Halka: A Lightweight, Software Friendly Block Cipher Using Ultra-lightweight 8-bit S-box

Sourav Das

Email:sourav10101976@gmail.com

Abstract. This paper presents the design of a lightweight, yet software friendly, block cipher. Most of the lightweight block ciphers are nibble-oriented as the implementation of a 4-bit S-box is much more compact than an 8-bit S-box. This paper uses a novel implementation of multiplicative inverse for 8-bit S-boxes using LFSR requiring only 138 gate-equivalent. With this powerful scheme, we design a lightweight block cipher competitive with existing standards in terms of hardware gate equivalent first time using an 8-bit S-box.

Key words: Multiplicative Inverse, AES, LFSR, Lightweight Cryptography, PRESENT

1 Introduction

The goal of lightweight cryptography is to have cryptographic primitives for extremely constrained devices. While the AES is suitable for most of the applications, the hardware requirement for the AES is considered to be high for these tiny devices. One of the main constituent of the AES is its multiplicative inverse based 8-bit S-box. However, the 8-bit S-boxes rather take a very large area hence they are not very suitable for lightweight cryptography. In [16], a multiplicative inverse based 8-bit S-box is proposed using LFSR that requires only 138 gate equivalent (GE).

We propose a new lightweight block cipher, *Halka*, using these 8-bit multiplicative inverse S-boxes with primitive polynomials. This block cipher has 80-bit Key size and 64-bit block size in accordance with current lightweight block cipher design practices. It uses 8-bit S-boxes as non-linear element. The linear permutation is done simply with wiring. For key-schedule we use a schema similar to PRESENT block cipher mutatis mutandis with 8-bit S-box. The number of rounds is 24 which is sufficient to thwart linear and differential cryptanalysis.

There has been quite a few lightweight block cipher proposals in recent years [9], [14], [19], [21], [24], [39], [38], [42]. Yet, *Halka* is unique in many respects. First and the foremost is the usage of an 8-bit S-box for the first time in lightweight cryptography. This enhances the confidence on the security as the cryptographic properties of 8-bit multiplicative inverse are very strong. Note that, the main constituent of AES security is its 8-bit multiplicative inverse S-box. Second, LFSRs are wonderful constructs for cryptography, but traditionally LFSRs have been used only in stream ciphers. *Halka* uses LFSR for block cipher enhancing

intra-S-box statistical properties. Moreover, LFSRs are extremely low-power devices making the power utilization low which is a requirement for lightweight cryptography. Third, as it will be shown later, the security model of *Halka* combines the strengths of both the AES and PRESENT [9], the later being one of the most popular lightweight block ciphers. This fact provides us with the confidence on the security of *Halka*. Fourth, S-boxes of *Halka* are calculated dynamically without the need for storing them in array. The code size is also very small. This helps software implementation in extremely memory constrained devices. Fifth, the hardware cost in terms of gate equivalent is low and seven percent better than PRESENT[9]. Sixth, it provides a unique parametrization feature which doesn't have any cryptographic weakness or strength but can be useful in various applications (e.g. multi-party communication, as a session parameter etc). Finally, *Halka* provides provision for "Super S-box", enabling super-fast software performance. With eight super S-boxes of size 8×256 bytes, a round transformation is achieved with only eight array lookup along with eight integer addition for 64-bit architecture. For 32-bit architecture, round transformation is achieved with only eight array lookup along with sixteen integer addition with the same size of the array. In this respect, *Halka* is unique in lightweight cryptography where both software aspects and hardware aspects are taken care of.

Lightweight cryptography has been subjected to attacks due to their simplicity [8], [23], [31], [37], [41]. Even PRESENT has been subjected to linear cryptanalysis in various forms in [12], [35], differential combined with algebraic attack [1], [43] and saturation attack [15]. Most of these weaknesses are due to small S-boxes. Hence, we get a strong motivation to design a new lightweight block cipher with 8-bit S-boxes with the existing security strengths of PRESENT.

This paper is organized as follows. Section 2 provides the schematic of multiplicative inverse implementation using LFSR that can be used for AES. It describes the compact hardware method in Section 2.1 and a method for better speed in Section 2.2. Section 2.3 provides the description of parametrization of the S-box. Section 3 introduces a new lightweight block cipher proposal *Halka*. The security analysis of *Halka* is performed in Section 4 and the hardware cost is determined in Section 5. Finally, Section 6 describes the super s-box structure where we can have a super-fast software performance.

2 Multiplicative Inverse Using LFSR

In this section, we reproduce the compact hardware implementations for calculating multiplicative inverse using maximum length LFSR as shown in [16]. See [29] for a detailed description of LFSR. Throughout the rest of the paper referring to LFSR would mean a maximum length LFSR with a given primitive polynomial.

2.1 Compact Hardware Mode

We begin this section with an introduction of how mathematically multiplicative inverse can be calculated using LFSR. The LFSR transformation can be written as for a single cycle:

$S(t+1) = T \cdot S(t)$

where, T is the LFSR transformation matrix, $S(t)$ is the state of the LFSR at t^{th} time instant or the initial seed and $S(t+1)$ is the state of the LFSR at $(t+1)^{th}$ time instant i.e. after running one clock cycle. We can generalize the above equation for any number of cycles p as:

$S(t+p) = T^p \cdot S(t)$

It can be noted that for a maximum length LFSR, running $2^n - 1$ cycles gives back the initial state (where n is the length of the LFSR), i.e.:

$S(t+2^n-1) = T^{2^n-1} \cdot S(t) = S(t) \Rightarrow T^{2^n-1} = 1$, which is the identity element.

