**Noname manuscript No.**
(will be inserted by the editor)

# Scrutinizing the Tower Field Implementation of the $\mathbb{F}_{2^8}$ Inverter – with Applications to AES, Camellia, and SM4

**Zihao Wei**[1,2,3] · **Siwei Sun**[1,2,3] * · **Lei Hu**[1,2,3] · **Man Wei**[1,2,3] · **Joan Boyar**[4] · **René Peralta**[5]

Received: date / Accepted: date

**Abstract** The tower field implementation of the $\mathbb{F}_{2^8}$ inverter is not only the key technique for compact implementations of the S-boxes of several internationally standardized block ciphers such as AES, Camellia, and SM4, but also the underlying structure many side-channel attack resistant AES implementations rely on. In this work, we conduct an exhaustive study of the tower field representations of the $\mathbb{F}_{2^8}$ inverter with normal bases by applying several state-of-the-art combinatorial logic minimization techniques. As a result, we achieve improved implementations of the AES, Camellia and SM4 S-boxes in terms of area footprint. Surprisingly, we are still able to improve the currently known most compact implementation of the AES S-box from CHES 2018 by 5.5 `GE`, beating the record again (Excluding this work, the latest improvement was proposed at CHES 2018, which achieves 11.75 `GE` improvement over the optimal implementation at the time). For Camellia and SM4, the improvements are even more significant. The Verilog codes of our implementations of the AES, Camellia and SM4 S-boxes are openly available.

**Keywords** Tower field · Inverter · S-box · AES · Camellia · SM4

## 1 Introduction

For encrypting and authenticating the largest part of the workload of today's secure communication, symmetric-key primitives are regarded as the crypto workhorse (whereas public-key schemes are generally used for setting up the session keys). In many cases, the components of symmetric-key schemes are built on operations over finite fields. Since the symmetric-key cryptographic algorithms will eventually be implemented in software or hardware to play a role in the real world, it is of great importance to investigate how to implement their common operations

* The corresponding author

1. State Key Laboratory of Information Security, Institute of Information Engineering, CAS
2. Data Assurance and Communication Security Research Center, CAS
3. School of Cyber Security, University of Chinese Academy of Sciences
4. Department of Mathematics and Computer Science, University of Southern Denmark
5. Information Technology Laboratory, NIST, USA

efficiently [6,9,18,30]. For instance, due to the rapid development of lightweight IoT devices, ongoing efforts have been made to obtain more compact ASIC implementations of symmetric-key ciphers [4,5,17]. Just recently, the most compact implementation of the MixColumns and the S-box of AES were reported at FSE 2018 [19] and CHES 2018 [26] respectively.

In this work, we focus on area-optimized implementations of the multiplicative inverse operation (and its affine equivalences) over $\mathbb{F}_{2^8}$. The AES S-box, which is affine equivalent to the $\mathbb{F}_{2^8}$ inverter, is the strongest $8 \times 8$ S-box known so far in terms of local security properties (*i.e.*, non-linearity, differential uniformity, algebraic degree, etc.). Several internationally standardized block ciphers, such as Camellia and SM4, apply variants of the AES S-box in their designs, which are all affine equivalent to the $\mathbb{F}_{2^8}$ inverter. Despite its desirable local cryptographic properties, to implement the AES S-box in ASIC with small footprint is not a trivial task. The naive approach that encodes the AES S-box as a look-up table in a hardware description language and produces the actual circuit relying on open-source or commercial CAD tools will certainly lead to unsatisfactory results for many resource constrained applications. Today's most compact ASIC implementations of the AES S-box are based on the tower field architecture, where the operations over $\mathbb{F}_{2^{2k}}$ are represented with operations over $\mathbb{F}_{2^k}$ recursively.

Moreover, several cost-effective threshold implementations of the AES S-box with resistance against side-channel attacks are built on top of the tower field architecture [24,8,12,7]. In threshold implementations, the most important design consideration includes the security level, number of fresh random bits required, and area consumption. Therefore, providing different implementations of the tower field structure without increasing the circuit footprint potentially offers more flexible area-randomness-security trade-off in threshold implementations.

Apart from these, breaking the AES S-box into several layers with the tower field architecture allows registers to be inserted into the middle of the computation such that the critical path can be reduced, and therefore the frequency of the system clock can be increased to boost the performance.

**Related work.** The tower field architecture was first proposed by Itoh and Tsujii for computing multiplicative inverse in finite fields of characteristic two [16]. At the beginning, it was applied in developing efficient implementations of public-key cryptographic algorithms involving inverse operations over $\mathbb{F}_{2^n}$ [15,25]. Later, after the development of the Advanced Encryption Standard (AES) – a block cipher using an S-box affine equivalent to $\mathbb{F}_{2^8}$ multiplicative inverter [13], the tower field architecture found applications in compact hardware implementations of AES [27,29,23,32]. After a series of improvements, Canright [11,10] and Boyar et al. [9,31] achieved the most compact implementations at the time, which has become the *de facto* standard for compact implementations of the AES S-box. Such tower field implementations of AES S-box were also intensively applied in side-channel resistant implementations of AES to reduce resource consumption. Recently Reyhani-Masoleh et al. [26] broke the record set by Canright and Boyar et al., presenting so far the most compact ASIC implementation of the AES S-box, which costs 182.25 `GE` under the `STM` 65`nm` CMOS technology.

Due to the strong local cryptographic properties of the AES S-box, several well known block ciphers employ affine equivalences of the AES S-box as their S-boxes,

including Camellia and SM4. Therefore, the technique of tower field implementation naturally applies to these ciphers [28,21,22,3,1].

In tower field implementations, a sequence of field extensions starting from $\mathbb{F}_2$ and ending at $\mathbb{F}_{2^8}$ of the type $\mathbb{F}_{2^k} \subseteq \mathbb{F}_{(2^k)^{2^l}}$ is considered. At each level of the field extension, an irreducible polynomial over $\mathbb{F}_{2^k}$ and a corresponding basis of $\mathbb{F}_{(2^k)^{2^l}}$ over $\mathbb{F}_{2^k}$ have to be specified. The irreducible polynomials and bases induce a basis of $\mathbb{F}_{2^8}$ over $\mathbb{F}_2$. The tower field architecture is implemented over this new basis with proper basis transformations to maintain the original field representation. Therefore, the choices of the field extensions, the corresponding irreducible polynomials and the bases determine the overall cost of the implementation. A summary of the choices of existing work is given in Table 1 for AES, Camellia and SM4 respectively.

Table 1: Previous tower field implementations of the AES, Camellia and SM4 S-boxes, where $\mathcal{P}$ means a polynomial basis is used, $\mathcal{N}$ means a normal basis is used, and #Cases denotes the number of cases considered.

| Cipher | Source | Tower field architecture and basis | #Cases |
|---|---|---|---|
| AES | [27][32] | $\mathbb{F}_2 \xrightarrow{w^4+w+1} \mathbb{F}_{2^4} \xrightarrow[\mathcal{P}]{y^2+y+C_1} \mathbb{F}_{2^8}$ | 1 |
| | [29] | $\mathbb{F}_2 \xrightarrow[\mathcal{P}]{w^2+w+1} \mathbb{F}_{2^2} \xrightarrow[\mathcal{P}]{z^2+z+C_2} \mathbb{F}_{2^4} \xrightarrow[\mathcal{P}]{y^2+y+C_3} \mathbb{F}_{2^8}$ | 1 |
| | [23] | $\mathbb{F}_2 \xrightarrow[\mathcal{P}]{w^2+w+1} \mathbb{F}_{2^2} \xrightarrow[\mathcal{P}]{z^2+z+C_2} \mathbb{F}_{2^4} \xrightarrow[\mathcal{P}]{y^2+y+\nu} \mathbb{F}_{2^8}$ | 64 |
| | [11] | $\mathbb{F}_2 \xrightarrow[\mathcal{P}/\mathcal{N}]{w^2+w+1} \mathbb{F}_{2^2} \xrightarrow[\mathcal{P}/\mathcal{N}]{z^2+z+N} \mathbb{F}_{2^4} \xrightarrow[\mathcal{P}/\mathcal{N}]{y^2+y+\nu} \mathbb{F}_{2^8}$ | 432 |
| | [9] | $\mathbb{F}_2 \xrightarrow[\mathcal{N}]{w^2+w+1} \mathbb{F}_{2^2} \xrightarrow[\mathcal{N}]{z^2+z+C_4} \mathbb{F}_{2^4} \xrightarrow[\mathcal{N}]{y^2+y+C_5} \mathbb{F}_{2^8}$ | 1 |
| | [26] | $\mathbb{F}_2 \xrightarrow[\mathcal{N}]{w^4+w^3+w^2+w+1} \mathbb{F}_{2^4} \xrightarrow[\mathcal{N}]{y^2+y+\nu} \mathbb{F}_{2^8}$ | 32 |
| Camellia | [28] | $\mathbb{F}_2 \xrightarrow{w^4+w+1} \mathbb{F}_{2^4} \xrightarrow[\mathcal{P}]{y^2+y+C_6} \mathbb{F}_{2^8}$ | 1 |
| | [21] | $\mathbb{F}_2 \xrightarrow[\mathcal{P}/\mathcal{N}]{w^4+w+1} \mathbb{F}_{2^4} \xrightarrow[\mathcal{P}/\mathcal{N}]{y^2+\tau y+\nu} \mathbb{F}_{2^8}$<br>$\mathbb{F}_2 \xrightarrow[\mathcal{P}/\mathcal{N}]{w^2+w+1} \mathbb{F}_{2^2} \xrightarrow[\mathcal{P}/\mathcal{N}]{z^2+Tz+N} \mathbb{F}_{2^4} \xrightarrow[\mathcal{P}/\mathcal{N}]{y^2+\tau y+\nu} \mathbb{F}_{2^8}$ | 13 |
| SM4 | [22] | $\mathbb{F}_2 \xrightarrow[\mathcal{P}/\mathcal{N}]{w^4+w+1} \mathbb{F}_{2^4} \xrightarrow[\mathcal{P}/\mathcal{N}]{y^2+C_7y+C_8} \mathbb{F}_{2^8}$ | 4 |
| | [3] | $\mathbb{F}_2 \xrightarrow[\mathcal{P}]{w^2+w+1} \mathbb{F}_{2^2} \xrightarrow[\mathcal{P}]{z^2+z+C_9} \mathbb{F}_{2^4} \xrightarrow[\mathcal{P}]{y^2+y+C_{10}} \mathbb{F}_{2^8}$ | 1 |
| | [1] | $\mathbb{F}_2 \xrightarrow[\mathcal{N}]{w^2+w+1} \mathbb{F}_{2^2} \xrightarrow[\mathcal{N}]{z^2+z+N} \mathbb{F}_{2^4} \xrightarrow[\mathcal{N}]{y^2+y+\nu} \mathbb{F}_{2^8}$ | 16 |

