ISSN(Online) : 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Website: www.ijirset.com

Vol. 6, Issue 4, April 2017

Spongent Implementation Using ModelSimSE

Ansy V Muhammed[1], Geethu T George[2]

P.G. Student, Department of Electronics and Communication Engineering, Indira Gandhi Institute of Engineering and Technology for Women, Nellikuzhy, Kerala, India[1]

Associate Professor, Department of Electronics and Communication Engineering, Indira Gandhi Institute of Engineering and Technology for Women, Nellikuzhy, Kerala, India[2]

ABSTRACT:One of the fundamental problems of cryptography is designing secure yet efficiently implementable cryptographic algorithms. Lately, lightweight cryptography optimizing the algorithms to fit the most constrained environments has received a great deal of attention, the recent research being mainly focused on building block ciphers. As opposed to that, the design of lightweight hash functions is still far from being well-investigated with only few proposals in the public domain. In this article, we aim to address this gap by exploring the design space of lightweight hash functions based on the sponge construction instantiated with present-type permutations. The resulting family of hash functions is called spongent. We also perform security analysis in terms of differential properties, linear distinguishers, and rebound attacks.

KEYWORDS:hash function, lightweight cryptography, sponge construction, present, spongent.

I. INTRODUCTION

Dueto the advancement of pervasive computing, the need for the security in RFID and sensor networks is tremendously increasing which require efficiently implementable cryptographic primitives likes hash function. In a constrained environment area and power consumption are becoming important issues and the implementation of standard algorithm is expensive. So go for lightweight cryptography. Once this problem was identified, the cryptographic community designed a number of lightweight algorithms to address this challenge. They are HIGHT, DESL and PRESENT etc. This paper reveals the design space of lightweight hash functions based on sponge construction instantiated with PRESENT type permutation. The resulting family of such hash function is called SPONGENT.

Basically a Hash function is a function which maps an arbitrary lengthy message into a fixed length output. The footprint of hash function is determined by the number of state bits as well as the size of the functional and control logic used in a round function. The main properties of hash functions are fixed length output, preimage resistance, second preimage resistance and collision resistance. Preimage resistance means one could not be capable of recovering the original message from its hash value. Whereas second preimage resistance or weak collision resistant means that given one message, can't find another message that has the same message digest And collision resistant or strong collision resistance means that can't find any two different messages with the same message digest.

II. RELATED WORK

A. *Theoretical Investigations*

A hash function is any function that can be used to map data of arbitrary size to data of fixed size. The values returned by ahash function are called hash values, hash codes, hash sums, or simply hashes. One use is a data structure called a hash table, widely used in computer software for rapid data lookup. Hash functions accelerate table or database lookup by detecting duplicated records in a large file. An example is finding similar stretches in DNA sequences. They are also useful in cryptography. A cryptographic hash function allows one to easily verify that some input data maps to a given hash value, but if the input data is unknown; it is deliberately difficult to reconstruct it (or equivalent alternatives) by

ISSN(Online) : 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Website: www.ijirset.com

Vol. 6, Issue 4, April 2017

knowing the stored hash value. This is used for assuring integrity of transmitted data, and is the building block for HMACs, which provide message authentication.Hashfunctions are related to checksums, check digits, fingerprints, randomization functions, error-correcting codes, and ciphers. Although these concepts overlap to some extent, each has its own uses and requirements and is designed and optimized differently. The Hash Keeper database maintained by the American National Drug Intelligence Center, for instance, is more aptly described as a catalogue of file fingerprints than of hash values.

B. *Security Applications*

The most prominent security applications targeted by a hash function are Lightweight signature schemes, RFID security protocols, Random number generation etc. Also hash algorithm can be used to verify the data integrity. After receiving the data one can compute the hash of the original data and can compare it with the original one that comes from a trusted source. If they are same it results in high confidence level. It means that message was not modified during the transmission. Indeed, hash functions are quite useful in determining whether any alteration has been made to message. In general, a hash function $h : \{0, 1\}^* \rightarrow \{0, 1\}^n$ maps strings of arbitrary length to strings of fixed length. A cryptographic hash function should behave like a random oracle. For every query a random oracle returns a truly random output from its output domain, with the exception that it always returns the same output for the same query. The term "a function should behave like a random oracle" means that the function should not have any weaknesses. If an algorithm is found that demonstrates that any of these properties can be broken more efficiently than for a random oracle, the hash function is considered broken, even if this algorithm is practically infeasible.