To calculate the multiplicative inverse of a given input $S(t+p)$, the task is to find out a new state $S(t+\acute{p})$ of the LFSR so that $p + \acute{p} = 2^n - 1$, implying, $S(t+p+\acute{p}) = S(t+2^n-1) = S(t)$ or alternatively, $T^p \cdot T^{\acute{p}} = T^{2^n-1}$. The above implies the following equation for an 8-bit LFSR:

$$S(t+\acute{p}) = S(t+255-p) \tag{1}$$

One way to implement this is to run the LFSR with a particular initial seed till the LFSR state matches the input, then re-initialize the LFSR with the same seed and run it. When the total number of cycles in both the run is 255 (for 8-bit LFSR) the state of the LFSR gives the multiplicative inverse. Comparison of two eight bit variable requires eight XOR gates, eight NOT gates and one eight input NAND gate along with the LFSR circuit. Additionally, eight 2:1 Mux are required for reloading the initial value to the LFSR. However, we find a better optimization as given below.

Note that, the comparison of the LFSR state with the constant initial seed is very easy. It only needs an eight input NAND (or AND) gate along with a few NOT gates. The input to this NAND gate are the LFSR state bits. For the bits that are zero in the initial seed, the corresponding state bits are negated by NOT gates. When the state becomes equal to the constant initial seed, the output of the NAND gate becomes zero.

Then, we use the following algorithm where the comparison is only performed with constant initial seed.

Require: 8-bit LFSR, initial_seed=S(t), S-box_input=S(t+p)

1: Initialize the LFSR with lfsr_state=S-box_input=S(t+p)
2: Run the LFSR in the forward direction till lfsr_state=initial_seed=S(t)
3: Run the LFSR in the reverse direction
4: Stop when total number cycles in both the above steps is 255
5: Output lfsr_state=S(t+$\acute{p}$)

Theorem 1. *The algorithm above outputs the multiplicative inverse of S-box_input, S(t+p).*

Proof. Let, S-box_input correspond to the lfsr_state after running p cycles of LFSR from initial_seed. This is the state of the LFSR at step 1. Then, in step 2, the number of cycles run is, $255-p$, but the LFSR contains the initial_seed=S(t) at this point. The number of cycles run in step 4 is $255-(255-p) = p$. But as

mentioned in step 3, the LFSR is running in reverse direction at this point with initial seed as S(t). Hence the state of the LFSR is S(t-p). Since, S(t)=S(t+255), the state of the LFSR can also be denoted as S(t+255-p). From Equation 1, this state is the multiplicative inverse of S(t+p). □

Hardware Implementation: The practical implementation is as follows.

1. Use eight two-input flip-flops (e.g. scan flip-flops) to store the LFSR state.

2. Arrange one of the inputs of the flip-flops to make the forward LFSR transformation for a given primitive polynomial. Use 2:1 Muxes at the input to load the S-box input on Reset signal. As the combinational logic of the LFSR is applied only on the first flip-flop, the initial loading can also be applied serially where the 2:1 Mux is used only at the first flip-flop. An eight input NAND gate from the existing counter can indicate the completion of eight cycles for the serial loading. The output of this NAND8 gate can go to another 2-input NAND gate to control the Reset signal. The state of the LFSR counter after eight cycles is considered as the initial state for counting the S-box cycles.

3. Arrange the other inputs of the flip-flops to make reverse LFSR transformation for the same primitive polynomial.

4. The output of the LFSR is connected to an 8-input NAND gate (via a few NOT gates) whose output is connected to the Select input of the flip-flops (via a flip-flop so that the output stays there after a match is found). This provides the comparison with the constant seed and the control logic for the LFSR to run in the reverse direction.

5. An 8-bit LFSR counter is used. The output of the counter is connected to an 8-input NAND gate (via a few NOT gates) to signal when LFSR state contains the multiplicative inverse of the input. This provides the control logic to indicate the completion of 255 cycles.

6. The circuit diagram is shown in Figure 1.

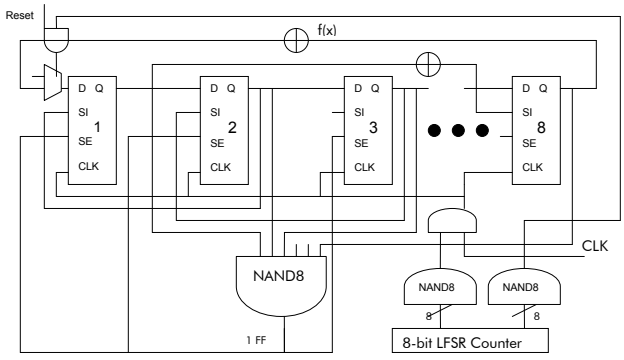

Fig. 1. Circuit Diagram for Compact Hardware Mode

Hardware Cost and Gate Count: We use the primitive polynomial $x^8+x^4+x^3+x^2+1$ which requires three XOR gates for the LFSR feedback function. The total numbers of various gates required to realize the circuit are eight 2-input flip-flops, six 2-input XOR gates, one NAND8 gates and two NOT gates. Using

serial loading, the initialization requires 1 mux, 1 NAND8 and 1 NAND gate. The counter requires one NAND8 gate, three XOR gates and eight 1-input flip-flops. In addition, to avoid the S-box giving zero as output when the input is zero, two NOT gates are added at the input. Using Standard Cells UMCL18G212T3, the total equivalent gate count is $6 \times 16 + 2.67 \times 9 + 4 \times 3 + 0.67 \times 2 + 2.33 \times 1 + 1 \times 1 + 0.67 \times 2 = 138$.