**Contributions.** As shown in Table 1, only a part of the design space of tower field implementation was explored by choosing irreducible polynomials of special forms in previous work. In particular, previous work preferred a class of parameter choices where the irreducible polynomials selected for the field extension $\mathbb{F}_{2^2} \subseteq \mathbb{F}_{2^4} \subseteq \mathbb{F}_{2^8}$ are of the form $z^2 + z + N$ and $y^2 + y + \nu$, and indeed the most well known implementations of Canright's and Boyar et al.'s schemes are in this class [11,9]. Although some work considered other irreducible polynomials, no systematic investigation was conducted [21]. The preference for this special class

is reasonable, since with these choices of parameters, the implementations of some subcomponents of the circuit are free. Despite this heuristic, there is no concrete evidence that this configuration will result in optimal implementations. Therefore, we exhaustively examine all possible tower field representations under normal bases induced by irreducible polynomials (720 cases in total), and find several cases which are never considered previously enjoy the most compact implementations. Along the way, we do not only apply well-known logic minimization techniques from Canright and Boyar et al., but also resort to several state-of-the-art combinatorial logic minimization techniques [14,30,18] developed in recent years. As a result, we beat the new record set by the work of Reyhani-Masoleh et al. [26] for compact implementations of the AES S-box. Moreover, the implementations of the Camellia and SM4 S-boxes are improved significantly, and we refer readers to Table 2 for a summary of the results. Naturally, these results serve to achieve more compact implementations of AES, Camellia and SM4, and potentially provide more flexible security-randomness-area trade-offs for threshold implementations of these block ciphers. The Verilog codes of our implementation of the AES, Camellia and SM4 S-boxes are provided in the Appendix.

**Organization.** In Section 2, we give a brief introduction of the mathematical background of the tower field representations under different bases, as well as the frequently-used logic gates for constructing digital circuits. Subsequently, we describe the details of the tower field implementation of the $\mathbb{F}_{2^8}$ inverter in Section 3. In Section 4, we apply state-of-the-art logic minimization techniques to a list of tower filed representations of the AES, Camellia and SM4 S-box under all possible normal bases. As a result, we obtain so far the most compact implementations of these S-boxes. We conclude the paper in Section 5 and propose possible future work. The source codes of the optimized implementations for the S-boxes of AES, Camellia and SM4 are provided in the Appendix.

## 2 Preliminaries

We first give a brief introduction of the tower field representation. Then we list a set of gates together with their functionalities and areas. These gates will be used to implement the circuits constructed in this paper, and the overall area of each circuit will be computed accordingly.

**Tower field representation.** Let $\mathbb{F}_2 = \{0, 1\}$ be the finite field of two elements. It is well known that the field $\mathbb{F}_{2^k}$ with $2^k$ elements can be induced by an irreducible polynomial $q(x) \in \mathbb{F}_2[x]$ with degree $k$, i.e., $\mathbb{F}_{2^k} \cong \mathbb{F}_2[x]/(q(x))$. Assuming that $X$ is a root of $q(x)$ over $\mathbb{F}_{2^k}$, then every element in $\mathbb{F}_{2^k}$ can be represented as an $\mathbb{F}_2$-linear combination $b_{k-1}X^{k-1} + \cdots + b_1X + b_0$ of $[X^{k-1}, \cdots, X, X^0]$, which is a *polynomial basis* of $\mathbb{F}_{2^k}$ over $\mathbb{F}_2$. To be concrete, we take $k = 8$, and we call $(b_7, \cdots, b_0)$ the bit-vector representation of $b_{k-1}X^{k-1} + \cdots + b_1X + b_0$ under the basis $[X^7, \cdots, X, X^0]$.

Considering a sequence of field extensions $\mathbb{F}_2 \subseteq \mathbb{F}_{2^2} \subseteq \mathbb{F}_{2^4} \subseteq \mathbb{F}_{2^8}$ shown in Fig. 1. Let $r(y) \in \mathbb{F}_{2^4}[y]$, $s(z) \in \mathbb{F}_{2^2}[z]$ and $t(w) \in \mathbb{F}_2[w]$ be irreducible polynomials over their respective fields, and let $Y \in \mathbb{F}_{2^8}$, $Z \in \mathbb{F}_{2^4}$ and $W \in \mathbb{F}_{2^2}$ be roots of $r(y)$, $s(z)$ and $t(w)$ over the corresponding fields respectively. Then we obtain a set of normal basis: $[Y^{16}, Y]$ is a basis of $\mathbb{F}_{2^8}$ over $\mathbb{F}_{2^4}$, $[Z^4, Z]$ is a basis of

Table 2: Synthesized Results, where the functionalities of some uncommon gates (e.g., `XOR3`, `OAI21`, `AOI21` etc.) are described in Section 2.

| Cipher | Source | Gates used | | | | | | | | | | Synthesis Results | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | XOR/XNOR | XOR3 | NAND | AND | NAND3 | NOR | NOT | OAI21 | AOI21 | OAI32 | SMIC 130nm | SMIC 65nm | STM 65nm | Nangate 45nm |
| AES | [27] | 111 | | | 58 | | | | | | | 336.33 | 336.75 | 294.50 | 299.33 |
| | [29] | 100 | | | 36 | | | | | | | 281.33 | 279.00 | 245.00 | 248.00 |
| | [23] | 96 | | | 36 | | | | | | | 272.00 | 270.00 | 237.00 | 240.00 |
| | [11] | 80 | | 34 | | | 6 | | | | | 226.67 | 220.00 | 200.00 | 200.00 |
| | [9] | 83 | | | 32 | | | | | | | 236.33 | 234.75 | 206.00 | 208.67 |
| | [31] | 81 | | | 32 | | | | | | | 231.67 | 230.25 | 202.00 | 204.67 |
| | | 81 | | 32 | | | | | | | | 221.00 | 214.25 | 194.00 | 194.00 |
| | [26] | 69 | | 39 | | 4 | 3 | 4 | | | | 211.00 | 205.25 | 188.00 | 188.00 |
| | | 69 | | 31 | | | 3 | 5 | 7 | 1 | | N/A | N/A | N/A | 186.00 |
| | | 63 | 3 | 27 | | | 7 | 4 | | | 4 | N/A | N/A | 182.25.00 | N/A |
| | Ours | 69 | | 33 | | | 8 | | | | | **202.00** | **196.25** | 179.00 | **179.00** |
| | | 51 | 9 | 33 | | | 8 | | | | | N/A | N/A | **176.75** | N/A |
| Camellia | [21] | 113 | | | 35 | | | 9 | | | | 316.33 | 313.50 | 276.50 | 278.67 |
| | Ours | 68 | | 33 | | | 8 | 1 | | | | **200.33** | **194.75** | **177.75** | **177.67** |
| SM4 | [22] | 99 | | | 58 | | | 11 | | | | 315.67 | 318.00 | 278.75 | 282.67 |
| | [3] | 157 | | | 63 | | | | | | | 450.33 | 447.75 | 392.75 | 398.00 |
| | [1] | 134 | | | 36 | | | | | | | 360.67 | 355.50 | 313.00 | 316.00 |
| | Ours | 66 | | 32 | | | 9 | 1 | | | | **195.67** | **190.25** | **173.75** | **173.67** |

