trail is a sequence pair $(\underline{b}, \underline{b}')$, which is the differential sequence that appears in the expression of T_{HASH}. A full valid trail is the one that makes the whole state from $\underline{0}$ to $\underline{0}$. Note that $\underline{b}'$ is decided by $\underline{b}$ and the output of $SB^{'}$(which must be 0 for inactive S-boxes). At step i, b_i 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.

4.2.1 An Upper Bound of Security Strength

In our HASH, 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 $\underline{b}$ ($\underline{b}'$=0) is chosen as a small multiple of minimal polynomial of A. 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 is the size of the register state, and c_1 is a small constant. According to previous subsection, the cost of this sub-trail is about

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

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 ($\underline{b} = \underline{m}, \underline{b}' = \underline{m}'$), 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 HASH.

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. $\underline{b}' = \underline{0}$. The length of these trails are $l_B + c_2$, where c_2 is a constant (in our example, $c_2 > 64$). In any valid trail the inactive S-boxes' input forms a linear space, which is a subspace of the space generated by variables in $\underline{b}$, 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.

5 Example

5.1 MAC and AE

- A is defined by $f_0 = t^{64} + t^{33} + t^{32}, f_1 = t^{64} + t^{36} + t^{34}, f_2 = t^{64} + t^{42} + t^{37}, f_3 = t^{64} + t^{56} + t^{45}$, $u_0 = 33, u_1 = 51, u_2 = 63, u_3 = 38$.
- S-box position $G = \{s^0_{-34}, s^0_{-35}, s^1_{-32}, s^1_{-40}, s^2_{-33}, s^2_{-60}, s^3_{-49}, s^3_{-50}\}$.
- $SB(G) = 1 \oplus s^0_{-34} \cdot s^0_{-35} \oplus s^1_{-32} \cdot s^1_{-40} \oplus s^2_{-33} \cdot s^2_{-60} \oplus s^3_{-49} \cdot s^3_{-50}$,
  $SB'(G) = s^0_{-34} \cdot s^2_{-60} \oplus s^0_{-35} \cdot s^3_{-50} \oplus s^1_{-32} \cdot s^3_{-49} \oplus s^1_{-40} \cdot s^2_{-33}$.
- $GK(S_0) = s^0_{-32} + SB'(G)$ with $v = s^0_{-32}$.
- Security strength for MAC state collision $AG = 148$.

Initialization: The round function is

$$S \longleftarrow AS + H(G, m) \cdot \beta + GK(S) \cdot \beta', \tag{6}$$

where

$$\beta' = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \end{pmatrix}.$$

At first, load the 128-bit key in right half of the four registers, and load constants in left half to get S, then

$$S \longleftarrow S^{IV},$$

$$S \longleftarrow S^{\underline{0}_{512}}.$$

$\underline{0}_{512}$ stands for 0-stream of length 512.

Processing Message:

$$S \leftarrow S^{\underline{m}},$$

where round function is

$$S \leftarrow AS + H(G, m)\beta.$$

Ciphertext: $GK(S) + m$

Finalization:

$$S \leftarrow S^{\underline{0}_{512}},$$

where the round function is the same as in the initiation stage. Then run the encryption stage for message $\underline{0}_{128}$ and Tag is the cipher text stream.

5.2 HASH

- $l_B = 1024, l_m = l_c = 32, \rho = \frac{1}{2}$. In this condition, the hardness of state collision is supposed to be 192 bits.
- In the initialization process, load the registers and buffer with fixed random constants, which is generated by a pseudorandom number generator such as B.B.S.[19] under the seed $\underline{0}$.
- In the RO stage, the number of rounds is set to be 1536.

5.3 Hardware

The hardware part could only treat 3 round functions MAC, AE initiation, register part of HASH and control logic for shifting between the 3-modes. These 3 round functions share almost the same circuit, so we can implement all these functionalities with a single small hardware.

References

1. Bellare, M., Kilian, J., Rogaway, P.: The security of the cipher block chaining message authentication code. Journal of Computer and System Sciences **61**(3) (2000) 362–399
2. Iwata, T., Kurosawa, K.: OMAC: one-key CBC MAC. In: Fast Software Encryption, 10th International Workshop, FSE 2003, Lund, Sweden, February 24-26, 2003, Revised Papers. (2003) 129–153
3. Black, J., Rogaway, P.: A block-cipher mode of operation for parallelizable message authentication. In: Advances in Cryptology - EUROCRYPT 2002, International Conference on the Theory and Applications of Cryptographic Techniques, Amsterdam, The Netherlands, April 28 - May 2, 2002, Proceedings. (2002) 384–397

4. Dworkin, M.: SP 800-38C. Recommendation for Block Cipher Modes of Operation: the CCM Mode for Authentication and Confidentiality. National Institute of Standards and Technology (2005)
5. Bellare, M., Rogaway, P., Wagner, D.A.: The EAX mode of operation. **3017** (2004) 389–407
6. McGrew, D.A., Viega, J.: The security and performance of the galois/counte mode of operation (full version). IACR Cryptology ePrint Archive **2004** (2004) 193
7. Krovetz, T., Rogaway, P.: The software performance of authenticated-encryption modes. In: FAST Software Encryption - International Workshop, FSE 2011, Lyngby, Denmark, February 13-16, 2011, Revised Selected Papers. (2011) 306–327
8. Bellare, M., Canetti, R., Krawczyk, H.: Keying hash functions for message authentication. In: Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings. (1996) 1–15
9. Alizadeh, J., Aref, M.R., Bagheri, N.: JHAE: A novel permutation-based authenticated encryption mode based on the hash mode JH. Cryptology ePrint Archive, Report 2014/193 (2014) `https://eprint.iacr.org/2014/193`.
10. Wu, H.: Acorn: A lightweight authenticated cipher (v3). (2016) `https://competitions.cr.yp.to/round3/acornv3.pdf`.
11. NIST: Announcing the standard for secure hash standard. (1995)
12. NIST: Announcing the standard for secure hash standard. (2002)
13. Rivest, R.L.: The MD5 message-digest algorithm. RFC **1321** (1992) 1–21
14. BERTONI, G., DAEMEN, J., PEETERS, M., ASSCHE, G.V.: The keccak sha-3 submission. (2011)
15. Guo, J., Peyrin, T., Poschmann, A.: The PHOTON family of lightweight hash functions. IACR Cryptology ePrint Archive **2011** (2011) 609
16. Aumasson, J., Henzen, L., Meier, W., Naya-Plasencia, M.: Quark: A lightweight hash. In: Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings. (2010) 1–15
17. Bogdanov, A., Knezevic, M., Leander, G., Toz, D., Varici, K., Verbauwhede, I.: spongent: A lightweight hash function. In: Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings. (2011) 312–325
18. Berger, T.P., D'Hayer, J., Marquet, K., Minier, M., Thomas, G.: The GLUON family: A lightweight hash function family based on fcsrs. In: Progress in Cryptology - AFRICACRYPT 2012 - 5th International Conference on Cryptology in Africa, Ifrance, Morocco, July 10-12, 2012. Proceedings. (2012) 306–323
19. Blum, L., Blum, M., Shub, M.: A simple unpredictable pseudo-random number generator. SIAM J. Comput. **15**(2) (1986) 364–383