To compare with the existing standards, Canright's implementation takes 253 GE and Satoh's implementation takes 275 GE. Hence Canright's implementation is 85 percent more expensive than this method. The speed is lower in terms of number of cycles. But as stated before, the number of cycles should not be the only metric as the experience shows that the speed reduces drastically when the number of gates is increased due to gate delay. However, as AES uses a polynomial which is irreducible but not primitive, getting the exact AES S-box using this method is not possible. Maximum length LFSRs necessarily need primitive polynomials. Hence, this comparison is with respect to AES-like S-boxes. The AES designers have mentioned that other S-boxes satisfying the same cryptographic properties as with the AES S-box can be used for AES. However, till date that replacement was never attempted as there was no real benefit of doing that. This method generates AES-like S-boxes with same cryptographic properties as in the AES S-box and provides a real benefit of saving the hardware count greatly. By reusing the data state flip-flops for the LFSR and with the common counter for all S-boxes in the S-box layer, the hardware count of AES S-box can be as low as 50 gate equivalent per S-box (see Appendix for a detailed calculation). Hence, we can think about replacing the non-primitive irreducible polynomial of AES with a primitive polynomial that can provide the implementation using LFSR with a great reduction in hardware. Note that, the "super S-box" implementation, Canright's and Satoh's implementations will still be applicable after changing the polynomial. But, since the AES S-box is widely scrutinized and deployed, we leave it at this point for the community to decide on that.

2.2 Parametrization

The multiplicative inverse S-box using LFSR takes the initial seed, $S(t)$, as a parameter. Selection of a seed doesn't have an impact on the main security properties of the S-box i.e. bias, non-linearity, differential uniformity, SAC etc. This also provides a great advantage that the intra-S-box linear transformation used in AES is not required for LFSR based implementation of multiplicative inverse. Since the internal linear transformations are different for different seeds, the number of terms in algebraic expression will vary for different seeds. The algebraic degree, however, is always 7, hence this is not a big threat. LFSRs are already known for their excellent statistical properties which are applied to the S-box automatically. The only additional hardware required is for avoiding the zero to zero mapping in the multiplicative inverse. This can be achieved by putting just a couple of NOT gates at the output of forward S-box and in the input of the reverse S-box. However, special care may be needed in selecting the S-box when used in a cipher depending on the linear layer or the structure of the

cipher. In order to alleviate any concern with this variable number of terms in algebraic expression, we provide a concrete S-box table and compare the security properties with AES.

2.3 A Concrete S-box

A concrete S-box Table was generated with initial seed as 0x16 and constant value 0x24 that is XORed at the output requiring two NOT gates. These values were used for the specific implementation of *Halka*. The full table is shown in Table 1.

Table 1. A Concrete S-box

24	2c	20	dc	26	73	d8	91	25	b7	8f	9c	da	1f	fe	e9
9f	a4	d5	6d	c3	71	32	78	96	db	55	b9	4c	49	6e	42
9a	f9	1d	64	3	5c	a0	0	4a	d7	e3	8e	75	af	b	a
7d	4d	5b	1a	1c	e7	6a	74	10	6	92	29	81	79	17	40
7	7b	69	ca	c8	b8	ef	84	c2	37	3a	98	df	66	12	b6
13	8	5d	fc	47	31	f1	21	8c	14	e1	51	33	19	b3	65
88	4e	90	70	1b	a8	3b	cc	38	15	45	a7	83	39	c	de
a1	3e	c1	b5	eb	7f	ac	a2	1	76	9b	8a	b4	bd	99	16
35	d4	8b	4f	2	54	53	be	52	c7	ea	9	41	c6	f4	b1
58	57	6b	2d	f8	ab	87	7a	f6	59	a3	85	61	3f	9e	ed
63	bf	fd	b2	e8	18	d2	48	7c	95	f	2e	44	ce	5f	a6
f0	8d	3c	f5	46	23	1e	d0	2f	ee	ba	34	6f	5a	4	5e
c5	f2	c4	11	e2	7e	e0	e	dd	bb	9d	62	80	2b	ae	50
aa	97	bc	c9	94	72	e5	d3	77	86	2a	cd	b0	5	d9	d1
e6	e4	a9	ad	d6	56	6c	30	43	ff	89	cb	60	f7	67	cf
a5	36	c0	d	93	fb	82	f3	27	ec	4b	68	22	fa	28	3d

The comparison of security properties of the S-box generated with the above parameters with AES S-box is shown in Table 2. It can be seen that the security properties are essentially same as it is expected. Note that, we have used algebraic normal form to compare the algebraic properties for convenience, unlike the univariate polynomial expression given in original AES specification.

Table 2. Comparison of Security Properties with AES

S-box	Max Alg Term	Min Alg Term	Alg Deg	Diff Uni	Bias	Max SAC	Min SAC	Max NL	Min NL
AES	145	110	7	4	2^{-4}	144	116	114	112
Halka	139	118	7	4	2^{-4}	140	112	114	112

Thus, we presented a novel and compact method for implementing multiplicative inverse for 8-bit S-boxes. The area requirement is so small that even a lightweight block cipher can be proposed with 8-bit S-boxes. In the next section, we provide a schematic of a lightweight block cipher, *Halka*, with 8-bit S-boxes.