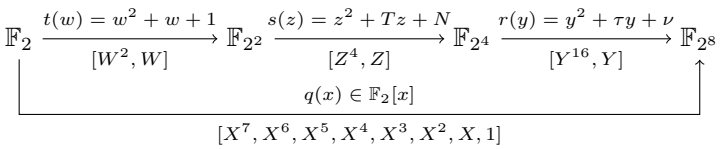


Fig. 1: The tower field structure

$\mathbb{F}_{2^4}$ over $\mathbb{F}_{2^2}$, and $[W^2, W]$ is a basis of $\mathbb{F}_{2^2}$ over $\mathbb{F}_2$. Therefore, for an element $b = b_7X^7 + \cdots + b_1X + b_0 \in \mathbb{F}_{2^8}$ we have

$$b = \gamma_1 Y^{16} + \gamma_0 Y, \gamma_1, \gamma_0 \in \mathbb{F}_{2^4},$$

$$\gamma_1 = \Gamma_3 Z^4 + \Gamma_2 Z, \gamma_0 = \Gamma_1 Z^4 + \Gamma_0 Z, \Gamma_3, \Gamma_2, \Gamma_1, \Gamma_0 \in \mathbb{F}_{2^2},$$

$$\Gamma_3 = g_7 W^2 + g_6 W, \Gamma_2 = g_5 W^2 + g_4 W, \Gamma_1 = g_3 W^2 + g_2 W, \Gamma_0 = g_1 W^2 + g_0 W,$$

$$g_i \in \mathbb{F}_2, 0 \leq i \leq 7,$$

which implies $b = b_7X^7 + \cdots + b_1X + b_0 = g_7W^2Z^4Y^{16} + g_6WZ^4Y^{16} + g_5W^2ZY^{16} + g_4WZY^{16} + g_3W^2Z^4Y + g_2WZ^4Y + g_1W^2ZY + g_0WZY$. That is, $b$ can be represented as $(g_7, \cdots, g_0)$ under the *tower basis*

$$\mathcal{TB} = [W^2Z^4Y^{16}, WZ^4Y^{16}, W^2ZY^{16}, WZY^{16}, W^2Z^4Y, WZ^4Y, W^2ZY, WZY]$$

induced by $W$, $Z$ and $Y$. We call $(g_7, \cdots, g_0)$ the bit-vector representation of $b$ under the tower basis. Assuming that the tower basis $\mathcal{TB}$ can be represented by the original polynomial basis with a matrix $M_t \in \mathbb{F}_2^{8\times 8}$ as

$$\mathcal{TB} = (X^7, \cdots, X^0)M_t,$$

we have

$$(b_7, \cdots, b_0)^\mathsf{T} = M_t \cdot (g_7, \cdots, g_0)^\mathsf{T} \quad \text{or} \quad (g_7, \cdots, g_0)^\mathsf{T} = M_t^{-1} \cdot (b_7, \cdots, b_0)^\mathsf{T}.$$

Therefore, we can change the representations by multiplying $M_t$ or $M_t^{-1}$, and we call $M_t$ the *basis transformation matrix*.

Considering the example from AES shown in Fig. 1, where $q(x)$ is the Rijdael polynomial $x^8+x^4+x^3+x+1$, $\tau = X^7+X^5+X^4+X^3+X^2+1$, $\nu = X^7+X^6+X^5$, $T = X^7 + X^5 + X^4 + X^3 + X^2 + 1$, $N = 1$, $W = X^7 + X^5 + X^4 + X^3 + X^2$, $Z = X^6 + X^4$ and $Y = X^6 + X^3$. Then we have $\mathcal{TB} = (X^7, \cdots, X^0) \cdot M_t$, where

$$M_t = \begin{pmatrix} 1&1&1&0&0&0&0&0 \\ 1&0&0&1&0&1&1&0 \\ 1&0&1&1&0&1&0&1 \\ 0&1&0&0&0&1&0&1 \\ 0&0&0&0&0&1&0&0 \\ 1&0&1&1&1&1&1&1 \\ 0&1&1&1&0&1&1&1 \\ 1&0&1&1&1&1&0&1 \end{pmatrix}.$$

**Frequently-used gates.** The circuits of this paper are eventually synthesized with the gates provided in common cell libraries. We list a set of frequently-used gates in Table 3, where the area is measured in gate equivalence (GE), corresponding to the area of a two-input drive-strength-one NAND gate.

Note that apart from those common gates (XOR, XNOR, AND, NAND, OR, NOR, NOT) which are available in almost all CMOS technology libraries, we also list some compound gates (XOR3, NAND3, OAI21, AOI21, OAI32).

The data of STM 65nm library is collected from Reyhani-Masoleh et al.'s paper [26], while the others comes from library files and databooks. The cell in blank of STM 65nm means the corresponding gate does not appear at [26], and the cell labeled as N/A means the library does not support this kind of gate.

Table 3: Frequently-used gates in common CMOS technology libraries.

| Gate | Area (GE) | | | |
|---|---|---|---|---|
| | SMIC 130nm | SMIC 65nm | STM 65nm | Nangate 45nm |
| XOR: $(a, b) \mapsto a \oplus b$ | 2.33 | 2.25 | 2 | 2 |
| XNOR: $(a, b) \mapsto a \odot b$ | 2.33 | 2.25 | 2 | 2 |
| XOR3: $(a, b, c) \mapsto a \oplus b \oplus c$ | 5.67 | 4.75 | 3.75 | N/A |
| AND: $(a, b) \mapsto a \cdot b$ | 1.33 | 1.5 | 1.25 | 1.33 |
| NAND: $(a, b) \mapsto \overline{a \cdot b}$ | 1 | 1 | 1 | 1 |
| NAND3: $(a, b, c) \mapsto \overline{a \cdot b \cdot c}$ | 1.33 | 1.25 | 1.25 | 1.33 |
| OR: $(a, b) \mapsto a \mid b$ | 1.33 | 1.5 | 1.25 | 1.33 |
| NOR: $(a, b) \mapsto \overline{a \mid b}$ | 1 | 1 | 1 | 1 |
| NOT: $a \mapsto \overline{a}$ | 0.66 | 0.75 | 0.75 | 0.66 |
| OAI21: $(a, b, c) \mapsto \overline{(a \mid b) \cdot c}$ | 1.67 | 1.5 | | 1.33 |
| AOI21: $(a, b, c) \mapsto \overline{(a \cdot b) \mid c}$ | 1.67 | 1.5 | | 1.33 |
| OAI32: $(a, b, c, d, e) \mapsto \overline{(a \mid b \mid c) \cdot (d \mid e)}$ | 2.33 | N/A | 2 | N/A |

## 3 Tower Field Implementation of the $\mathbb{F}_{2^8}$ Inverter

In this subsection, we give an introduction to the tower field implementation of the $\mathbb{F}_{2^8}$ inverter. Please note that the derivation of these results can all be found in Canright's paper [11,10].

Consider the field extension $\mathbb{F}_{2^4} \subseteq \mathbb{F}_{2^8}$ with an irreducible polynomial $r(y) = y^2 + \tau y + \nu \in \mathbb{F}_{2^4}[y]$. Let $Y \in \mathbb{F}_{2^8}$ be a root of $r(y)$. Then $Y^{16}$ and $Y$ form a normal basis, and every element $G \in \mathbb{F}_{2^8}$ can be represented as $G = \gamma_1 Y^{16} + \gamma_0 Y$, where $\gamma_1$, $\gamma_0 \in \mathbb{F}_{2^4}$. Let $G^{-1} = \delta_1 Y^{16} + \delta_0 Y$ with $\delta_1$, $\delta_0 \in \mathbb{F}_{2^4}$ be the inverse of $G$. By Solving the equation

$$G \cdot G^{-1} = (\gamma_1 Y^{16} + \gamma_0 Y)(\delta_1 Y^{16} + \delta_0 Y) = 1$$

for $\delta_1$ and $\delta_0$, we obtain

$$\delta_1 = [\gamma_1 \gamma_0 \tau^2 + (\gamma_1 + \gamma_0)^2 \nu]^{-1} \gamma_0$$
$$\delta_0 = [\gamma_1 \gamma_0 \tau^2 + (\gamma_1 + \gamma_0)^2 \nu]^{-1} \gamma_1.$$

Therefore, given $r(y)$ and the basis $[Y^{16}, Y]$, we can compute $G^{-1} = (\delta_1, \delta_0)$ from $G = (\gamma_1, \gamma_0)$ using operations over $\mathbb{F}_{2^4}$, which is illustrated in Fig. 2, where $\phi = \gamma_1\gamma_0\tau^2 + (\gamma_1 + \gamma_0)^2\nu$ and $\lambda = \phi^{-1}$.

