Revisiting AES SBox Composite Field Implementations for FPGAs

Aditya Pradeep, Vishal Mohanty, Adarsh Muthuveeru Subramaniam, and Chester Rebeiro

***Abstract*—Composite fields are used for implementing the advanced encryption standard (AES) SBox when compact and side-channel resistant constructions are required. The prior art has investigated efficient implementations of such SBoxes for application specific integrated circuit (ASIC) platforms. On field programmable gate arrays (FPGAs); however, due to the considerably different structure compared with ASICs, these implementations perform poorly. In this letter, we revisit composite field AES SBox implementations for FPGAs. We show how design choices and optimizations can be made to better suit the granular look-up tables that are present in modern FPGAs. We investigate 2880 SBox constructions and show that about half of them are better than the state-of-the-art composite field implementation. Our best SBox implementation is 18% smaller compared with the state-of-the-art implementation on an FPGA.**

***Index Terms*—Advanced encryption standard (AES) SBox, composite fields, field programmable gate array (FPGA).**

I. Introduction

THE BIGGEST challenge in designing compact implementations of advanced encryption standard (AES) is in realizing the SBox. For a given input, a, the AES SBox computes the inverse element (a^{-1}) in the field $GF(2^8)$ and then performs an affine transformation. The SBox is typically implemented either by lookup tables or by using composite fields. Lookup table-based implementations "look up" the SBox output for the given input. Composite field implementations perform the SBox operation using a composite field of the form $GF((2^m)^k)$, where $m \times k = 8$. For application specific integrated circuit (ASIC) platforms, it is well established that the composite field approach leads to more compact AES SBox realizations. Considerable efforts have gone into finding the ideal composite field that would produce the smallest SBox. Canright [5], for instance, evaluated a large number of composite fields to identify the most suitable one. The composite fields that Canright found require 180 equivalent NAND gates to implement the forward SBox and 181.75 equivalent NAND gates for the inverse SBox. Boyar and Peralta [2] further reduced the size of the SBox to 128 gates. This is the bestknown result to date.

Unlike ASICs where logic gates can be as small as having 2-inputs, field programmable gate arrays (FPGAs) use look-up tables (LUTs), which are of much larger granularity. A single 6-input LUT, which is the smallest programmable element in many modern FPGAs, can implement Boolean functions comprising of six variables. Thus, an optimal design for an ASIC platform may not be the best choice for an FPGA. For instance, the Boyar and Peralta SBox implementation [2], which is tuned for an ASIC platform, utilizes 85 LUTs on a Xilinx Artix 7 FPGA when synthesized with the Xilinx Vivado 2017.4. This is more than double the number of LUTs required by our SBox implementation on the same FPGA. This motivates the exploration of optimal AES SBox implementations for FPGA platforms.

In an FPGA with 6-input LUTs, a lookup table-based implementation of an SBox takes just 32 LUTs, while Canright's best composite field-based SBox occupies 48 LUTs [3]. Thus, for FPGAs, a lookup table approach for the AES SBox leads to more compact implementations of AES compared to composite field-based implementations.

Many applications of AES require side-channel protection. Many new side-channel countermeasures, such as masking [4] and threshold implementations [1], [7] are more efficiently realizable with composite field-based SBox implementations. Thus, the composite field-based implementations is often the preferred choice for FPGA platforms despite needing a larger number of LUTs.

In this letter, we revisit composite field implementations of AES SBoxes in the hope of finding implementations that are compact for FPGA platforms. We explore over 2800 AES SBox constructions using composite fields of the form $GF((2^4)^2)$. We argue that these fields are more suited for FPGA platforms compared to composite fields of the form $GF(((2^2)^2)^2)$, which Canright had evaluated for ASICs [5]. Each of the SBox constructions we evaluate is optimized to best utilize the FPGA resources. We were able to identify several AES SBox implementations that are better than Canright's implementation. Our best results require 39 6-input LUTs; 9 LUTs fewer than Canright's best SBox implementation. Our best inverse SBox requires 40 LUTs; 8 fewer than Canright's best inverse SBox implementation.