3 *Halka*: A Lightweight Block Cipher

Halka is a block cipher having 64-bit block size and 80 bit Key size with possible applications in pervasive computing including RFID, sensor network and smart devices where 80-bit Key size is sufficient. Internally, it has a substitution permutation network structure where a layer of 8-bit S-boxes are used for substitution with wiring permutation and an XOR with a round Key. The S-boxes are calculated dynamically and have the properties of multiplicative inverse with

respect to a primitive polynomial. In this instance of Halka, the polynomial $X^8 + X^4 + X^3 + X^2 + 1$ is used. There are 24 rounds of this transformation before the cipher text is produced. The initial seed used in LFSRs provides an optional parameter for the cipher. The algorithmic description of block cipher *Halka* is given below:

Require: Key K, Plaintext P
1: STATE S=P
2: $generateRoundKeys(K)$
3: **for** i=1 to 24 **do**
4: $addRoundKey(S, K_i)$
5: $transformSBoxLayer(S)$
6: $permute(S)$
7: **end for**
8: $addRoundKey(S, K_{25})$

3.1 Details of a Round

A particular round of *Halka* consists of an XOR with round key, S-box transformation and a wire permutation.

XOR with Round Key: Let us denote the state at i^{th} round as $S_i = (s^i_0 \cdots s^i_{63})$ and round key $K_i = (k^i_0 \cdots k^i_{63})$. At the beginning of each round a bitwise XOR is performed between the state and the round key:

$$S_i \leftarrow S_i \oplus K_i$$

S-box Transformation: The S-box transformation layer consists of eight 8-bit S-boxes arranged in parallel. Let us denote the S-box transformation $G(S)$. The state S_i is divided into eight byte words, bw_j $(0 \leq j \leq 7)$ with $bw_j = (s^i_{8*j+0} \cdots s^i_{8*j+7})$. With this transformation, we have:

$$S_i \leftarrow G(bw_0) \parallel \cdots \parallel G(bw_7)$$

Permutation: After the S-box layer, a linear permutation is performed. The permutation is random with the constraint that every bit from each 8-bit S-box affects one and only one bit in each of the eight 8-bit S-boxes in the next layer. The permutation is performed as:

$s^i_k \leftarrow s^i_j$, $j, k \in (0, \cdots, 63)$, where j and k mapping is shown in Table 3.

Table 3. Halka Permutation Mapping

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

Key Schedule: *Halka* takes an 80-bit Key as input which is stored in a key-state register $KS = (ks_0, \cdots, ks_{79})$. At every round, the round key K_i is derived from the leftmost 64 bits of KS. Hence,

$$K_i \leftarrow (ks_{79} \cdots ks_{16})$$

The key state registers are then rotated by 57 bit position to the left and then the leftmost eight bits are transformed using the S-box and finally the 5-bit round counter is XORed with $ks_{19}ks_{18}ks_{17}ks_{16}ks_{15}$ i.e. the rightmost bits which will be transferred to round-key. The round_counters are the actual counter value for that round. Mathematically,

$$\begin{gathered}(ks_{79}ks_{78} \cdots ks_1ks_0) = (ks_{22}ks_{21} \cdots ks_{24}ks_{23}) \\ (ks_{79} \cdots ks_{72}) = G(ks_{79} \cdots ks_{72}) \\ (ks_{19}ks_{18}ks_{17}ks_{16}ks_{15}) = (ks_{19}ks_{18}ks_{17}ks_{16}ks_{15}) \oplus \text{ round\_counter}\end{gathered}$$

4 Security Analysis of *Halka*

In this section, we provide the security analysis of *Halka*. The security model of *Halka* is derived from both AES and PRESENT. It uses 8-bit multiplicative inverse S-box that makes the resistance against differential and linear cryptanalysis easily provable as shown next and uses bit-wise permutation that prevents structural attacks.

4.1 Differential and Linear Cryptanalysis

Differential cryptanalysis [2] was introduced by Biham against the DES cipher. On the other hand, linear cryptanalysis was introduced by Matsui [30], also applied to DES first. These two are the most devastating attacks against block cipher and it is imperative that any new block cipher shows the resistance against differential and linear cryptanalysis.

For *Halka* it is straightforward to show the resistance against linear and differential cryptanalysis.

Theorem 2. *After 24 rounds of* Halka*, the probability of any differential characteristics is* 2^{-288} *and the linear bias is* 2^{-192}

Proof. Since *Halka* uses a wiring permutation where each bit from the previous layer affects a different S-box, the *branch number* is 3. The branch number gives the lower bound of the *active* S-boxes. Hence, the total number of output active S-boxes is two. Now the differential uniformity of multiplicative inverse is 4 [34]. Hence the differential probability is 2^{-6} for eight bit S-boxes used in Halka. The probability of a differential per round is $(2^{-6})^2 = 2^{-12}$ [22]. After 24 rounds, the total probability of any differential characteristic is $(2^{-12})^{24} = 2^{-288}$.

The linear bias of multiplicative inverse is 2^{-4}. Hence, on a similar reasoning with differential probability, it can be shown that the total linear bias of *Halka* is $(2^{-4})^2)^{24} = 2^{-192}$. □

An improvement of the above bound is possible as follows. For an S-box, there are several differential with probability 2^{-7} such that the input differential has one active bit and output differential has just one active bit too (this kind of analysis is used in cryptanalysis of Keccak). Since the linear layer is a bit

permutation, we can find simply a differential characteristic which has exactly one active S-box in each round. So the actual lower bound of the probability for r rounds is $2^{-7\times r}$. When the number of rounds is 24, the probability is 2^{-168}. We also tried to perform advanced differential attack like truncated differential analysis [25], [26]. The first hurdle was that the S-box listed in the paper is a perfect S-box with respect to differentials. For all the input differences, the number of non-zero entries in the difference distribution table was exactly 127. Every row in the difference distribution table (of course, except the first row) contains 126 twos and 1 four. Hence, we found it extremely difficult and tedious to find a specific truncated differential characteristic that propagates through the S-boxes easily. Such a good distribution made the impossible differential analysis also very difficult. Augmented with this was the difficulty for the random permutation. The above facts provides us with the confidence of the security of *Halka* with respect to differential attacks. See Appendix for a couple of specific cases.