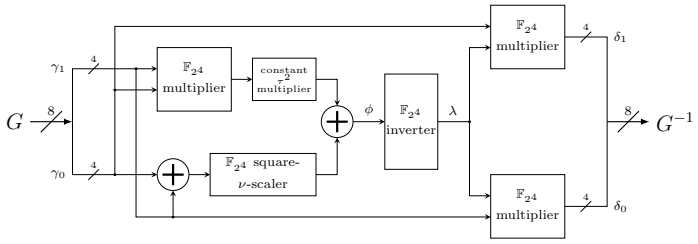


Fig. 2: The $\mathbb{F}_{2^8}$ inverter

**Multiplication and Inverse over** $\mathbb{F}_{2^4}$ Extend $\mathbb{F}_{2^2}$ to $\mathbb{F}_{2^4}$ with an irreducible polynomial $s(z) = z^2 + Tz + N \in \mathbb{F}_{2^2}[z]$, and let $Z \in \mathbb{F}_{2^4}$ be a root of $s(z)$. Then every element in $\mathbb{F}_{2^4}$ can be represented as an $\mathbb{F}_{2^2}$-linear combination of the normal basis $[Z^4, Z]$. Let $\gamma = \Gamma_1 Z^4 + \Gamma_0 Z$, and $\lambda = \Lambda_1 Z^4 + \Lambda_0 Z$, where $\Gamma_i$, $\Lambda_j \in \mathbb{F}_{2^2}$. Then the multiplication of $\gamma$ and $\lambda$ can be calculated as

$$\begin{aligned} \gamma\lambda = &\ (\Gamma_1 Z^4 + \Gamma_0 Z)(\Lambda_1 Z^4 + \Lambda_0 Z) \\ = &\ [\Gamma_1\Lambda_1 T + (\Gamma_1 + \Gamma_0)(\Lambda_1 + \Lambda_0)NT^2]Z^4 + \\ &\ [\Gamma_0\Lambda_0 T + (\Gamma_1 + \Gamma_0)(\Lambda_1 + \Lambda_0)NT^2]Z, \end{aligned} \tag{1}$$

which is illustrated in Figure 3.

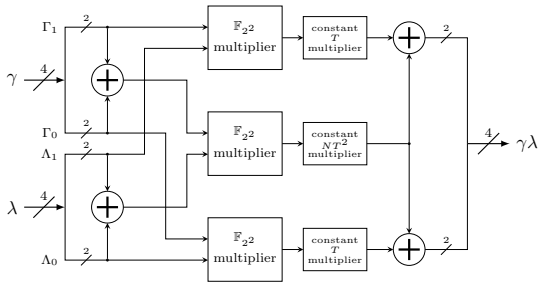


Fig. 3: The $\mathbb{F}_{2^4}$ multiplier

Let $\phi = \Phi_1 Z^4 + \Phi_0 Z$ with $\Phi_i \in \mathbb{F}_{2^2}$. It can be shown that the inverse $\phi^{-1}$ of $\phi$ is

$$[\Phi_1\Phi_0 T^2 + (\Phi_1 + \Phi_0)^2 N]^{-1}\Phi_0 Z^4 + [\Phi_1\Phi_0 T^2 + (\Phi_1 + \Phi_0)^2 N]^{-1}\Phi_1 Z, \tag{2}$$

whose circuit is depicted in Figure 4.

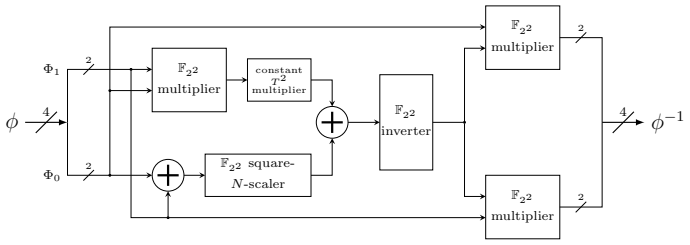


Fig. 4: The $\mathbb{F}_{2^4}$ inverter

**Multiplication and Inverse over $\mathbb{F}_{2^2}$.** Consider the field extension $\mathbb{F}_2 \subseteq \mathbb{F}_{2^2}$ with irreducible polynomial $t(w) = w^2 + w + 1 \in \mathbb{F}_2[w]$ (the only irreducible polynomial in $\mathbb{F}_2[w]$). Let $W$ be a root of $t(w)$ over $\mathbb{F}_{2^2}$. Then every element $\Gamma \in \mathbb{F}_{2^2}$ can be represented as an $\mathbb{F}_2$-linear combination $\Gamma = u_1W^2 + u_0W$ of the normal basis $[W^2, W]$, with $u_i \in \mathbb{F}_2$. Let $\Delta = v_1W^2 + v_0W$ with $v_j \in \mathbb{F}_2$ be another element in $\mathbb{F}_{2^2}$. The multiplication is given by

$$\begin{aligned} \Gamma\Delta = &\ (u_1W^2 + u_0W)(v_1W^2 + v_0W) \\ = &\ [u_1 \cdot v_1 \oplus (u_1 \oplus u_0) \cdot (v_1 \oplus v_0)]W^2 + \\ &\ [u_0 \cdot v_0 \oplus (u_1 \oplus u_0) \cdot (v_1 \oplus v_0)]W, \end{aligned} \tag{3}$$

whose implementation is shown in Figure 5. In addition, if $\Gamma\Delta = 1$, it can be shown that $v_1 = u_0$ and $v_0 = u_1$. That is, the $\mathbb{F}_{2^2}$ inverter can be implemented by swapping the two 1-bit input signals, which is free.

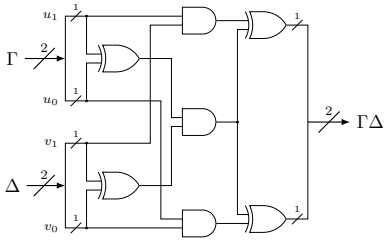


Fig. 5: The $\mathbb{F}_{2^2}$ multiplier

**Remark.** Finally, we would like to mention another two formulas which are useful later:

$$\begin{aligned} \Gamma\Delta \cdot W &= (u_1 \cdot v_1 \oplus u_0 \cdot v_0)W^2 + [u_1 \cdot v_1 \oplus (u_1 \oplus u_0) \cdot (v_1 \oplus v_0)]W \\ \Gamma\Delta \cdot W^2 &= [u_0 \cdot v_0 \oplus (u_1 \oplus u_0) \cdot (v_1 \oplus v_0)]W^2 + (u_1 \cdot v_1 \oplus u_0 \cdot v_0)W. \end{aligned} \tag{4}$$

According to Equation 4, the implementation cost of a multiplication followed with a $W$ (or $W^2$) scaler is the same as that of the multiplication $\Gamma\Delta$, which requires 4 `XOR` gates and 3 `AND` gates.

## 4 Applications to the S-boxes of AES, Camellia, and SM4

The S-boxes of AES, Camellia, and SM4 are all affine equivalent to the $\mathbb{F}_{2^8}$ inverter, which can be unified into the following form

$$S(b) = M_2 \cdot I^q_{\mathcal{PB}}(M_1 \cdot b \oplus C_1) \oplus C_2,\ b \in \mathbb{F}_{2^8},$$

where $M_1$, $M_2$ are $8 \times 8$ matrices over $\mathbb{F}_2$, $C_1$, $C_2$ are constant column vectors in $\mathbb{F}_2^8$, and $I^q_{\mathcal{PB}} : \mathbb{F}_2^8 \to \mathbb{F}_2^8$ *is a function that maps the bit-vector representation of an element in* $\mathbb{F}_{2^8}$ *to the representation of its inverse in* $\mathbb{F}_{2^8}$ *under a polynomial basis of* $\mathbb{F}_{2^8}$ *over* $\mathbb{F}_2$ *induced by an irreducible polynomial* $q(x) \in \mathbb{F}_2[x]$. We refer readers to Table 4 for the concrete values of these parameters for AES, Camellia and SM4.