1. **Preimage Resistance:** A hash function $h : \{0, 1\}^* \rightarrow \{0, 1\}^n$ is preimage resistant, if for any given value $t \in \{0, 1\}^n$ it is computationally infeasible to find a preimage $m \in \{0, 1\}^*$ with $h(m) = t$ with non-negligible probability. A brute-force attack on a random oracle has an expected time complexity of 2^{n-1}.
2. **Second Preimage Resistance:** A hash function $h : \{0, 1\}* \rightarrow \{0, 1\}^n$ is second preimage resistant, if for any given value $m1 \in \{0, 1\}^*$it is computationally infeasible to find a second preimage $m_2 \neq m_1 \in \{0, 1\}^*$ with $h(m_1) = h(m_2)$ with non-negligible probability. A brute-force attack on a random oracle also has an expected time complexity of 2^{n-1}.
3. **Collision Resistance:** A hash function $h : \{0, 1\}^* \rightarrow \{0, 1\}^n$ is collision resistant, if it is computationally infeasible to find $m1, m2 \in \{0, 1\}^*$ with $m_1 \neq m_2$ and $h(m_1) = h(m_2)$ with non-egligible probability. A brute-force attack on a random oracle has an expected time complexity of $2^{n/2}$.

Most hash functions use a finite internal state to compute the output to a certain input. Usually this state is initialized with a fixed value, I V, given in the specification. For cryptanalytic purposes it is often of interest to explore the possibility of finding a collision for some arbitrary initial value of the internal state that is fixed across different applications. If such a collision is found it is usually denoted as semi-freestart collision.

4. **Semi-free-start collision:** For a hash function hI V: $\{0, 1\}^* \rightarrow \{0, 1\}^n$ using an inner state initialized with IV, the values IV', m_1, and m_2 constitute a semi-free-start collision if $h_{I V'}(m_1) = h_{I V'}(m_2)$ and $m_1 \neq m_2$.

The availability of hash functions that have the above properties is of significant importance in the field of cryptography. The properties are relied upon in numerous protocol designs and the corresponding security proofs.

C. *Sponge Function*

If In the context of cryptography, sponge functions provide a particular way to generalize hash functions to more general functions whose output length is arbitrary. A sponge function instantiates the sponge construction, which is a simple iterated construction building a variable-length input variable-length output function based on a fixed length permutation. With this interface, a sponge function can also be used as a stream cipher, hence covering a wide range of functionality with hash functions and stream ciphers as particular points.

From a theoretical point of view, sponge functions model in a very simple way the finite memory any concrete construction has access to. A random sponge function is as strong as a random oracle, except for the effects induced by the finite memory. This model can thus be used as an alternative to the random oracle model for expressing security claims.

ISSN(Online) : 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Website: www.ijirset.com

Vol. 6, Issue 4, April 2017

Sponge Construction: The sponge construction is a simple iterated construction for building a function F with variable-length input and arbitrary output length based on a fixed-length transformation or permutation f operating on a fixed number b of bits. Here b is called the width. The sponge construction operates on a state of b = r + c bits. The value r is called the bit rate and the value c the capacity. First, all the bits of the state are initialized to zero. The input message is padded and cut into blocks of r bits. The sponge construction then proceeds in two phases: the absorbing phase followed by the squeezing phase.

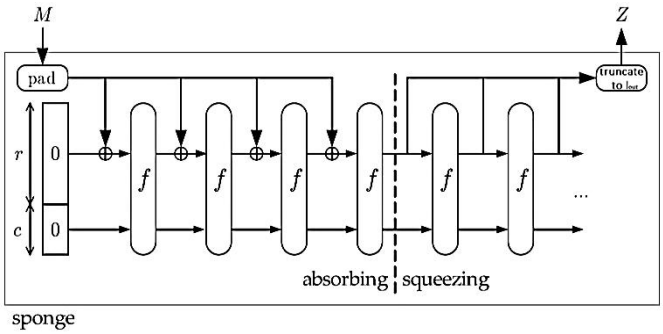

Fig.1 Sponge construction

1. In the absorbing phase, the r-bit input message blocks are XORed into the first r bits of the state, interleaved with applications of the function f . When all message blocks are processed, the sponge construction switches to the squeezing phase.
2. In the squeezing phase, the first r bits of the state are returned as output blocks, interleaved with applications of the function f. The number of output blocks is chosen at will by the user. The last c bits of the state are never directly affected by the input blocks and are never output during the squeezing phase.