4.2 Related Key and Slide Attacks

The related key attack [3] and slide attack [4] exploit the weakness in key schedule algorithm. These attacks try to find distinguishers in the sub-keys. The related key attack has even been successful to find a related key distinguisher on reduced round AES-256 [5]. Other related attacks are a boomerang attack [40] on the full round of AES [6] and a near practical related key attack [7].

However, *Halka* uses PRESENT style key-scheduling algorithm. There is no evidence that such attacks could be applicable to PRESENT key scheduling algorithm. *Halka* simply strengthens the PRESENT key scheduling algorithm by using an 8-bit S-box. We see now how the arguments against related key and slide attacks given in PRESENT are made stronger for Halka.

- all bits in the key state are a non-linear function of 80-bit key by round 11.
- each bit in the key register after round 11 depends on at least eight of the user-supplied key bits, and
- each bit has an algebraic degree 7 by round 11. This is further enhanced for 32 bits in the remaining four rounds.

4.3 Other Attacks

We describe how a few more common attacks are thwarted in *Halka*.

Structural Attacks: Structural attacks like integral attack[27] and bottleneck attack[18] exploit the word like structures in the cipher. *Halka* permutation is bitwise and hence deriving such word like structures is not possible.

Algebraic Attack: *Halka* uses AES-like S-box which can be described by eight algebraic equations of degree 7 where the number of terms range from 139 to 118. Unlike 4-bit S-boxes used in PRESENT, this system cannot be described in quadratic equations with small number of terms. Hence, it doesn't require an extensive analysis unlike PRESENT against algebraic attacks for their applicability on *Halka* reinforcing the claim of enhanced security over PRESENT.

Cube Attack: Cube attack [17] has been mounted on LFSR based stream ciphers by deriving low degree black-box polynomial. Since the cipher uses LFSR

one may think Cube Attack may be applicable. But note that the LFSR is used only for implementation. The theory of the cipher is derived from the strengths of multiplicative inverse like AES. AES is believed to be immune to cube attacks. Hence, such an attack is not applicable for this cipher.

Side Channel Attack: Among the side channel attacks, simple power attack and simple timing attack are not possible as there is always a constant number of cycles in the S-box and the permutation is only wiring. But as shown in [11] the multiplicative inverses have a rather high transparency order [36] leading to differential power analysis (DPA) [28]. Since the S-box is generated using LFSR, small redundant circuits could be developed to keep the power consumption constant in every cycle which would help preventing DPA.

There have been some cryptanalytic efforts against PRESENT reported in the literature [1], [43], [12], [35], [15] which are mainly based on the smaller S-box of PRESENT. For example, [12] specifically builds on weak correlation property of the PRESENT S-box and some what regular bit permutation. However, the multiplicative inverse based S-box in *Halka* seems to weaken them as its linear bias is 2^{-4} and the differential probability is 2^{-6}. Also, the permutation in *Halka* is perfectly random. Hence such attacks unlikely to succeed against *Halka*. The block ciphers have undergone major cryptanalytic efforts in the last few decades. So it is impossible to cover all of them here. Nevertheless, we covered the important ones to have enough confidence on the security of *Halka*. In the next section, we check the hardware cost of *Halka*.

5 Hardware Cost of Halka

As the goal of *Halka* is to provide strong security with very minimal hardware, the S-boxes are implemented in the compact hardware mode as described before. The following are the hardware modules and their associated cost i.e. gate count for hardware implementation of *Halka*.

S-box Layer: The S-box layer requires eight 8-bit S-boxes. As shown previously, the hardware requirement for each S-box is 138 gate equivalent. However, this cost includes the cost of the 8-bit counter which will be common for all the S-boxes. Hence we need to subtract the cost of the counter circuit. The counter requires two NOT gates, one NAND8 gate, three XOR gates and eight 2-input flip-flops. The total hardware cost of the counter is $0.67 \times 2 + 4 \times 1 + 2.67 \times 3 + 6 \times 8 = 61.33$. Hence the cost of the each S-box sans the counter is 138-61.33=76.67 gate equivalent. Hence the total hardware cost of the S-box layer is $76.67 \times 8 + 61.33 = 674.69$.

Data State: Since the S-box layer already uses flip-flops for LFSR, the same set of LFSRs will be used for data state. However, to load the LFSR content for different rounds eight 2:1 Muxes, a NAND2 and a NAND8 gate will be required using serial loading for each eight bit block. Hence, the total hardware cost for the Data State is $2.33 \times 8 + 1 + 4 = 23.64$ gate equivalent.

Permutation: Since *Halka* uses wire permutation, the hardware cost for permutation is zero.

