Spongy-Gift: A New Lightweight Message Authentication Code

Cuauhtemoc Mancillas López*, Mridul Nandi†
*Hubert Curien Laboratory UMR CNRS 5516, University of Lyon at Saint Etienne
cuauhtemoc.mancillas.lopez@univ-st-etienne.fr
†Indian Statistical Institute, Kolkata India
mridul@isical.in

***Abstract*—In this work we introduce Spongy-Gift, a Message Authentication Code based on a sponge function that use Gift-cipher round function to construct its public permutation. Some implementation results on FPGAs are presented to show Sponge-Gift needs very few resources to be implemented.**

I. Introduction

Actually with the introduction of Internet of Things (*IoT*) technology, where devices that we used in our normal life could be connected to internet to receive or send information, in this scenario security is a very important issue. For example a medical device that monitoring the status of the glucose in the blood of some patient and send such status to the doctor, if some attacker can change such messages could result in a bad diagnosis. In this example secrecy of the information is not require, but assure that the doctor receives the correct message generated by the device is mandatory. For this important task we can use a Message Authentication Code (*MAC*) .

(*MACs*) are symmetric key algorithms which provides data authentication and data integrity between two parties. The first step is to establish a common key K, if one entity wants to send a message m then it needs to compute $T \leftarrow MAC_K(M)$, in the insecure channel will travel the pair (M, T). To verity the received message (M, T) the receiver entity needs to compute $T' \leftarrow MAC_K(M)$ and then check if $T = T'$ then the message is valid in other way the message is rejected. If the verification fails two possible reasons, the key is not same (this means a not authorized entity tries to communicate with us failing the authentication) and the message was modified (data integrity fails).

In this paper we introduce a new MAC that is suitable for constrained environments in memory, energy or physical resources. First we explain in general our proposal, then a specific instantiation is detailed and finally we show some results for its implementation on FPGAs.

II. Spongy Gift

A. Preliminaries

Let M a message, to be processed it is splitted as $(M_1^{(1)}, M_2^{(2)}), ..., (M_\ell^{(1)}, M_\ell^{(2)})$ where $|M_i^{(1)}| = |M_i^{(2)}| = n$ and $|M_i^{(1)}| + |M_i^{(2)}| = 2n$ for $i = 1...\ell$. We will call as blocks all pairs $(M_i^{(1)}, M_i^{(2)})$ of 2n-bit size. A message authentication code is a function $\mathcal{IV} \times \mathcal{K} \times \mathcal{M} \rightarrow \mathcal{T}$ where $\mathcal{IV}, \mathcal{K}, \mathcal{M}, \mathcal{T}$ are IV space, key space, message space and tag space respectively. Informally a MAC is secure when it is hard for an adversary to generate a pair of message tag pass the verification test, also there is a reduction of a MAC as psudorandom function.

B. Definition of a Public Permutation

The basic idea behind our new construction if that if we have a $n-$bit permutation of with nice properties we can construct other permutation of $2n-$blocks of message using a few additional components, just one xor at its input Figure.

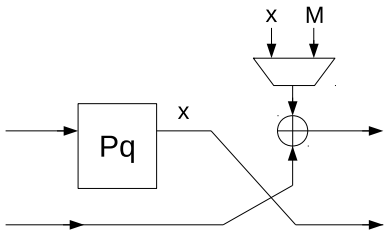

Fig. 1. Basic structure to construct a $2n-$bit permutation from an $n-$bit one.

In Figure 1 P_q is an $n-$bit permutation iterated q times, this structure could be used as a modified Feistel network to construct the secure $2n-$bit permutation as is shown in Figure 2.

The overhead to convert the basic structure in Figure 1 to the complete $2n-$ bit permutation $\overline{P}$ is very low, just some additional inputs to the multiplexer and additional $\oplus$. This will be clarify in implementation section.

Using $\overline{P}$ we can compute a MAC as follows:

1) $S_{top}^{(0)}, S_{bot}^{(0)} \leftarrow 0^n$
2) $M_0^{(1)} || M_0^{(2)} \leftarrow K || IV$
3) **for** $i = 0$ **to** ℓ **do**
4) $\quad w_1 \leftarrow P_q(M_i^{(1)} \oplus S_{bot}^{(i)})$
5) $\quad w_2 \leftarrow P_q(M_i^{(2)} \oplus S_{top}^{(i)})$
6) $\quad$ **if** $i = \ell$ **then** $k = 7$ **else** $k = 6$
7) $\quad$ **for** $j = 3$ **to** k **do**
8) $\quad\quad w_j \leftarrow P_q(w_{j-2} \oplus w_{j-1})$
9) $\quad$ **return** w_7 as tag

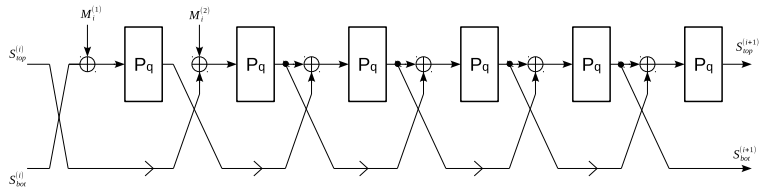

Fig. 2. Complete view of how to create a secure $2n-$bit permutation $\overline{P}$ from Pq, in this case six rounds are used.