D. *Present Permutation*

Present is a hardware-oriented lightweight block cipher designed to meet tight area and power restrictions. It features a 64-bit block size and 80-bit key size. Present implements a substitution-permutation network in 32 rounds. Each of the 31 rounds consists of a xor operation to introduce a round key Ki for $1 \leq i \leq 32$, where K32 is used for post-whitening, a linear bitwise permutation and a non-linear substitution layer. The permutation layer consists only of bit permutations. Together with the tiny 4-bit S-box, the design enables minimalistic hardware implementations.

The key scheduling consists of a single S-box lookup, a counter addition and a rotation.Present is an example of an SP-network and consists of 31 rounds. The block length is 64 bits and two key lengths of 80 and 128 bits are supported. Given the applications we have in mind, we recommend the version with 80-bit keys. This is more than adequate security for the low-security applications typically required in tag-based deployments, but just as importantly, this matches the design goals of hardware-oriented stream ciphers in the eSTREAM project and allows us to make a fairer comparison. Each of the 31 rounds consists of anxor operation to introduce a round key Ki for $1 \leq i \leq 32$, where K32 is used for post-whitening, a linear bitwise permutation and a non-linear substitution layer. The non-linear layer uses a single 4-bit S-box S which is applied 16 times in parallel in each round. The cipher is described in pseudo-code in Figure 1, and each stage is now specified in turn.

S-Box Layer: We use a single 4-bit to 4-bit S-box S: $F_2^4 \rightarrow F_2^4$ in present. The action of the S-box in hexadecimal notation is given by the following figure.

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S[x]$	E	D	B	0	2	1	4	F	7	A	8	5	9	C	3	6

Fig. 2: S-Box Layer

ISSN(Online) : 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Website: www.ijirset.com

Vol. 6, Issue 4, April 2017

This is a direct consequence of our pursuit of hardware efficiency, with the implementation of such an S-box typically being much more compact than that of an 8-bit S-box. Since we use a bit permutation for the linear diffusion layer, AES-like diffusion techniques are not an option for present. Therefore we place some additional conditions on the S-boxes to improve the so-called avalanche of change.

Permutation Layer: Present is a hardware-oriented lightweight block cipher designed to meet tight area and power restrictions. It features a 64-bit block size and 80-bit key size.

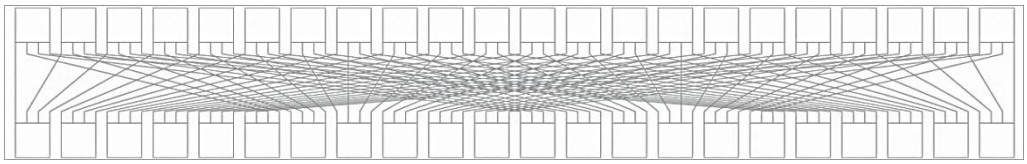

Fig.3: Permutation Layer

Present implements a substitution-permutation network in 32 rounds. Each of the 31 rounds consists of anxor operation to introduce a round key Ki for $1 \leq i \leq 32$, where K32 is used for post-whitening, a linear bitwise permutation and a non-linear substitution layer. The permutation layer consists only of bit permutations.When choosing the mixing layer, our focus on hardware efficiency demands a linear layer that can be implemented with a minimum number of processing elements, i.e. transistors. This leads us directly to bit permutations. Given our focus on simplicity, we have chosen a regular bit-permutation and this helps to make a clear security analysis.

III. SPONGENT

The spongent family is a family of hash functions designed using the hermetic sponge construction. The parameters *b*, *r*, *n*, and *c* differ for each instantiation of the family, but they all employ a very similar permutation, which we will denote by π_b. spongent is a sponge construction based on a wide present-type permutation. Given a finite number of input bits, it produces an n-bit hash value. A design goal for spongent is to follow the hermetic sponge strategy.The overall design approach for spongent is to target low area while favoring simplicity. The 4-bit S-box is the major block of functional logic in a serial low-area implementation of spongent. It fulfills the present design criteria in terms of deferential and linear properties. Moreover, any linear approximation over the S-box involving only single bits both in the input and output masks is unbiased. This aims to restrict the linear hull effect discovered in round-reduced present.The function of the bit permutation pLayer is to provide good diffusion, by acting together with the S-box, while having a limited impact on the area requirements. This is its main design goal, while a bit permutation may occupy additional space in silicon. The counters lCounter and reverse lCounter are mainly aimed to prevent sliding properties and make prospective cryptanalysis approaches using properties like invariant subspaces more involving.