The organization of this letter is as follows. Section II provides the necessary background on the AES SBox, composite

Manuscript received November 23, 2018; accepted February 2, 2019. Date of publication February 12, 2019; date of current version August 28, 2019. This manuscript was recommended for publication by R. Karri. *(Corresponding author: Chester Rebeiro.)*

A. Pradeep is with the Electrical Engineering Department, Indian Institute of Technology Madras, Chennai 600036, India (e-mail: ee14b068@smail.iitm.ac.in).

V. Mohanty and C. Rebeiro are with the Computer Science and Engineering Department, Indian Institute of Technology Madras, Chennai 600036, India (e-mail: cs15b039@smail.iitm.ac.in; chester@iitm.ac.in).

A. M. Subramaniam is with the Department of Computer Science and Engineering, Texas A&M University, College Station, TX 77843 USA (e-mail: msadarsh231996@gmail.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/LES.2019.2899113

finite fields, and FPGA architecture. Section III discusses composite field implementation aspects of the AES SBox, while Section IV has the results of our findings. Section V discusses the use of composite fields in threshold implementations. The final section concludes this letter.

II. Background

In this section, we provide a background of composite fields and a typical FPGA architecture.

A. AES Sbox

The input to the AES SBox is an element a in the field $GF(2^8)$, which is defined over the irreducible polynomial $R(x) = x^8 + x^4 + x^3 + x + 1$ [6]. The output is defined by the following equation:

$$\text{SBox}(a) = \begin{cases} Ma^{-1} + N & \text{if } a \neq 0 \\ N & \text{if } a = 0 \end{cases} \quad (1)$$

where a^{-1} is the multiplicative inverse of $a, N \in GF(2^8)$, M is an 8×8 binary matrix. The inverse AES SBox is similarly defined as follows:

$$\text{invSBox}(b) = \begin{cases} \{M^{-1}(b+N)\}^{-1} & \text{if } b \neq N \\ 0 & \text{if } b = N. \end{cases} \quad (2)$$

The $GF(2^8)$ inverse operation needed to be computed in both the forward and inverse AES SBox is the most computationally intensive and often implemented using composite fields, which we will next introduce.

B. Composite Fields

A composite field, represented by $GF((2^m)^k)$, is a finite field where the elements are polynomials with degree k that have coefficients in $GF(2^m)$. This composite field is isomorphic to the $GF(2^n)$, where $n = m \times k$. To construct such a composite field would require two monic irreducible polynomials $P(y)$ and $Q(z)$. The polynomial $P(y)$ is of degree k with coefficients in $GF(2^m)$, while $Q(z)$ is of degree m with coefficients in $GF(2)$.

For the AES field in $GF(2^8)$, the complex inverse operation required by *SBox* (1) can be computed more efficiently using an isomorphic field of the form $GF((2^m)^k)$ as follows:

$$a^{-1} = \phi^{-1}(\phi(a)^{-1}))$$

where ϕ is a map from $GF(2^8)$ to the composite field and ϕ^{-1} is the inverse map [8]. The maps ϕ and ϕ^{-1} are linear operations and can be interpreted as matrices. In the left hand side of the above equation, the inverse term a^{-1} is computed in $GF(2^8)$, while in the right hand side the inverse term $\phi(a)^{-1}$ is computed in the composite field. Similarly, the inverse operation in (2) is computed as follows:

$$\{M^{-1}(b+N)\}^{-1} = \phi^{-1}\{(\phi M^{-1})(b+N)\}^{-1}.$$

To construct a mapping (ϕ) between the $GF(2^n)$ field and a composite field in $GF((2^m)^k)$, we first map 0 in $GF(2^m)$ to 0 in $GF((2^m)^k)$. For the remaining elements, we pick a generator g_1 from the composite field and map it to a generator g_2 in the field $GF(2^n)$. Then we map the element g_1^i to g_2^i for all i $(1 \leq i \leq 255)$.

Arithmetic operations in the composite field along with ϕ and ϕ^{-1} depend heavily on the choice of the irreducible polynomials $P(y)$ and $Q(z)$. In this letter we evaluate various choices for these irreducible polynomials to identify the best polynomials for FPGA platforms.