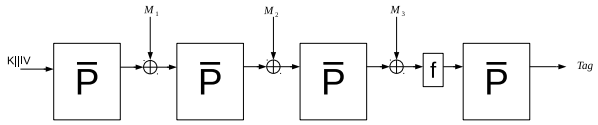

Fig. 3. MAC constructed using $\overline{P}$ each M_i is a $2n-$bit block.

TABLE I
GIFT SBOX.

x	0	1	2	3	4	5	6	7
GS(x)	1	A	4	C	6	F	3	9
x	8	9	A	B	C	D	E	F
GS(x)	2	D	B	7	5	0	8	E

For an specific instantiation of our construction we use the round function of block cipher Gift as P, that give the name of Spongy-Gift. In the next section we explain how Gift round is computed.

C. Gift permutation

Based on security and efficiency facts we have selected as permutation the round function of the lightweight block cipher Gift [1]. In Figure 4 such round function is showed, it consist of three transformations: substitution boxes, bit oriented permutation and add round key. Four our purpose it is not necessary to use round keys, so we just omit this transformation. The substitution box is nibble oriented so it implementation cost is very low, this step consist only in the substitution of input value for the value in the table. In TableI the corresponding values of substitution box $GS(\cdot)$ are shown. This represents the non-linear layer. The bit oriented permutation is a shuffle of current state after the application of $GS(\cdot)$, the corresponding bit permutation is shown in Table II.

Security: The security of this scheme depends on the correct selection of the permutation P, the number of iterations q and the number of rounds of P_q. As a concrete instantiation of P we use the round of $64-$bit block cipher Gift [1]. The authors of gift claim that for nine round of Gift the differential probability is $2^{-46.99}$, taking this fact we can get a good security for six round of P_9 to construct $\overline{P}$. This is a conservative parameters and we are studding the trade off in the selection of parameters q and number of rounds. It is important to take in to account that from $64-$bit permutation we are constructing a $128-$bit permutation. So to process $128-$bit $\overline{P}$ takes $9 * 6 = 54$ iteration of base permutation P.

In the next section we present some implementation results on FPGAs.

III. HARDWARE IMPLEMENTATION

Taking about lightweight primitives there are many studies about the implementation of block ciphers such as [4] and [2]. In general when a specific lightweight primitive is introduced, it hardware implementation is also given to prove that it is lightweight.

The structure of Gift is similar to PRESENT, so some of the techniques using to implement present could be used for Gift. In the literature there are many optimized architectures for PRESENT, presenting different strategies to minimized the size of the implemented design. For example in [3] an implementation using a block ram to store the state is presented, in [5] the SBOX is implemented as a function and in [4] 16-bit data path optimized architectures. In the following subsection we will explain the design decisions.

A. Design Decisions

- Our design is suitable for the FPGAs with 6-input LUTs.
- As we are locking for a very small implementation, we use some specific features of xilinx FPGAs such as SRLC32E (LUT as a 32-bit shift register with dynamic selection of the output).
- Sequential fashion implementation, using data-path sizes lesser than 64 bits to get small area paying the cost of the execution of a round in more clock cycles.
- Get a extremely high frequency is not that important due to the target application, in general IoT requires low power consumption so high frequencies are not require.

B. Implemented Architectures

Reviewing the existing literature for implementation of lightweight block ciphers, in special PRESENT which is quite similar to Gift we decided to implement two ultra lightweight architectures. One is based in shift registers with parallel outputs and load to store the state with 16-bit as data-path size [4], the second one uses a 4-bit data-path and is based in one bit shift registers to store the state and to performed the bits permutations. This architectures use the special features inside the LUTs in Xilinx FPGAs).

Next we will explain in details the 4-bit data-path architecture, the general architecture is presented in Figure 5. As we can see it is very simple, there are four functional blocks and the control unit. In xilinx FPGAs slices of type SLICEM contain LUTs which can be configured as 32-bit shift registers,

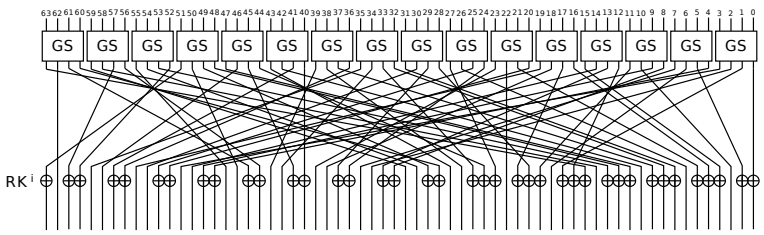

Fig. 4. Gift round function.