Permutation-Based Sponge Construction: Spongent relies on a sponge construction a simple iterated design that takes a variable-length input and can produce an output of an arbitrary length based on a permutation π_b operating on a state of a fixed number b of bits. The size of the internal state b = r + c _ n is called width, where r is the rate and c the capacity. The sponge construction proceeds in three phases, as shown in fig.4.

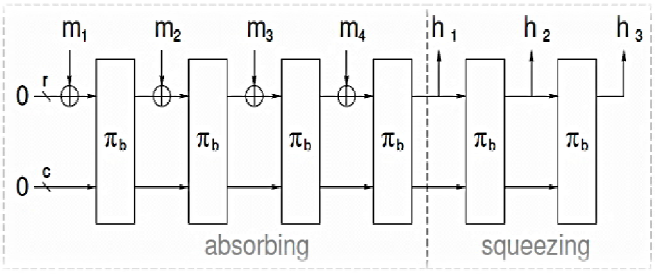

Fig.4 : Sponge Construction

ISSN(Online) : 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Website: www.ijirset.com

Vol. 6, Issue 4, April 2017

1. Initialization phase: the message is padded by a single bit 1 followed by a necessary number of 0 bits up to a multiple of r bits (e.g., if r = 8, then the 1-bit message `0' is transformed to `01000000'). Then it is cut into blocks of r bits.
2. Absorbing phase: the r-bit input message blocks are xored into the first r bits of the state, interleaved with applications of the permutation π_b.
3. Squeezing phase: the first r bits of the state are returned as output, interleaved with applications of the permutation π_b, until n bits are returned.

In spongent, the b-bit 0 is taken as the initial value before the absorbing phase. In all spongent variants, except spongent-88/80/8, the hash size n equals either capacity c or 2c. The message chunks are xored into the r rightmost bit positions of the state. The same r bit positions form parts of the hash output.Let a permutation-based sponge construction have $n \geq c$ and $c/2 > r$ which is fulfilled for the parameter choices of most of the spongent variants. Then the works imply the preimage security of 2^{n-r} as well as the second preimage and collision securities of $2^{c/2}$ if this construction is hermetic (ie, if the underlying permutation does not have any structural distinguishers). The best preimage attack we are aware of in this case has a computational complexity of $2_{n-r} + 2^{c/2}$. Later, this work is extended and preimage security is defined more generalized form: $\min(2^{\min(n,c+r)}; \max(2^{\min(n-r,c)}; 2^{c/2}))$.
For permutation-based sponge constructions with $n < c$ and $c/2 \leq r$ such as the remaining spongent variants, it follows from the same works that the second preimage security is 2^n and collision security is $2^{c/2}$. The previous preimage attack also works for this case hence we claim that the preimage security is $\min(2^n; \max(2^{n-r}; 2^{c/2}))$ since $n - r < c$.

Present-Type Permutation

for $i = 1$ to R do
  STATE $\leftarrow$ reverseOf$\text{lCounter}_b(i) \oplus$ STATE $\oplus \text{lCounter}_b(i)$
  STATE $\leftarrow \text{sBoxLayer}_b$(STATE)
  STATE $\leftarrow \text{pLayer}_b$(STATE)
end for

Fig.5: Present Permutation

The permutation π_b: $F_b^2 \rightarrow F_b^2$ is an R-round transform of the input state of b bits that can be outlined at a top-level as shown in figure 5.2. Present is a hardware-oriented lightweight block cipher designed to meet tight area and power restrictions. It features a 64-bit block size and 80-bit key size. Present implements a substitution-permutation network in 32 rounds.WheresBoxLayerb and pLayerb describe how the state evolves. For ease of design, only widths b with 4jb are allowed. The number R of rounds depends on block size b and can be found in lCounterb(i) is the state of an LFSR dependent on b at time i which yields the round constant in round i and is added to the rightmost bits of state. Reverse lCounter b(i) is the value of lCounterb(i) with its bits in reversed order and is added to the leftmost bits of state. The following building blocks are generalizations of the present structure to larger b-bit widths:

sBoxLayerb: This denotes the use of a 4-bit to 4-bit S-box S : $F_2^4 \rightarrow F_2^4$which is applied b/4 times in parallel. The action of the S-box in hexadecimal notation is given by the following figure:

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S[x]$	E	D	B	0	2	1	4	F	7	A	8	5	9	C	3	6

Fig.6: Spongent S-Box (hex)

pLayerb: This function simply permutes the bits of the state. This is an extension of the (inverse) present bit-permutation, for this the j-th bit is moved to bit P_b(j) as shown in figure,

ISSN(Online) : 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Website: www.ijirset.com

Vol. 6, Issue 4, April 2017

$$P_b(j) = \begin{cases} j \cdot b/4 \mod b-1, & \text{if } j \in \{0,\ldots,b-2\} \\ b-1, & \text{if } j = b-1. \end{cases}$$

Fig.7: The bit permutation layer of spongent

lCounterb: This is one of the four [dlog$_2$ R] bit LFSRs. The LFSR is clocked once every time its state has been used and its final value is all ones. If ζ is the root of unity in the corresponding binary finite field, the n-bit LFRSs defined by the polynomials given below are used for the spongent variants.

LFSR size (bit)	Primitive Polynomial
6	$\zeta^6 + \zeta^5 + 1$
7	$\zeta^7 + \zeta + 1$
8	$\zeta^8 + \zeta^4 + \zeta^3 + \zeta^2 + 1$
9	$\zeta^9 + \zeta^4 + 1$

Fig.8: n-bit LFRSs defined by the polynomials

IV. SECURITY ANALYSIS

In this section, we discuss the security of spongent against known cryptanalytic attacks by applying the most important state-of-the-art methods of cryptanalysis and investigating their complexity.

A. Differential Cryptanalysis

Here we analyze the resistance of spongent against differential attacks. The similarities of the spongent permutations and the basic present cipher allow reusing some of the results obtained for present.By examining the substitution and linear layers, one can make the following observations:

1. The S-box of spongent is such that a difference in single input bit causes a difference in at least two outputs bits or vice versa.
2. The input bits to an S-box come from four distinct S-boxes of the same subgroup.
3. The input bits to a subgroup of four S-boxes come from 16 distinct S-boxes of the same group.
4. The input bits to a group of 16 S-boxes come from 64 different S-boxes.
5. The four output bits from a particular S-box enter four distinct S-boxes, each of which belongs to a distinct group of S-boxes in the subsequent round.
6. The output bits of S-boxes in distinct groups go to distinct S-boxes in distinct subgroups.
7. The output bits of S-boxes in distinct subgroups go to distinct S-boxes.

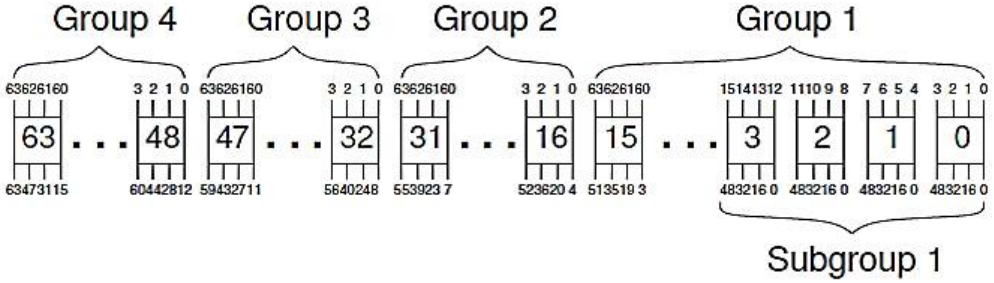

Fig.9: The grouping and subgrouping of S-boxes for b = 256.

ISSN(Online) : 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Website: www.ijirset.com

Vol. 6, Issue 4, April 2017

B. Collision Attacks

A natural approach to obtain a collision for a sponge construction is to inject a difference in a message block and then cancel the propagated difference by a difference in the next message block, i.e., $(0 \dots 0 \| \Delta_{mi}) \rightarrow (0 \dots 0 \| \Delta_{mi+1})$. For this purpose, we follow a narrow trail strategy using truncated differential characteristics. We start from a given input differenceand look for all paths that go to a fixed output difference. Based on our experiments, even by using truncated differential characteristics, the probability of such a path is quite low and it is not possible to attack the full number of rounds.