Table 4: The parameters of the S-boxes of AES, Camellia, and SM4, where a hexadecimal number represents an irreducible polynomial in $\mathbb{F}_2[x]$ (e.g., $x^8 + x^4 + x^3 + x + 1$ is represented by `0x11B`).

| Cipher | $M_1$ | $C_1$ | $M_2$ | $C_2$ | $q(x)$ |
|---|---|---|---|---|---|
| AES | $\begin{pmatrix}1&0&0&0&0&0&0&0\\0&1&0&0&0&0&0&0\\0&0&1&0&0&0&0&0\\0&0&0&1&0&0&0&0\\0&0&0&0&1&0&0&0\\0&0&0&0&0&1&0&0\\0&0&0&0&0&0&1&0\\0&0&0&0&0&0&0&1\end{pmatrix}$ | $\begin{pmatrix}0\\0\\0\\0\\0\\0\\0\\0\end{pmatrix}$ | $\begin{pmatrix}1&1&1&1&1&0&0&0\\0&1&1&1&1&1&0&0\\0&0&1&1&1&1&1&0\\0&0&0&1&1&1&1&1\\1&0&0&0&1&1&1&1\\1&1&0&0&0&1&1&1\\1&1&1&0&0&0&1&1\\1&1&1&1&0&0&0&1\end{pmatrix}$ | $\begin{pmatrix}0\\1\\1\\0\\0\\0\\1\\1\end{pmatrix}$ | `0x11B` |
| Camellia* | $\begin{pmatrix}0&0&1&0&1&0&0&1\\1&0&1&0&1&0&0&0\\0&0&0&0&0&0&1&1\\0&1&1&0&0&0&0&1\\1&1&1&0&1&1&0&0\\0&1&0&1&0&1&0&1\\0&0&1&1&1&1&1&1\\0&0&0&0&0&1&1&0\end{pmatrix}$ | $\begin{pmatrix}1\\1\\1\\0\\1\\1\\0\\1\end{pmatrix}$ | $\begin{pmatrix}0&0&0&1&1&1&0&0\\0&1&1&1&0&0&0&0\\0&1&1&1&0&1&1&0\\1&0&0&0&1&1&0&1\\0&1&0&0&0&0&0&0\\0&0&1&0&0&1&0&1\\0&0&1&0&0&1&0&0\\0&0&0&1&0&0&0&0\end{pmatrix}$ | $\begin{pmatrix}0\\1\\1\\0\\1\\1\\1\\0\end{pmatrix}$ | `0x169` |
| SM4 | $\begin{pmatrix}1&1&0&1&0&0&1&1\\1&1&1&0&1&0&0&1\\1&1&1&1&0&1&0&0\\0&1&1&1&1&0&1&0\\0&0&1&1&1&1&0&1\\1&0&0&1&1&1&1&0\\0&1&0&0&1&1&1&1\\1&0&1&0&0&1&1&1\end{pmatrix}$ | $\begin{pmatrix}1\\1\\0\\1\\0\\0\\1\\1\end{pmatrix}$ | $\begin{pmatrix}1&1&0&1&0&0&1&1\\1&1&1&0&1&0&0&1\\1&1&1&1&0&1&0&0\\0&1&1&1&1&0&1&0\\0&0&1&1&1&1&0&1\\1&0&0&1&1&1&1&0\\0&1&0&0&1&1&1&1\\1&0&1&0&0&1&1&1\end{pmatrix}$ | $\begin{pmatrix}1\\1\\0\\1\\0\\0\\1\\1\end{pmatrix}$ | `0x1F5` |

* The description of the Camellia S-box in the original specification [2] is different from ours. Readers could check the substitution table to confirm the equivalence.

However, it is difficult to implement the function $I^q_{\mathcal{PB}}$ directly with small circuit footprint. Therefore, we first implement the function $I_{\mathcal{TB}} : \mathbb{F}_2^8 \to \mathbb{F}_2^8$ *which maps the representation of an element in* $\mathbb{F}_{2^8}$ *to the representation of its inverse element under the tower basis* $\mathcal{TB}$. According to the discussion of Section 2, we have

$$I^q_{\mathcal{PB}}(b) = M_t \cdot I_{\mathcal{TB}}(M_t^{-1} \cdot b).$$

Therefore, $S(b)$ can be implemented in practice as

$$S(b) = M_2 M_t \cdot I_{\mathcal{TB}}(M_t^{-1} M_1 \cdot b \oplus M_t^{-1} C_1) \oplus C_2,\ b \in \mathbb{F}_2^8. \tag{5}$$

Our goal is to identify a proper tower basis such that the overall circuit footprint of the implementation of $S(b)$ is minimized. Recalling the tower field architecture shown in Figure 1, the tower basis is completely determined by the three irreducible polynomials $r(y) = y^2 + \tau y + \nu \in \mathbb{F}_{2^4}[y]$, $s(z) = z^2 + Tz + N \in \mathbb{F}_{2^2}[z]$, $t(w) = w^2 + w + 1 \in \mathbb{F}_2[w]$, and their roots $Y$, $Z$ and $W$. Therefore, the $2^{15}$ possible choices of $\tau$, $\nu$, $T$, $N$, $Y$, $Z$, and $W$ form the overall design space[1], in which there are only 720 valid cases (we discard equivalent classes and non-irreducible polynomials).

Concretely, there are six possibilities for $(T, N)$ making $s(z)$ irreducible, and they are $\{(1, W), (1, W^2), (W, 1), (W, W), (W^2, 1), (W^2, W^2)\}$. For each possible choice of $(T, N)$, 120 cases of $(\tau, \nu)$ can be identified such that $r(y)$ is irreducible. We can choose either one of the two roots for $W$, $Z$ and $Y$ because the other roots are exactly $W^2$, $Z^4$ and $Y^{16}$. So altogether there are $6 \times 120 \times 1 \times 1 \times 1 = 720$ valid cases. We exhaust all these cases for AES, Camellia and SM4 and list the optimal parameter choices in terms of compactness in Table 5.

Table 5: Optimal choices of parameters for AES, Camellia and SM4 in terms of compactness. The parameters are given with their polynomial basis representations (e.g., $X^7 + X^5 + X^4 + X^3 + X^2$ is represented by `0xBC`).

| Cipher | $W$ | $T$ | $N$ | $Z$ | $\tau$ | $\nu$ | $Y$ |
|---|---|---|---|---|---|---|---|
| AES | `0xBC` | `0xBC` | `0x01` | `0xB0` | `0xBD` | `0x5C` | `0xF4` |
| Camellia | `0x7E` | `0x7E` | `0x01` | `0x15` | `0x01` | `0x06` | `0x02` |
| SM4 | `0x5C` | `0x5C` | `0x01` | `0x7A` | `0x77` | `0x27` | `0x66` |

According to Table 5, for the parameters of AES, we have the following relationship:

$$T = W, N = 1, \tau = T \cdot Z^4 + T \cdot Z, \nu = 0 \cdot Z^4 + T^2 \cdot Z. \tag{6}$$

Similarly, for Camellia, we have $T = W, N = 1, \tau = T^2 \cdot Z^4 + T^2 \cdot Z, \nu = T \cdot Z^4 + 0 \cdot Z$, and for SM4, we have $T = W, N = 1, \tau = T^2 \cdot Z^4 + 1 \cdot Z, \nu = T \cdot Z^4 + T^2 \cdot Z$. In what follows, we focus on the optimization of the AES S-box with the optimal parameter we identified as an example. For Camellia, SM4 and other parameters, the same procedure is performed.

### 4.1 Optimized Implementation of the $\mathbb{F}_{2^4}$ Multiplier, $\tau^2$ Multiplier, and the Square-$\nu$-scaler as a Whole

Before giving the optimized implementation, we unfold the circuits of the $\mathbb{F}_{2^4}$ multiplier, $\tau^2$ multiplier, and square-$\nu$-scaler one by one without any optimization. Based on these unfolded circuits, we reduce their areas by applying several logic minimization techniques with necessary tweaks.

[1] There are $2^2$ choices for $T$, $2^2$ choices for $N$, $2^4$ choices for $\tau$, $2^4$ choices for $\nu$, and 2 choices for each of $W$, $Z$ and $Y$ whenever $T, N, \tau, \nu$ are fixed.

$\mathbb{F}_{2^4}$ **Multiplier.** By plugging Figure 5 into Figure 3, we obtain the gate-level circuit of the $\mathbb{F}_{2^4}$ multiplier shown in Figure 6, which can also be derived by substituting Equation 3 and 4 into Equation 1.