C. FPGA Architecture

The LUT is the smallest programmable element in an FPGA. A typical 6:1 LUT, present in most modern Xilinx FPGAs, allows the user to define any 6-input Boolean function. The large granularity of LUTs provides several optimization opportunities. For instance, consider the hardware implementations of the following Boolean functions:

$$f : x = a_0a_1 + a_2a_3 + a_4a_5 \text{ and}$$
$$g : y = (a_0 + a_1)(a_2 + a_3)a_4 + a_2a_3. \quad (3)$$

In a standard library that supports two-input gates, function f requires a total of five gates (2 `OR` and 3 `AND`), while function g is larger requiring six gates (3 `OR` and 3 `AND`). In an FPGA on the other hand, since both functions have six or fewer inputs, each requires exactly one 6-input LUT to realize. Thus, we see that circuits are mapped differently in an FPGA compared to an ASIC.

Internally, each 6-input LUT comprises of a 64-bit storage that holds the output of the 2^6 possible input combinations. Thus, function g, which has only five inputs utilizes half (2^5 bits) of the storage in the LUT compared to function f, which has six inputs and fully utilizes the LUT. To reduce LUT underutilization, modern FPGAs, in fact, have the 64-bit storage split into two 32-bit components and a multiplexer at the output [9]. One of such LUTs can either implement a single 6-input Boolean function or two 5-input Boolean functions that have the same parameters.

III. Implementation of the AES SBox

In this section, we will discuss the composite field theory behind the construction of the AES SBox.

A. Composite Fields for the AES SBox

For the AES SBox, two forms of composite fields are often used in implementations: $GF((2^4)^2)$ and $GF(((2^2)^2)^2)$. In the former case, each element has the form $\alpha_1 y + \alpha_2$, where $\alpha_1, \alpha_2 \in GF(2^4)$. Computing an SBox operation would require mapping the input from $GF(2^8)$ to $GF((2^4)^2)$, performing operations in $GF(2^4)$ and mapping the result back into the AES field.

SBoxes implemented in the $GF(((2^2)^2)^2)$ field follow a similar approach. However, the $GF(2^4)$ operations are done in another isomorphic composite field namely $GF((2^2)^2)$. The inner composite field [i.e., $GF(2^2)$] performs arithmetic operations on two-bit numbers. In ASIC platforms, where cell libraries support a variety of two-input gates, this field would result in very efficient designs. For an FPGA on the other hand with much larger LUTs, the two-bit operations do not provide any special advantages. As an example, consider the multiplication operation in $GF(2^4)$ with the irreducible polynomial x^4+x+1. In a 6-input LUT-based FPGA, this operation, using a combinational multiplier, occupies seven 6-input LUTs. The

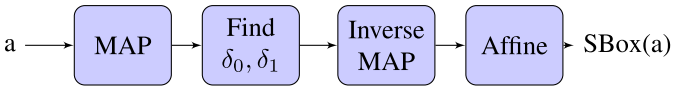

Fig. 1. High level view of an AES SBox implemented with composite fields.

$GF(2^4)$ multiplication can also be performed in the composite field $GF((2^2)^2)$. The best $GF((2^2)^2)$ multiplier considering all possibilities requires eleven 6-input LUTs, which is four more than doing the multiplication in $GF(2^4)$. Similar results are obtained for computing the inverse operations in $GF(2^4)$ as compared to $GF((2^2)^2)$.

For the rest of this letter, we, therefore, consider composite fields of the form $GF((2^4)^2)$ defined by two irreducible polynomials $P(y)$ and $Q(z)$. The polynomial $P(y)$ has the form $y^2 + \tau y + \nu$ with coefficients $\tau, \nu \in GF(2^4)$. Similarly, $Q(z) = z^4 + q_3 z^3 + q_2 z^2 + q_1 z^1 + q_0$ and has coefficients in $GF(2)$. There are only three possibilities for $Q(z)$ namely $z^4 + z + 1$, $z^4 + z^3 + 1$, and $z^4 + z^3 + z^2 + z + 1$. Now, for each choice of $Q(z)$, we need to find a $P(y)$ that is irreducible in the composite field $GF((2^4)^2)$. Notice that since τ and ν are in $GF(2^4)$ therefore each can take 16 values. Therefore, there are 256 possibilities for $P(y)$, of which we found 120 to be irreducible. Thus, each $Q(z)$ has a distinct set of 120 possibilities for each $P(y)$, giving a total of $3 \times 120 = 360$ different composite fields.