C. Rebound Attack

The rebound attack, a recent technique for cryptanalysis of hash functions, is applicable to both block cipher based and permutation based hash constructions. It consists of two main steps: the inbound phase where the freedom is used to connect the middle rounds by using the match-in-the-middle technique and the outbound phase where the connected truncated differentials are calculated in both forward and backward directions. It has been mostly used to improve the results on AES-based but it has also been successfully applied to similar permutations. Compared to the other algorithms the rebound attack has been successfully applied to, the design of spongent imposes some limitations. First of all, since the permutation is bit-oriented, and not byte-oriented, it might be non-trivial to find the path followed by a given input difference and to determine the number of active S-boxes after several rounds. This is mainly due to the difference propagation that strongly depends on the values of the passive part of the state. Moreover, the probability that two inbound phases match requires more detailed analysis. Below we attempt to develop rebound attacks on several spongent variants. Rebound analysis applies similarly to the remaining variants.

D. Preimage Resistance

Here we apply a meet-in-the-middle approach to obtain preimages on spongent. The attack has two main steps: pre-computation and matching phase. Complexity of the attack is dominated by pre-computation phase. Since the hash size is n bits, and the data is extracted in r bit chunks, there exist n/r rounds in the squeezing phase. This is mainly due to the difference propagation that strongly depends on the values of the passive part of the state. To be able to compute the data backwards in the absorbing phase, we need to know not only h_i's but also d_i values to obtain the input value of the permutationπ_b, where h_i denotes the part of the hash value and d_i is the concatenated part to h_i. The figure below shows the diagram for meet in middle attack.

The algorithm is as follows:

1. Pre-computation: $\pi^{-1}(h_{i+1}; d_{i+1}) = (h_i; d_i)$ for each i in the squeezing phase. Since h_i (r-bits) is already fixed, the probability of finding such d_i is 2^{-r}. Therefore, we start with $2^{((n/r)-1)*r)} = 2^{n-r}$ different $d_{n/r}$ values to have a solution for d_1.

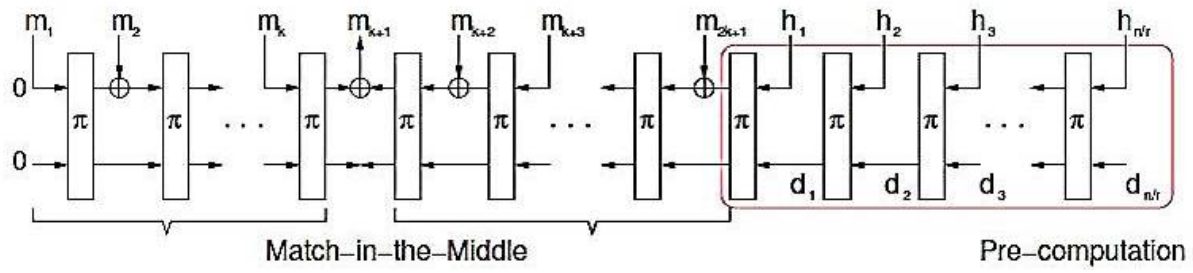

Fig.10: Meet-in-the-middle attack against sponge construction.

2. Match-in-the-middle: Choose k such that $k .r \geq c/2$.
   a. Then Generate $2^{c/2}$ elements in the backward direction by using $(h_1; d_1)$ and possible values for $m_{k+2} \dots m_{2k+1}$ and store them in a table.
   b. Generate $2^{c/2}$ elements in the forward direction by using possible values for $m_1 \dots m_k$ and b compare with list in the previous step to find a match of c bits in the middle.

ISSN(Online) : 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Website: www.ijirset.com

Vol. 6, Issue 4, April 2017

c. Obtain m_{k+1} by xoring the r bits for the matching elements.

E. Linear Attacks

The most successful attacks, the attacks that can break the highest number of rounds, for the block cipher present are those based on linear approximations. In particular the multidimensional linear attack and the statistical saturation attack claim to break up to 26 rounds. It was shown that both attacks are closely related. Moreover, the main reason why these attacks are the most successful attacks on present so far is the existence of many linear trails with only one active Sbox in each round. It is not immediately clear how linear distinguishers on the spongent permutation π_b could be transferred into collision or (second) pre-image attacks on the hash function.