Round Counter: A round counter is required to count upto 24 for 24 rounds of Halka. This needs a 5-bit LFSR counter where the hardware requirement is

five flip-flops, one XOR gate, one 5-input NAND gate. Total hardware cost is $6 \times 5 + 2.67 \times 1 + 4 \times 1 = 36.67$
Key State: The key state requires 80-bit flip-flops and the total cost is 480.49 (same as in PRESENT) gate equivalent.
Key State S-box: It requires 76.67 gate equivalent reusing the counter from S-box layer.
Key State Counter XOR: Four XOR gates are required totalling $4 \times 2.67 = 10.68$ equivalent gates.
Key XOR: Sixty four XOR gates are required totalling gate equivalent count as 170.84 (same as PRESENT).
Total Gate Count: The total hardware cost of *Halka* is $676.49+23.64+36.67+480.49+76.67+10.68+170.84 = 1475$.

So, the total hardware cost of *Halka* is 7 percent better than PRESENT (which requires 1569 gate equivalent). However, *Halka* increases the security of PRESENT greatly by using 8-bit multiplicative inverse as S-box similar to AES. A detailed comparison of other lightweight block ciphers (including PRESENT) can be found in [21] that could be extended for comparison with *Halka*. We don't reproduce the Table here due to space limitation. Note that, PRESENT reports an implementation on later work with less than 1000 GE by serializing the S-boxes. Such serialization of S-boxes is also possible in *Halka* greatly reducing the gate count in overall implementation (a future work). Note that, the mux and the counter hardware required for serialization becomes half of PRESENT in *Halka*. This compensates the additional hardware required for 8-bit S-boxes over 4-bit S-boxes.

6 Software Implementation

One main advantage of *Halka* over PRESENT is that it allows efficient software implementation. In this respect, it can have "super s-box" structure like AES and hence, it is comparable with AES in terms of software efficiency.

6.1 Implementation in 64-bit Architecture

A super S-box of 256 elements with each element of size 64 bits is constructed as follows. For each index of the 256 element array, the S-box transformation is performed with that index. The output of the S-box are mapped according to the permutation defined in Table 3 for each of the eight S-boxes to eight bits of a 64-bit integer. Rest all bits of the 64-bit integer is kept as zero. This 64-bit integer is kept at that index of the array. Thus a table of size $8 * 256$ bytes is formed. This is performed for all the eight S-boxes. Thus there are eight such tables (arrays).

The round transformation (including substitution and permutation) is performed as follows. For j^{th} input byte (of 64-bit input block), after the j^{th} super S-box lookup, a simple integer addition is performed. Note that, simple addition is sufficient as for each bit position of this 64-bit block, there could be a maximum one logic 1 in this addition operation implying no carry bit.

6.2 Implementation in 32-bit, 16-bit and 8-bit Architectures

In architectures below 64-bits, the size of the Table remains the same, but the 64 state bits should be divided into required number of integers. The table look-up and the addition operations are performed in the same way as with 64-bit architecture, but on the individual integers. For example in 32-bit architecture, two 32-bit integers are used for each of the super s-box entry. After super s-box lookup, both the 32-bits integers are added with the output of other bytes.

Note that, the schema presented above can also be extended for PRESENT which is still unpublished. The only previously reported work [20] for efficient implementation of PRESENT in software provides an implementation for 8-bit architecture. But this method is applicable for all 64-bit, 32-bit, 16-bit and 8-bit architectures. However, if this schema is adopted for PRESENT, the number of array look-ups required is sixteen and the number of integer additions is also sixteen for 64-bit architecture. In addition, PRESENT has 31 rounds as opposed to 24 rounds in *Halka*. Hence, PRESENT will be slower by three times compared to *Halka* if this approach is adopted.

7 Conclusion

In this paper a new lightweight block cipher is proposed. The cipher has 8-bit multiplicative inverse which is implemented using a novel algorithm requiring very compact hardware. The cipher is shown to require similar gate count as PRESENT yet provides a lot more security.

References

1. M. Albrecht, C. Cid. Algebraic techniques in differential cryptanalysis. FSE 2009. LNCS, vol. 5665, pp. 193-208. Springer, Heidelberg. 2009.
2. E. Biham and A. Shamir: Differential Cryptanalysis of DES-like Cryptosystems. Journal of Cryptology, vol. 4, no. 1. pp. 3–72. 1991.
3. E. Biham. New Types of Cryptanalytic Attacks Using Related Keys. Proceedings of Eurocrypt 93. LNCS. vol. 765, pp 398-409, Springer-Verlag. 1994.
4. A. Biryukov and D. Wagner. Advanced Slide Attacks. Proceedings of Eurocrypt 2000. LNCS. vol. 1807, pp. 589-606, Springer-Verlag. 2000.
5. A. Biryukov, D. Khovratovich and I. Nikolić. Distinguisher and Related-Key Attack on the Full AES-256. http://eprint.iacr.org/2009/241. 2009.
6. A. Biryukov and D. Khovratovich. Related-key Cryptanalysis of the Full AES-192 and AES-256. http://eprint.iacr.org/2009/317. 2009.
7. A. Biryukov, O. Dunkelman, N. Keller, D. Khovratovich and A. Shamir. Key Recovery Attacks of Practical Complexity on AES Variants With Up To 10 Rounds. http://eprint.iacr.org/2009/374.
8. C. Blondeau, M. Naya-Plasencia, M. Videau and E. Zenner. Cryptanalysis of ARMADILLO2. http://eprint.iacr.org/2011/160.
9. A. Bogdanov et al. PRESENT: An Ultra-Lightweight Block Cipher. CHES 2007. LNCS vol. 4727. pp. 450–466. Springer. 2007.
10. D. Canright. A very compact S-box for AES. CHES 2005. LNCS vol. 3659. pp. 441-455. Springer, 2005.
11. C. Carlet. On highly nonlinear S-boxes and their inability to thwart DPA attacks. Indocrypt 2005. LNCS vol. 3797, pp. 49–62, 2005.