For every choice of $P(y)$ and $Q(z)$ there are eight distinct mappings between the fields. Thus, there are a total of $8 \times 360 = 2880$ different mappings. These maps are stored as simple linear operations between the input bits and the output bits.

B. High-Level View of Implementation

Fig. 1 shows the high-level view of the SBox. The input a, in the AES field is first mapped to the composite field $GF((2^4)^2)$. The latter defined by the irreducible polynomials $P(y)$ and $Q(z)$. The mapped element has the form $g = \gamma_1 y + \gamma_0$, where γ_1 and γ_0 are in $GF(2^4)$.

The inverse of $g \in GF((2^4)^2)$ is next computed. Let the inverse be denoted $d = \delta_1 y + \delta_0$ and irreducible polynomial $P(y)$ have the form $y^2 + \tau y + \nu$, where δ_1, δ_0, and τ are in $GF(2^4)$. It is not difficult to represent δ_1 and δ_0 in terms of γ_1 and γ_0 using the fact that $g \times d = 1$ [5]

$$\begin{aligned} \delta_1 &= (\gamma_1^2 \nu + \gamma_1 \gamma_0 \tau + \gamma_0^2)^{-1} \gamma_1 \\ \delta_0 &= (\gamma_1^2 \nu + \gamma_1 \gamma_0 \tau + \gamma_0^2)^{-1} (\gamma_0 + \gamma_1 \tau). \end{aligned} \tag{4}$$

Notice that all the operations here are in the inner field $GF(2^4)$ that is defined over the irreducible polynomial $Q(z)$. The choice of this polynomial plays a role in how efficiently the finite field operations are performed.

We next use the inverse map ϕ^{-1} to transform d in $GF((2^4)^2)$ to a^{-1} in GF(2^8). This is followed by the AES SBox affine transformation as shown in Fig. 1.

C. Implementation and Optimization of Operations

The mapping function ϕ from the AES field to the composite field can be represented by eight linear equations. This mapping theoretically would take 8 or more 6-input LUTs; atleast one LUT for each output bit. However, for an optimally chosen composite field, much fewer LUTs are required. For our best composite map, each linear equation is a function of about five input parameters. We find that this mapping takes five 6-input LUTs, fewer than expected. This becomes possible because the FPGA synthesis tool configures a 6-input LUT as two 5-input LUTs [9].

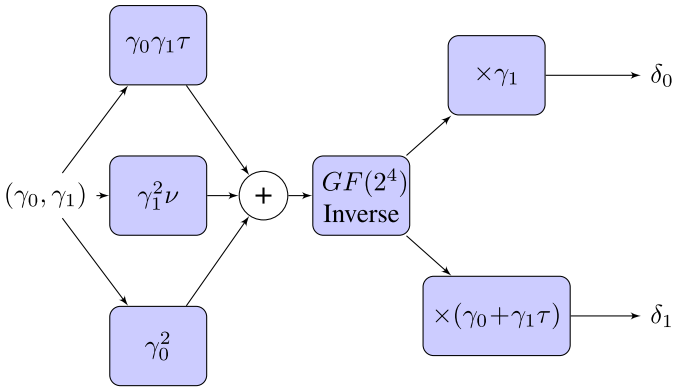

Fig. 2. Finding δ_1 and δ_0—The second step in Fig. 1. Each of these modules is optimized for LUT-based FPGA platforms.