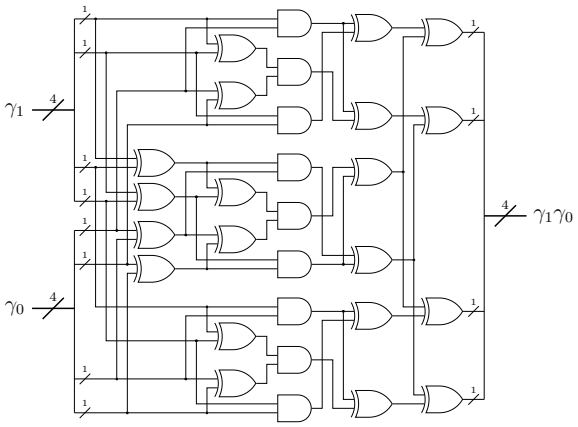


Fig. 6: $\mathbb{F}_{2^4}$ multiplier

$\tau^2$ **multiplier.** According to Table 5 and Equation 6, we have $\tau = TZ^4 + TZ = WZ^4 + WZ$ and $\tau^2 = Z^4 + Z$. Let $\alpha = (a_3W^2 + a_2W)Z^4 + (a_1W^2 + a_0W)Z$ be an arbitrary element in $\mathbb{F}_{2^4}$. We have

$$\alpha\tau^2 = [(a_3 \oplus a_2)W^2 + a_3W]Z^4 + [(a_1 \oplus a_0)W^2 + a_1W]Z, \tag{7}$$

leading to the gate-level circuit of the $\tau^2$ multiplier shown in Figure 7.

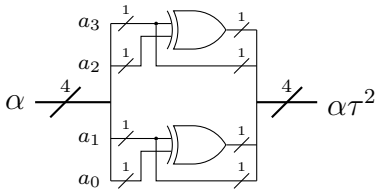


Fig. 7: $\tau^2$ multiplier

$\mathbb{F}_{2^4}$ **square-$\nu$-scaler.** According to Table 5 and Equation 6, $\nu = W^2Z$. Let $\alpha = (a_3W^2 + a_2W)Z^4 + (a_1W^2 + a_0W)Z$ be an arbitrary element in $\mathbb{F}_{2^4}$. Then $\alpha^2\nu$ can be computed as $\alpha^2\nu = [(a_3 + a_1)W^2 + (a_3 + a_2 + a_1 + a_0)W]Z^4 + [(a_1 + a_0)W^2 + a_0W]Z$, whose gate-level circuit is shown in Figure 8.

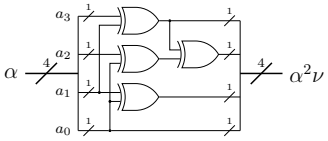


Fig. 8: $\mathbb{F}_{2^4}$ square-$\nu$-scaler

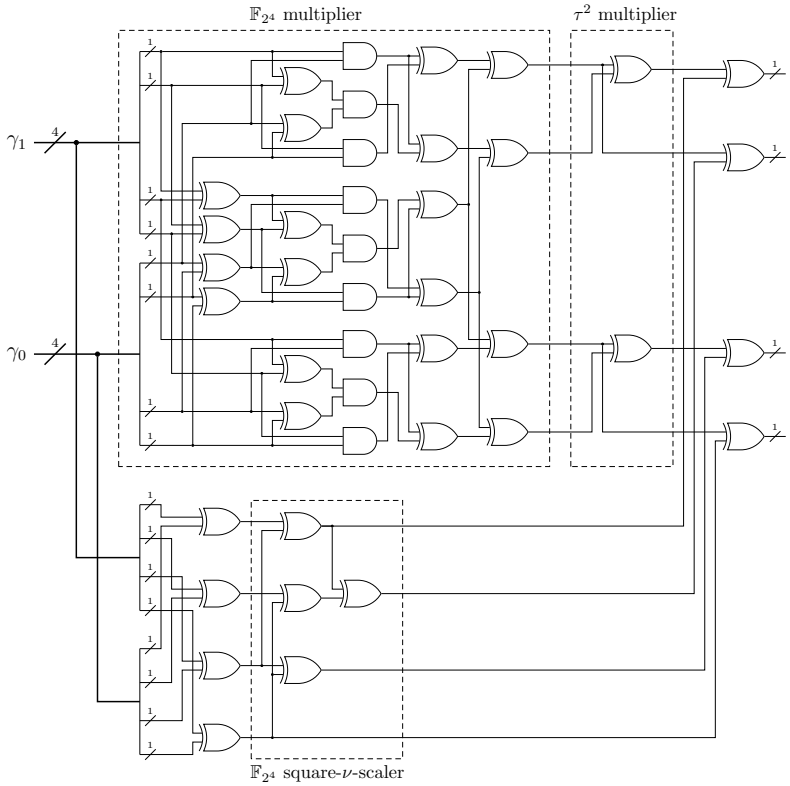


Fig. 9: A part of the $\mathbb{F}_{2^8}$ inverter (unoptimized)

Now, we can assemble the $\mathbb{F}_{2^4}$ multiplier, $\tau^2$ multiplier, and square-$\nu$-scaler to obtain a part of the $\mathbb{F}_{2^8}$ inverter according to Figure 2, which gives the circuit shown in Figure 9. According to Equation 4, this circuit can be partially optimized with some tweaks on the eight `XOR` gates appearing at the lower part of Figure 9, leading to the circuit shown in Figure 10. Subsequently, by applying the formula

$$a \cdot b \oplus a \oplus b = a \mid b \tag{8}$$

we can transform the circuit shown in Figure 10 into the circuit presented in Figure 11. According to Equation 8, at best, we can replace two `XOR` gates and

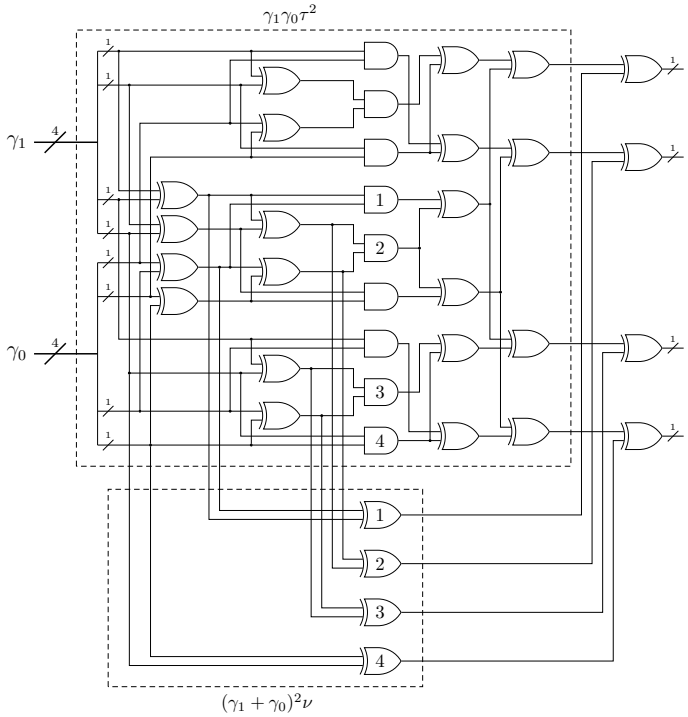


Fig. 10: A part of the $\mathbb{F}_{2^8}$ inverter (partially optimized)

one `AND` gate by a single `OR` gate. This happens for the gates marked by number 3. However, when some intermediate value of the computation $a \cdot b \oplus a \oplus b$ is required, we still need to keep some intermediate gates. For example, we can only replace two `XOR` gates with one `OR` gate and keep the `AND` gate intact. Similarly, for the gates marked with number 1 and number 2, we can only replace one `XOR` gates with one `OR` gate. Finally, by applying the formulas $a \cdot b \oplus c \cdot d = \overline{a \cdot b} \oplus \overline{c \cdot d}$ and $a \cdot b \oplus c \mid d = \overline{a \cdot b} \oplus \overline{c \mid d}$, the `AND` gates and `OR` gates can be substituted by `NAND` gates and `NOR` gates respectively.

### 4.2 Optimized Implementation of the $\mathbb{F}_{2^4}$ Inverter

Based on the selected parameters for AES ($T = W$ and $N = 1$) given in Table 5 and Equation 6, Equation 2 can be simplified as

$$\phi^{-1} = [\Phi_1\Phi_0 W^2 + (\Phi_1 + \Phi_0)^2]^{-1}\Phi_0 Z^4 + [\Phi_1\Phi_0 W^2 + (\Phi_1 + \Phi_0)^2]^{-1}\Phi_1 Z. \quad (9)$$

Deviating from previous implementations [10,11,9], we regard the $\mathbb{F}_{2^4}$ inverter as a $4 \times 4$ S-box whose permutation table is determined by Equation 9:

$$[\texttt{0x0}, \texttt{0x8}, \texttt{0x4}, \texttt{0xC}, \texttt{0x2}, \texttt{0xF}, \texttt{0x7}, \texttt{0x6}, \texttt{0x1}, \texttt{0xD}, \texttt{0xA}, \texttt{0xE}, \texttt{0x3}, \texttt{0x9}, \texttt{0xB}, \texttt{0x5}].$$