12. J. Y. Cho. Linear Cryptanalysis of Reduced-Round PRESENT. Topics in Cryptology - CT-RSA 2010 LNCS vol. 5985, pp 302–317. 2010.
13. P. Chodowiec and K. Gaj. Very compact FPGA implementation of the AES algorithm. CHES 2003, LNCS vol. 2279. pp. 319-333. Springer, 2003.
14. C. de Cannire, O. Dunkelman, M. Knezevi: Katan and ktantana family of small and efficient hardware-oriented block ciphers. CHES 2009. LNCS, vol. 5747, pp. 272-288. Springer, Heidelberg. 2009.
15. B. Collard, F. Standaert. A statistical saturation attack against the block cipher PRESENT. Topics in Cryptology CT-RSA 2009. LNCS vol. 5473, pp. 195-210. Springer, Heidelberg. 2009.
16. Sourav Das. Ultra-lightweight 8-bit Multiplicative Inverse Based S-box Using LFSR. http://eprint.iacr.org/2014/022
17. I. Dinur, A. Shamir. Cube Attacks on Tweakable Black Box Polynomials, EUROCRYPT 2009. Also on Cryptology ePrint Archive, Report 2008/385. 2009.
18. H. Gilbert and M. Minier. A Collision Attack on 7 Rounds of Rijndael. In Proceedings of Third Advanced Encryption Standard Conference, National Institute of Standards and Technology, pp. 230-241, 2000.
19. Z. Gong, S. Nikova, Y. W. Law. KLEIN: A New Family of Lightweight Block Ciphers. RFIDSec 2011. LNCS vol. 7055, pp. 1–18. Springer. 2011.
20. Z. Gong, P. Hartel, S. Nikova and B. Zhu. Towards Secure and Practical MACs for Body Sensor Networks. Indocrypt 2009. LNCS vol. 5922. pp. 182–198. Springer. 2009. Also see: http://cis.sjtu.edu.cn/index.php/Software_Implementation_of_Block_Cipher_PRESENT_for_8-Bit_Platforms
21. J. Guo, T. Peyrin, A. Poschmann. The LED Block Cipher. Cryptographic Hardware and Embedded Systems - CHES 2011, LNCS, Springer-Verlag 2011.
22. S. Hong, S. Lee, J. Lim, J. Sung, D. Hyeon Cheon, and I. Cho, Provable security against differential and linear cryptanalysis for the spn structure, Fast Software Encryption - FSE 2000, LNCS. vol. 1978. pp. 273-283. Springer, 2000.
23. T. Isobe, K. Shibutani. Security Analysis of the Lightweight Block Ciphers XTEA, LED and Piccolo. ACISP 2012. LNCS vol. 7372. pp. 71–86. 2012.
24. L. R. Knudsen, G. Leander, A. Poschmann, M.J.B. Robshaw, PRINTcipher: A Block Cipher for ICPrinting, In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS vol. 6225, pp. 16–32. Springer-Verlag. 2010.
25. L.R. Knudsen and T. Berson. Truncated Differentials of SAFER. FSE 1996. LNCS vol. 1039. pp. 15-26, Springer–Verlag, 1996.
26. L.R. Knudsen, M.J.B. Robshaw, and D.Wagner. Truncated Differentials and Skipjack. Crypto 1999. LNCS vol. 1666. pp. 165-180. Springer–Verlag. 1999.
27. L.R. Knudsen and D.Wagner. Integral Cryptanalysis. FSE 2002, LNCS vol. 2365, pp. 112-127, Springer-Verlag, 2002.
28. P. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In M. Wiener, editor, Advances in Cryptology - CRYPTO 1999, LNCS vol.1666, pp. 388–397. Springer-Verlag, 1999.
29. R. Lidl and H. Niederreiter. Introduction to Finite Fields and Their Applications. Cambridge. Cambridge Univercity Press. 1994.
30. M. Matsui. Linear Cryptanalysis Method for DES Cipher. Advances in Cryptology - EUROCRYPT 93. LNCS vol. 765. pp. 386–397. Springer-Verlag. 1994.
31. F. Mendel, V. Rijmen, D. Toz and Kerem Varici. Differential Analysis of the LED Block Cipher. http://eprint.iacr.org/2012/544.pdf. 2012.