Computing the inverse using composite fields requires the implementation of δ_0 and δ_1, which is defined by (4). In an FPGA, the typical way to implement these circuits is by representing each output bit as a function of the input bits. Further, due to the large granularity of the LUTs, multiple of these operations can be compressed into a single LUT. Our scheme is shown in Fig. 2. The module γ_0^2 takes a 4-bit input and generates a 4-bit output. On an FPGA, this would require just two 6-input LUTs, with each LUT configured as 5-input 2-output. Further, notice that in (4), since ν is constant for a given implementation, the module $\gamma_1^2 \nu$ can be similarly compressed into two 6-input LUTs. Likewise, since τ is a constant, the module $\gamma_1 \gamma_0 \tau$ requires the same resources as a single $GF(2^4)$ multiplier. The multiplication with τ comes for free. One of our two best SBox implementations, in fact, uses $\tau = 2$ on FPGA. However, in ASICs, an additional multiplier would be needed, which would add to the costs. To avoid this multiplication, Canright had kept $\tau = 1$ [5].

We can further optimize the affine and inverse mappings. Both these operations are linear and therefore can be combined. That is, in (1), we compute $M\phi^{-1}(d) + N$, where M and N denote the affine transformation and ϕ^{-1} is the inverse map. Since we also know M, N, and ϕ^{-1}, we can find the equations which relate the bits of the SBox output to the input bits in d. Since the SBox output is in $GF(2^8)$ there are eight such equations, each having at most eight parameters corresponding to the bits in d. We observed that the best SBox implementations has 4 LUTs similar to the forward mapping function.

IV. Results

Each option for the forward and inverse AES SBox was implemented in Verilog, synthesized using Xilinx Vivado 2017.1 for an Artix-7 FPGA. Fig. 3 shows the cumulative histograms for LUTs required by the 2880 forward and inverse

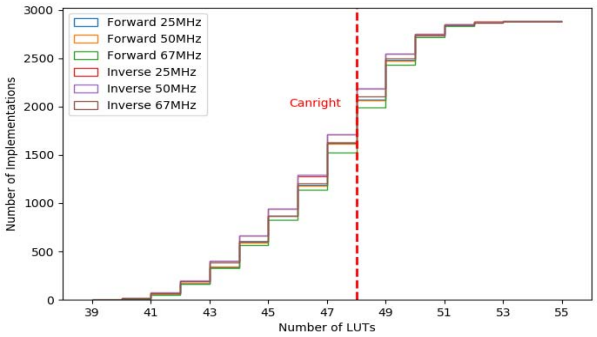

Fig. 3. Cumulative histograms of LUTs required for AES forward and inverse SBox implementations with different composite fields. Canright's SBox implementation [5] synthesized for the same FPGA platform is also shown.

TABLE I
COMPOSITE FIELDS THAT PROVIDE BEST SBOX IMPLEMENTATIONS. BOTH REQUIRE 72 mW AND HAVE A MAXIMUM SLACK OF 29 ns

SBox	Q(z)	P(y)	g_1	g_2	#LUTs
Forward	z^4+z+1	y^2+y+13	16	31	39
Inverse	z^4+z+1	y^2+y+13	16	31	40

SBox implementations, at various frequencies. We found multiple good implementations which provided equivalent results. Table I shows one such implementation.[1]

The state-of-the-art SBox that uses Canright's implementations requires 48 LUTs for both the forward and inverse SBoxes [5] when implemented in Verilog. As seen in Fig. 3 there are over 1500 SBox implementations better than Canright's. We obtain this advantage because Canright used composite fields of the form $GF(((2^2)^2)^2)$, which are ideally suited for 2-input gates typically found in ASICs [5]. However, these architectures are not suited for FPGAs with LUTs. On the other hand, our implementations are based on $GF((2^4)^2)$, which are better suited for FPGAs.

To identify where the gains were obtained, we synthesized each module of our best SBox independently and compared it with the corresponding modules from Canright's SBox implementation. We observe that many of the modules require fewer LUTs compared to Canright's. One aspect that works in our favor is that many of our $GF(2^4)$ functions have five or fewer parameters. This fits well with 6-input LUTs, which can be configured as two 5-input LUTs. Due to this feature, we were able to fit two functions in a single LUT. An example is the $\gamma_1^2\nu$ module which requires just one LUT to implement. Of the 4 output bits, 2 are same as the input, while the other two fits into a single LUT. Another example is the $GF(2^4)$ inversion circuit. Our implementation only requires 2 LUTs, while Canright's inversion circuit in $GF((2^2)^2)$ requires 7 LUTs.