TABLE II
GIFT BIT-WISE PERMITUTATION.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Q(i)	0	17	34	51	48	1	18	35	32	49	2	19	16	33	50	3
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Q(i)	4	21	38	55	52	5	22	39	36	53	6	23	20	37	54	7
i	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
Q(i)	8	25	42	59	56	9	26	43	40	57	10	27	24	41	58	11
i	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
Q(i)	12	29	46	63	60	13	30	47	44	61	14	31	28	45	62	15

so they can implement a delay of 32 clock cycles using only one LUT instance 32 flip-flops. Other very useful characteristic of them is that their output could be selected dynamically from 0 to 31, doing them very useful for small implementations because they work at the same time as memory and shift registers. The specific instantiated primitive is SRLC32E [6].

The implementation of SBOX is performed with LUTs. As each LUT can implement two five-input or less functions when all the outputs are common as in this case, the implementation of 4-bit inputs 4-bit outputs SBOX is implemented using only two 6-input LUTs.

The structure of Spongy-gift needs to store the previous value, so one register is required. As the architecture uses an 4-bit data-path it needs to store sixteen 4-bit values in a FIFO mode, and then recover each of them when require. For this purpose only four LUTs configured as 16-bit shift register are used.

Block labeled as permutation register is also composed of 4 LUTs configure as shift registers but in this case, the output of each of them is dynamically selected depending of the value that is necessary to compute the bit oriented permutation in Gift round. As we explain before this functionality allow us to store all the sixteen nibbles and then extract them depending of the value required for the permutation, taking in to account that each clock cycle the values will be shifted in one position and the first position will contain the actual round values.

The case of the multiplexer in the input of SBOX takes 4 LUTs, as it has four inputs and two bits for selection it is a 6-bit input function per each bit. This multiplexer selects between the value of $K||IV$ when the computation is started, the xor of $M_i^{(1)}$, $M_i^{(2)}$ or directly feedback from round function and the value stored in the previous value registers when each , the third input is the xor of actual computed value and the previous one.

Control unit was implemented as a finite state machine which feed the necessary control signals of all the modules in the top par of Figure 5. It produces the control signal for each SRLC32E to produce the bit-oriented permutation output, and the selector of multiplexer.

Data-flow:

Each round of permutation takes nine clock cycles so the total numbers of clock cycles to process $128-$bit block is 54 clock cycles for 6 rounds of permutation. Depending of the data-path size this number increase, for $16-$bit is 216 and for $4-$bit is 864.

When a reset signal is received the first two 64-bit block processed are the $IV||K$, the two first rounds multiplexer selects the input labeled IV, K, M_i and then for next four rounds the input to SBOX is selected from the xor of actual and previous values. When the message blocks are processed the two first rounds the input from the xor of previous value and message M_i, and the following rounds are as before the xor of actual and previous values.

16-bit data-path architecture: As Gift state can be seen as a 4x4 matrix with 4-bit elements, computing the bit permutation of the first row produce the first column of the next round state. The strategy is to store the state column-wise, but replicated four times to allow the parallel reading of a row. For this purpose we used the primitive SRL16E of Xilinx FPGAs.

16-bit Gift and OMAC: To have a point of comparison with other lightweight MAC algorithm we have implemented OMAC using Gift cipher. Gift was implemented using the same strategy as for 16-bit based Gift MAC and the key scheduling was also implemented with SRL16E.

In the next subsection we present experimental results of implemented architectures.

IV. RESULTS AND DISCUSSIONS

In the Table III we list the results obtained using Xilinx ISE tool for Spartan 6 and Virtex 5 FPGAs and Vivado for Artix 7. Some papers also present results for Spartan 3 FPGAs but in our case our architecture is not suitable for 4-input LUTS.

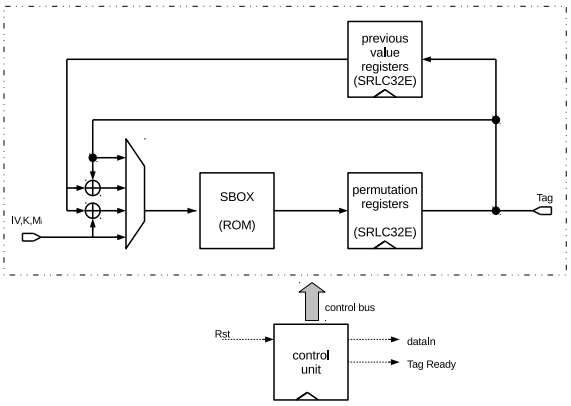

Fig. 5. 4-bit data-path architecture for Spongy-Gift.