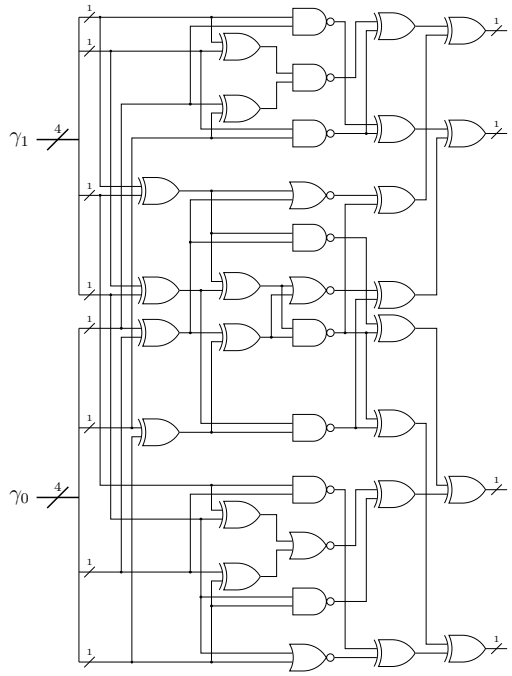


Fig. 11: A part of the $\mathbb{F}_{2^8}$ inverter (optimized)

To obtain optimized implementations of this S-box, we consider two recently proposed techniques. First, we employ Stoffelen's SAT-based technique [30] to produce a circuit of the $4 \times 4$ S-box: $(x_3, x_2, x_1, x_0) \mapsto (y_3, y_2, y_1, y_0)$ with minimized gate complexity, and the result is shown below:

$$
\begin{array}{lll}
t_1 = \overline{x_3 \cdot x_0} & t_2 = t_1 \mid x_2 & t_3 = \overline{x_2 \cdot x_0} \\
t_4 = x_1 \oplus t_3 & t_5 = \overline{x_2 \mid t_4} & t_6 = \overline{x_1 \cdot t_4} \\
t_7 = x_3 \mid t_4 & t_8 = t_7 \cdot t_2 & t_9 = t_5 \oplus t_7 \\
t_{10} = t_9 \odot x_3 \quad (y_0) & t_{11} = \overline{t_6 \cdot t_8} \quad (y_2) & t_{12} = \overline{t_8 \cdot x_1} \\
t_{13} = x_0 \odot t_{12} \quad (y_3) & t_{14} = \overline{t_1 \cdot x_2} & t_{15} = \overline{t_9 \cdot t_{14}} \quad (y_1),
\end{array}
$$

which contains 4 `XOR`/`XNOR` gates, 1 `AND` gate, 7 `NAND` gates, 2 `OR` gates and 1 `NOR` gate.[2]. This circuit (referred as `SAT`) can be further optimized manually. Since $a \cdot b = \overline{\bar{a} \mid \bar{b}}$, we can change the `AND` gate in $t_8$ to `NOR` gate, and negate its input signals without changing the overall functionality of the circuit. This new circuit (referred as `SAT`$^*$) contains 4 `XOR`/`XNOR` gates, 7 `NAND` gates and 4 `NOR` gates (modified signals

[2]It costs about 38 minutes to obtain this 15-gate circuit on a PC. But we cannot confirm whether there is a 14-gate circuit, since we terminate the SAT solver after about two weeks' computation

are colored in red):

$$
\begin{array}{lll}
t_1 = \overline{x_3 \cdot x_0} & t_2 = \overline{t_1 \mid x_2} & t_3 = \overline{x_2 \cdot x_0} \\
t_4 = x_1 \oplus t_3 & t_5 = \overline{x_2 \mid t_4} & t_6 = \overline{x_1 \cdot t_4} \\
t_7 = \overline{x_3 \mid t_4} & t_8 = \overline{t_7 \mid t_2} & t_9 = t_5 \odot t_7 \\
t_{10} = t_9 \odot x_3 \quad (y_0) & t_{11} = \overline{t_6 \cdot t_8} \quad (y_2) & t_{12} = \overline{t_8 \cdot x_1} \\
t_{13} = x_0 \odot t_{12} \quad (y_3) & t_{14} = \overline{t_1 \cdot x_2} & t_{15} = \overline{t_9 \cdot t_{14}} \quad (y_1).
\end{array}
$$

We also apply the `LIGHTER` [18] tool to the $4 \times 4$ S-box (the $\mathbb{F}_{2^4}$ inverter) for four different technology libraries, which leads to the same circuit (referred as `LIGHTER`) containing 7 `XOR/XNOR` gates, 4 `NAND` gates, 1 `NAND3` gate, 1 `NOR` gate and 1 `NOT` gate shown in the following:

$$
\begin{array}{lll}
t_1 = x_2 \oplus x_3 & t_2 = \overline{t_1 \cdot x_0 \cdot x_3} & t_3 = x_1 \odot t_2 \\
t_4 = \overline{t_3 \cdot t_1} & t_5 = x_0 \oplus t_4 & t_6 = \overline{x_3 \cdot t_5} \\
t_7 = t_1 \oplus t_6 & t_8 = \overline{t_5 \mid t_7} & t_9 = x_3 \oplus t_8 \quad (y_0) \\
t_{10} = \overline{t_9 \cdot t_3} & t_{11} = t_5 \oplus t_{10} \quad (y_3) & t_{12} = \overline{t_7} \quad (y_1) \\
t_{13} = \overline{t_{11} \cdot t_{12}} & t_{14} = t_3 \odot t_{13} \quad (y_2). &
\end{array}
$$

A comparison of the above three circuits together with their synthesizing results is given in Table 6 and Table 7, from which we can see that `SAT`$^*$ is always the best, whose circuit is depicted in Figure 12.

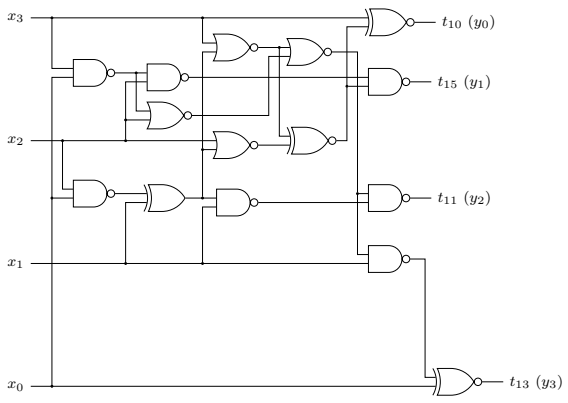


Fig. 12: The optimized circuit for the $\mathbb{F}_{2^4}$ inverter (`SAT*`)

### 4.3 Optimized Implementation of the Two $\mathbb{F}_{2^4}$ Multipliers with 4-bit Common Input

Observing Figure 2, the $\mathbb{F}_{2^8}$ inverter contains three $\mathbb{F}_{2^4}$ multipliers, from which we can identify three pairs of $\mathbb{F}_{2^4}$ multipliers such that each pair shares a 4-bit

Table 6: Gate counts for different implementations of $\mathbb{F}_{2^4}$ inverter, where the circuit named `Canright` is the implementation of Equation 9 using the method in [10,11], and the circuit named `Boyar` uses the method in [9].

| Circuit | Gates used | | | | | | |
|---|---|---|---|---|---|---|---|
| | XOR/XNOR | AND | NAND | NAND3 | OR | NOR | NOT |
| Canright | 9 | | 8 | | | 2 | |
| Boyar | 9 | 6 | | | | | |
| Boyar* | 9 | | 6 | | | | |
| SAT | 4 | 1 | 7 | | 2 | 1 | |
| SAT* | 4 | | 7 | | | 4 | |
| LIGHTER | 7 | | 4 | 1 | | 1 | 1 |

Table 7: Synthesized results for different implementations of the $\mathbb{F}_{2^4}$ inverter

| Circuit | Synthesis results | | | |
|---|---|---|---|---|
| | SMIC 130nm | SMIC 65nm | STM 65nm | Nangate 45nm |
| Canright | 30.97 | 30.25 | 28.00 | 28.00 |
| Boyar | 28.95 | 29.25 | 25.50 | 25.98 |
| Boyar* | 26.97 | 26.25 | 24.00 | 24.00 |
| SAT | 21.31 | 21.50 | 20.25 | 19.99 |
| SAT* | **20.32** | **20.00** | **19.00** | **19.00** |
| LIGHTER | 23.30 | 22.75 | 21.00 | 20.99 |

input signal: the leftmost $\mathbb{F}_{2^4}$ multiplier and the rightmost upper $\mathbb{F}_{2^4}$ multiplier, the leftmost $\mathbb{F}_{2^4}$ multiplier and the rightmost lower $\mathbb{F}_{2^4}$ multiplier, and the two rightmost $\mathbb{F}_{2^4}$ multipliers. It is shown in [11,10] that whenever two $\mathbb{F}_{2^4}$ multipliers share a common 4-bit input signal, some `XOR` gates can be saved via signal reuse.