V. DISCUSSION: APPLICATION TO THRESHOLD IMPLEMENTATIONS

A threshold implementation of an AES SBox is used to protect against side-channel attacks [1]. It achieves provable security against side-channel attacks by splitting the inputs into various shares, such that none of the component functions operating on the shares reveal the input entirely.

[1]The complete results can be accessed from https://bitbucket.org/casl/aes_sbox.

TABLE II
COMPARISON OF NUMBER OF LUTS REQUIRED FOR UNPROTECTED AND THRESHOLD IMPLEMENTATIONS OF THE AES SBOX WITH $Q(z)=z^4+z+1$ AND $P(y)=y^2+Ay+B$

	#LUTs				
A, B	**1,13**	1,10	1,11	10,10	5,5
Unprotected	**39**	40	42	45	48
Protected	**219**	222	224	230	233

Efficient threshold implementations have structures similar to Fig. 2, with each $GF(2^4)$ multiplication implemented with four shares and the $GF(2^4)$ inverse implemented with five shares. The overhead incurred due to the threshold implementation is almost invariant to the choice of composite field as it depends only on the choice of shares. Adapting the implementation from [1], we find that the composite fields used in our smallest unprotected SBox (39 LUTs) also results in the smallest threshold implementation, which requires 219 LUTs. Table II compares threhold implementations our smallest SBox with a few other randomly chosen SBox implementations.

VI. CONCLUSION

This letter presents composite field AES SBox implementations that are suited for LUT-based FPGAs. The composite field SBox designs developed in this letter specifically for FPGAs, would be useful for constructing efficient threshold-based side-channel resistant SBox implementations. The designs exploit the large LUTs, ubiquitous in modern FPGAs. Our results show a saving of 9 and 8 LUTs for the forward and inverse SBox implementations, respectively, compared to the state-of-the-art composite field implementation. These savings are likely to increase as the LUT size increases. Further improvements can be achieved if normal basis representations are also considered. This can be done in the future.

REFERENCES

[1] B. Bilgin, B. Gierlichs, S. Nikova, V. Nikov, and V. Rijmen, "A more efficient AES threshold implementation," in *Proc. AFRICACRYPT*, 2014, pp. 267–284.
[2] J. Boyar and R. Peralta, "A small depth-16 circuit for the AES S-box," in *Proc. IFIP Int. Inf. Security Conf.*, Jun. 2012, pp. 287–298.
[3] P. Bulens, F.-X. Standaert, J.-J. Quisquater, P. Pellegrin, and G. Rouvroy, "Implementation of the AES-128 on Virtex-5 FPGAs," in *Proc. AFRICACRYPT*, 2008, pp. 16–26.
[4] D. Canright and L. Batina, "A very compact 'perfectly masked' S-box for AES," in *Proc. 6th Int. Conf. Appl. Cryptography Netw. Security (ACNS)*, 2008, pp. 446–459.
[5] D. Canright, "A very compact S-box for AES," in *Cryptographic Hardware and Embedded Systems—CHES 2005*. Heidelberg, Germany: Springer, 2005.
[6] *Federal Information Processing Standards Publication FIPS-PUB-197*, Announcing the Advanced Encryption Standard, 2001.
[7] A. Moradi, A. Poschmann, S. Ling, C. Paar, and H. Wang, "Pushing the limits: A very compact and a threshold implementation of AES," in *Proc. EUROCRYPT*, 2011, pp. 69–88.
[8] C. Paar, "Efficient VLSI architectures for bit-parallel computation in Galois fields," Ph.D. dissertation, Inst. Exp. Math., Universität Duisburg-Essen, Duisburg, Germany, Jun. 1994.
[9] *7 Series FPGAs Configurable Logic Block User Guide*, XILINX, San Jose, CA, USA, Sep. 2016.