There are no implementations of lightweight MACs, so we design an architecture for Gift and implemented OMAC using it. As Present algorithm is very similar to Gift we also compare our designs with it.

From the Table III we can see that our implementation using 4-bit data-path is the smallest in all scenarios even using only 15 slices on Artix 7 FPGA. We compute the throughput using the original frequency but taking into account the number of rounds for Gift-mac. Only in the case of Virtex 5 the design in [4] is better than our, but as we said before for target applications high throughput is not required.

The results in Table III show that we save a significant amount of flip-flops because we are using LUTs as shift registers. The designs synthesized for Spartan 6 FPGAs are more compact than the others, the reason is that the shift registers (not more than 16-bit deep) are packed in pairs in only one LUT.

Our design based on 4-bit data-path is amazingly compact and its frequency is high because the simplicity of the critical path, this architecture is suitable for embedded systems.

These results show that our MAC is really ultra lightweight, it requires very few resources to be implemented in hardware, and its implementation ratio can be study deeper.

V. Conclusions

In this work we introduce a new $2n-$bit public permutation $\overline{P}$, we clarify that it can use any $n-$bit secure permutation but for this specific work we use Gift round function. Using $\overline{P}$ we introduce Spongy-Gift, a new Message Authentication Code, it is a Sponge fashion using $\overline{P}$ as permutation. The implementation results show that Spongy-Gift can be implemented using fey resources on FPGAs and its speed is enough for lightweight cryptography application scenarios.

References

[1] Subhadeep Banik, Sumit Kumar Pandey, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, and Yosuke Todo. GIFT: A small present - towards reaching the limit of lightweight encryption. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, volume 10529 of *Lecture Notes in Computer Science*, pages 321–345. Springer, 2017.

[2] William Diehl, Farnoud Farahmand, Panasayya Yalla, Jens-Peter Kaps, and Kris Gaj. Comparison of hardware and software implementations of selected lightweight block ciphers. In Marco D. Santambrogio, Diana Göhringer, Dirk Stroobandt, Nele Mentens, and Jari Nurmi, editors, *27th International Conference on Field Programmable Logic and Applications, FPL 2017, Ghent, Belgium, September 4-8, 2017*, pages 1–4. IEEE, 2017.

[3] Elif Bilge Kavun and Tolga Yalçin. Ram-based ultra-lightweight FPGA implementation of PRESENT. In Peter M. Athanas, Jürgen Becker, and René Cumplido, editors, *2011 International Conference on Reconfigurable Computing and FPGAs, ReConFig 2011, Cancun, Mexico, November 30 - December 2, 2011*, pages 280–285. IEEE Computer Society, 2011.

[4] Carlos Andres Lara-Nino, Arturo Diaz-Perez, and Miguel Morales-Sandoval. Lightweight hardware architectures for the present cipher in FPGA. *IEEE Trans. on Circuits and Systems*, 64-I(9):2544–2555, 2017.

[5] J. J. Tay, M. L. D. Wong, M. M. Wong, C. Zhang, and I. Hijazin. Compact fpga implementation of present with boolean s-box. In *2015 6th Asia Symposium on Quality Electronic Design (ASQED)*, pages 144–148, Aug 2015.

[6] Xilinx. 7 Series FPGAs Configurable Logic Block, User guide. https://www.xilinx.com/support/documentation/user_guides/ug474_7Series_CLB.pdf. Online; accessed 17 December 2017.

TABLE III
PERFORMANCE OF IMPLEMENTED ARCHITECTURES AND COMPARISON WITH PUBLISHED IMPLEMENTATIONS.

-	**FF**	**LUTs**	**Slices**	**Frequency** (MHz)	**Thoughput** (Mbps)
Spartan 6					
Present-16b [4]	153	170	48	257.40	123.86
Gift-16b	46	165	45	256.85	140.49
Gift-OMAC-16b	75	198	65	249.25	136.341
Spongy-Gift–16b	30	106	36	200.00	117.97
Spongy-Gift–4b	31	42	24	307.225	45.46
Virtex 5					
Present-16b [4]	153	190	67	542.30	2892.26
Present-8b [5]	201	222	62	236.57	630.85
Gift-16b	42	202	61	300.25	164.23
Gift-OMAC-16b	65	225	70	289.12	158.15
Spongy-Gift-16b	26	157	54	287.77	169.74
Spongy-Gift-4b	33	64	21	318.06	47.06
Artix 7					
Gift-16b	50	185	66	376.25	205.81
Gift-OMAC-16b	72	225	70	369.12	201.91
Gift-mac-16b	25	139	61	329.65	194.77
Gift-mac-4b	31	43	15	387.90	57.40