32. N. Mentens, L. Batina, B. Preneel and I. Verbauwhede. A systematic evaluation of compact hardware implementations for the Rijndael S-box. CTRSA 2005. LNCS vol. 3376, pp. 323–333. Springer, 2005.
33. A. Moradi, A. Poschmann, S. Ling, C. Paar and H. Wang. Pushing the Limits: A Very Compact and a Threshold Implementation of AES. Eurocrypt 2011. LNCS vol. 6632. pp. 69–88. Springer, 2011.
34. K. Nyberg. Differentially uniform mappings for cryptography. Advances in Cryptology, Eurocrypt'93. LNCS vol. 765. pp. 55–64. Springer-Verlag. 1994.
35. K. Ohkuma. Weak keys of reduced-round PRESENT for linear cryptanalysis. SAC 2009. LNCS Vol. 5867, pp 249–265. 2009.
36. E. Prouff. DPA attacks and S-boxes. FSE 2005, LNCS vol. 3557, pp. 424–441, 2005.
37. M.-J. O. Saarinen. Cryptanalysis of Hummingbird-1. Proceedings of FSE 2011, LNCS. vol. 6733. pp. 328–341. Springer. 2011.
38. K. Shibutani, T. Isobe, H. Hiwatari, A. Mitsuda, T. Akishita, T. Shirai. Piccolo: An Ultra-Lightweight Blockcipher. CHES 2011. LNCS. vol. 6917. pp. 342–357. Springer. 2011.
39. T. Suzaki, K. Minematsu, S. Morioka, E. Kobayashi. Twine: A Lightweight, Versatile Blockcipher. ECRYPT Workshop on Lightweight Cryptography (2011). http://www.uclouvain.be/crypto/ecrypt_lc11/static/post_proceedings.pdf. 2011.
40. D. Wagner. The boomerang attack. In FSE'99, LNCS. vol. 1636. pp. 156–170. Springer, 1999.
41. M. Wagren. Some Instant- and Practical-Time Related-Key Attacks on KTANTAN32/48/64. http://eprint.iacr.org/2011/140. 2011.
42. H. Yap, K. Khoo, A. Poschmann, M. Henricksen. EPCBC - A Block Cipher for Electronic Product Code Encryption. CANS 2011, LNCS, Springer-Verlag, 2011.
43. M. Wang. Differential cryptanalysis of reduced-round PRESENT. AFRICACRYPT 2008. LNCS. vol. 5023, pp. 40-49. Springer,Heidelberg. 2008.

8 Appendix

8.1 Test Vectors

Key=0xff ff ff ff ff ff ff ff ff ff
Plaintext=0x00 00 00 00 00 00 00 00
Ciphertext=0xf7 4f 84 47 f6 80 64 38
==========
Key=0x00 00 00 00 00 00 00 00 00 00
Plaintext=0x00 00 00 00 00 00 00 00
Ciphertext=0x01 36 ff 2b 22 fd ae d5
==========
Key=0xff ff ff ff ff ff ff ff ff ff
Plaintext=0xff ff ff ff ff ff ff ff
Ciphertext=0xca 6f 36 92 22 52 f0 5a

8.2 Notes on Gate Equivalent

We have not done the actual implementation on ASIC as we don't have those tools. Instead, we have used Xilinx 7i FPGA to check the hardware. Gate equivalents for various hardware primitives that are used in this paper for estimation are given in the following Table. These figures are mainly taken from the thesis

of Poschman (http://eprint.iacr.org/2009/516.pdf) to have a fare comparison with PRESENT. Note that, NAND8 does not exist in those libraries. But ATL 60 and ATLS60 series datasheet shows that a NAND8 gate needs 3.5 times the site count of a NAND2 gate. The datasheet can be found in the following link: http://www.datasheetcatalog.org/datasheet/atmel/DOC0388.PDF. So, I think we can safely assume that if the support of NAND8 gate is provided in the library used in PRESENT, the gate equivalent count will be 4. If the reader is not convinced with that assumption, the error margin is really less. We have only three NAND8 gates in the compact circuit. In the worst case, the NAND8 gate will require 7 GE by combining 2-input NAND gates. In that case, the gate equivalent count of compact hardware mode will be 138+3(7-4)=147.

Table 4. Gate Vs Gate Equivalent Count

Gate	GE	Gate	GE	Gate	GE	Gate	GE
NOT	0.67	NAND, NOR	1	2:1 MUX	2.33	NAND8	4
XOR	2.67	AND, OR	1.33	2-input FF	6		

8.3 A Note on Permutation of *Halka*

Halka uses a random permutation with a constraint that each bit in the input S-box affects a different S-box in the next layer. This could have been achieved by the following permutation which is easy to describe and easier to implement in software.

$$s^i_{8*j+k} \rightarrow s^i_{8*k+j}, \forall j, k \in (0, \cdots, 7)$$

Now observe the following with respect to the above permutation layer (thanks to an anonymous referee). The state of input differences: $(00, 0x7F, 0x7F, \cdots, 0x7F)$ results in the identical state of output differences $(00, 0x7F, 0x7F, \cdots, 0x7F)$ and hence is invariant with respect to the permutation. Similarly, for the state $(0x80, 00, \cdots, 00)$. Depending on the exact structure of the S-box this may be exploited to build iterative trails. For example, if the differentials $(0x7F \rightarrow 0x80)$ and $(0x80 \rightarrow 0x7F)$ both have non-zero probability with respect to the S-box, one can construct the following 4-round iterative trail:

0x8000000000000000 = Input XOR difference to round 1
[S-LAYER]
0x7F00000000000000
[P-LAYER]
0x0080808080808080
[S-LAYER]
0x007F7F7F7F7F7F7F
[P-LAYER]
0x007F7F7F7F7F7F7F
[S-LAYER]
0x0080808080808080
[P-LAYER]
0x7F00000000000000
[S-LAYER]

0x8000000000000000
[P-LAYER]
0x8000000000000000 = Output XOR difference after round 4

But, for the specific S-box in *Halka*, both the above differentials have zero probability and hence this trail is not possible. However, this could be easily extended for any pair of values from the following two sets (0x80, 0x40, 0x20, 0x10, 0x08, 0x04, 0x02, 0x01) and (0x7F, 0xBF, 0xDF, 0xEF, 0xF7, 0xFB, 0xFD, 0xFE). We don't have any LFSR based S-box using all the seeds with all the primitive polynomials of degree eight where the differentials are zero for all the pairs above.

The current permutation of *Halka* is such that such trails cannot be constructed. We did not find any such trail with the current *Halka* permutation. But we keep this section to let the cryptanalysts know about this and explore this further for the cryptanalysis of *Halka*.