However, as we claim that spongent is a hermetic sponge construction, the existence of such distinguishers has to be excluded. So the spongent S-box was chosen in a way that allows for at most one trail with this property given a linear approximation. Unlike for the block cipher present, where the key determines the actual linear correlation between an input and an output mask, for the permutation π_b we can compute the actual linear trail contribution for all trails with only one active S-box in every round. Each such trail over *W* rounds has a correlation of $\pm 2^{-2}$ and for each trail determining the sign is easy.

V. EXPERIMENTAL RESULTS

Figures shows the simulation result of spongent 88/80/8 in ModelSimSE

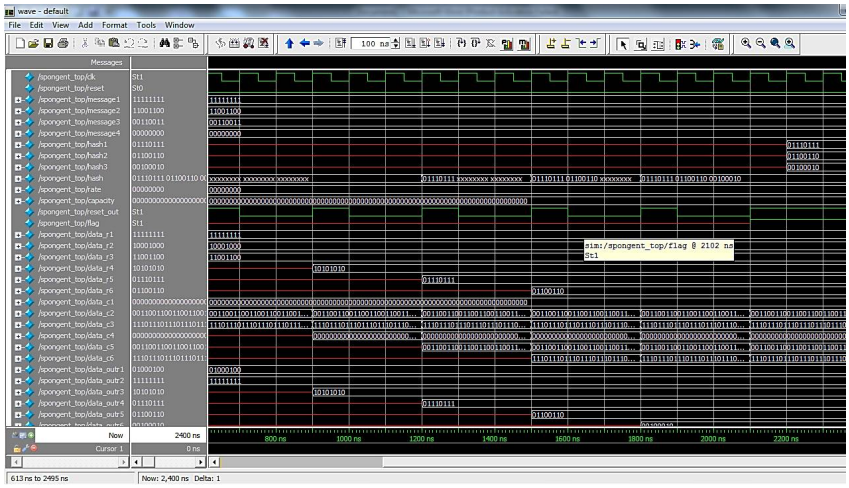

Figure 5.1: ModelSimSE Output

The process in ModelsimSE is summarized as follows:

- In the first stage the inputs are loaded in the first clock cycle.
- After executing the program for nearly 10 clock cycle the absorbing state completed.
- In the next clock cycle squeezing state started.
- After executing the complete program the flag is set in the next clock cycle.
- The hash values are outedin next clock cycle.

VI. CONCLUSION

In this work, we have explored the design space of lightweight cryptographic hashing by proposing the family of new hash functions spongent tailored for resource-constrained applications. While for multiple block ciphers, designs

ISSN(Online) : 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Website: www.ijirset.com

Vol. 6, Issue 4, April 2017

have already closely approached the minimum ASIC hardware footprint theoretically attainable, it does not seem the case for some recent lightweight hash functions so far. We proposed the family of sponge-based lightweight hash functions spongent with a smaller footprint than most existing dedicated lightweight hash functions: present in hashing modes and Quark. Its area is comparable to that of photon, though most of the time being slightly more compact. However, a fair comparison in terms of area requirements is a challenging task, since the area occupation is highly dependent on the implementation, technology and tools used. We also perform security analysis in terms of differential properties, linear distinguishers, and rebound attacks.

REFERENCES

[1] Avoine G Oechslin :A Scalable and Provably Secure Hash-Based RFID Protocol. In: PerCom Workshops. IEEE Computer Society ,pp. 110-114, 2005.
[2] Babbage S Dodd M.: The MICKEY Stream Ciphers. In: Robshaw and Billet, pp. 191-209,2008.
[3] Badel S Dagtekin, N Nakahara, J Oua_, P. Susil, P.Vaudenay,S.: ARMADILLO: A Multi-purpose Cryptographic Primitive Dedicated to Hardware. In: Mangard and Standaert , pp. 398-412,2010.
[4] Barreto, P.S.L.M., Rijmen, V.: The Whirlpool hashing function. In: Proceedings of the 1st NESSIE Workshop.pp. 15. Leuven,B (2000).
[5] Bertoni G., Daemen J., Peeters M., Van Assche G.: On the Indi_erentiability of the Sponge Construction. In: Smart, N.P. (ed.) EUROCRYPT'08. LNCS, vol. 4965, pp. 181-197, 2013.
[6] Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Sponge-Based Pseudo-Random Number Generators. In: Mangard and Standaert , pp. 33-47, 2010.