As an example, we unfold the two rightmost $\mathbb{F}_{2^4}$ multipliers in Figure 2 according to Figure 6, and the schematic is shown in Figure 13. By observing Figure 13 carefully, we can spot some outputs of `XOR` gates which are computed twice in the circuit [11,10] (labeled with same numbers in the figure). Therefore, for each pair of $\mathbb{F}_{2^4}$ multipliers sharing a 4-bit input signal, we can remove 5 `XOR` gates by signal reuse. Therefore, 3 pairs of $\mathbb{F}_{2^4}$ multipliers with shared input signals in total save $5 \times 3 = 15$ `XOR` gates.

### 4.4 Optimized Implementation of the Input and Output Affine Parts

According to Equation 5, before going into the $\mathbb{F}_{2^8}$ inverter $I_{\mathcal{TB}}(\cdot)$, the 8-bit input signal of the AES S-box first goes through an affine transformation

$$b \mapsto g = M_t^{-1} M_1 \cdot b \oplus M_t^{-1} C_1,$$

which then spawns 18 1-bit signals (see Figure 11) subsequently fed into some non-linear gates (`NAND`, `NOR`). The transformation from the 8 1-bit input signals to

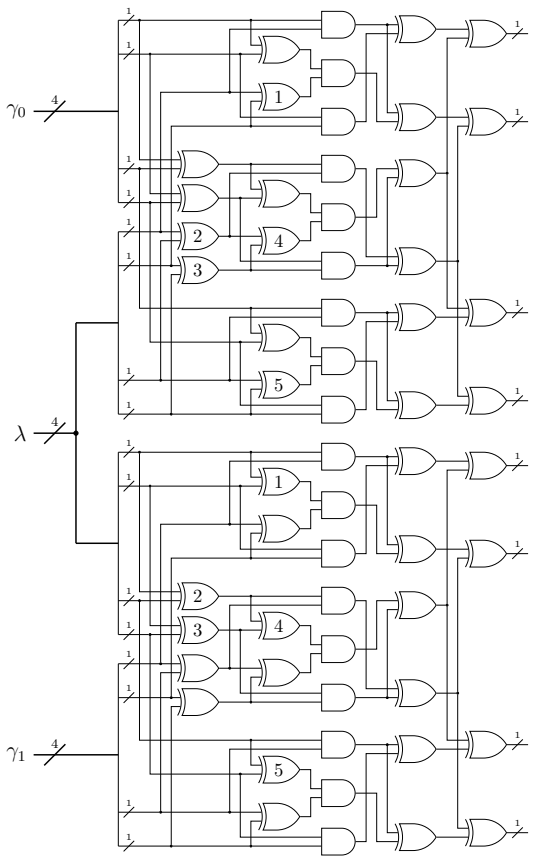


Fig. 13: The two rightmost $\mathbb{F}_{2^4}$ multipliers with a shared 4-bit input

the 18 1-bit signals is affine, and can be represented as an $18 \times 8$ matrix

$$U = \begin{pmatrix} 0&1&0&0&0&1&1&0&1&1&0&1&1&1&1&0&0&1 \\ 0&0&1&1&1&1&1&1&0&0&1&0&1&0&0&0&0&0 \\ 0&0&0&1&1&1&1&1&0&0&0&0&1&0&1&0&1&0 \\ 0&1&0&0&1&0&0&0&1&1&0&1&1&0&1&1&0&0 \\ 0&1&1&0&0&0&0&0&1&0&1&0&1&0&0&0&1&0 \\ 0&0&1&0&0&1&0&0&0&1&1&0&0&1&1&1&1&0 \\ 0&1&1&0&0&1&0&1&1&1&1&0&1&0&0&0&1&1 \\ 1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0 \end{pmatrix}^{\mathsf{T}},$$

with the constant

$$M_t^{-1}C_1 = \begin{pmatrix} 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \end{pmatrix}^{\mathsf{T}}.$$

By applying the SAT-based method for solving SLP (Shortest Linear Straight-Line Program) [14], we obtain the optimal implementation of $U$, which costs 19

`XOR` gates [3]:

$$
\begin{array}{lll}
y_{17} = x_0 \quad (y_{17}) & t_1 = x_7 \oplus x_4 \quad (y_6) & t_2 = x_3 \oplus x_1 \\
t_3 = x_2 \oplus t_2 & t_4 = x_6 \oplus t_3 \quad (y_7) & t_5 = x_0 \oplus t_4 \quad (y_{15}) \\
t_6 = x_5 \oplus t_3 \quad (y_1) & t_7 = t_5 \oplus t_6 \quad (y_{14}) & t_8 = x_4 \oplus t_7 \quad (y_{13}) \\
t_9 = t_1 \oplus t_2 \quad (y_9) & t_{10} = t_1 \oplus t_8 \quad (y_{11}) & t_{11} = x_0 \oplus t_9 \quad (y_{16}) \\
t_{12} = x_4 \oplus x_2 \quad (y_2) & t_{13} = x_1 \oplus t_7 \quad (y_{10}) & t_{14} = x_7 \oplus x_2 \quad (y_4) \\
t_{15} = t_{13} \oplus t_{14} \quad (y_{12}) & t_{16} = x_7 \oplus x_1 \quad (y_0) & t_{17} = t_{12} \oplus t_{16} \quad (y_8) \\
t_{18} = t_6 \oplus t_9 \quad (y_3) & t_{19} = t_7 \oplus t_{11} \quad (y_5) &
\end{array}
$$

where $x_i$'s are the input signals, $t_i$'s are intermediate signals, and $y_i$'s are output signals.

Similarly, according to Equation 5, at the output end of the AES S-box, the 8-bit output of the two rightmost $\mathbb{F}_{2^4}$ multipliers (see Figure 2 and Figure 14) is transformed by the affine mapping $M_2 M_t(\cdot) \oplus C_2$ to recover the polynomial basis representation. Observing Figure 14, the 8 input bits of the affine mapping (also the output bits of the two $\mathbb{F}_{2^4}$ multipliers) are originated from the 18 output bits of the `NAND` gates. Moreover, only `XOR` gates are involved to generate the 8 input bits of the affine mapping from these 18 bits. Therefore, the mapping from the 18 output bits of the `NAND` gates to the 8 output bits of the S-box is affine, which can be described by the following matrix

$$
B = \begin{pmatrix}
0\,1\,1\,0\,1\,1\,0\,0\,0\,0\,0\,0\,0\,1\,1\,0\,1\,1 \\
0\,1\,1\,0\,1\,1\,0\,0\,0\,0\,1\,1\,0\,1\,1\,0\,0\,0 \\
0\,0\,0\,1\,1\,0\,1\,1\,0\,1\,1\,0\,1\,1\,0\,0\,0\,0 \\
0\,1\,1\,0\,1\,1\,0\,0\,0\,0\,1\,1\,0\,0\,0\,0\,1\,1 \\
0\,1\,1\,0\,1\,1\,0\,0\,0\,1\,0\,1\,0\,0\,0\,1\,0\,1 \\
1\,0\,1\,0\,1\,1\,1\,1\,0\,1\,1\,0\,1\,0\,1\,0\,1\,1 \\
0\,0\,0\,0\,1\,1\,0\,1\,1\,0\,0\,0\,1\,0\,1\,1\,0\,1 \\
0\,0\,0\,0\,1\,1\,0\,1\,1\,1\,1\,1\,0\,0\,0\,0\,1\,0
\end{pmatrix}.
$$

Observing Figure 14, there are two layers of `XOR` gates following the 18 `NAND` gates. According these two layers of `XOR` gates, we decompose the whole output affine transformation (from the 18 output bits of the 18 `NAND` gates to the 8 output bits of the AES S-box) into two parts. The first part maps the 18 output bits of the 18 `NAND` gates to the 12 output bits of the first layer of 12 `XOR` gates, which can be implemented with in total 12 `XOR` gates as shown in Figure 14. The second part maps the 12 output bits of the 12 `XOR` gates to the 8 output bits of the S-box, and its matrix representation $B'$ is given in the following:

$$
B' = \begin{pmatrix}
0\,1\,0\,1\,0\,0\,0\,0\,0\,1\,0\,1 \\
0\,1\,0\,1\,0\,0\,0\,1\,0\,1\,0\,0 \\
0\,0\,1\,0\,1\,0\,1\,0\,1\,0\,0\,0 \\
0\,1\,0\,1\,0\,0\,0\,1\,0\,0\,0\,1 \\
0\,1\,0\,1\,0\,0\,1\,1\,0\,0\,1\,1 \\
1\,1\,0\,1\,1\,0\,1\,0\,1\,1\,0\,1 \\
0\,0\,0\,1\,0\,1\,0\,0\,1\,1\,1\,1 \\
0\,0\,0\,1\,0\,1\,1\,0\,0\,0\,1\,0
\end{pmatrix}, \tag{10}
$$

[3] It costs about 25 days on a PC to produce this result.

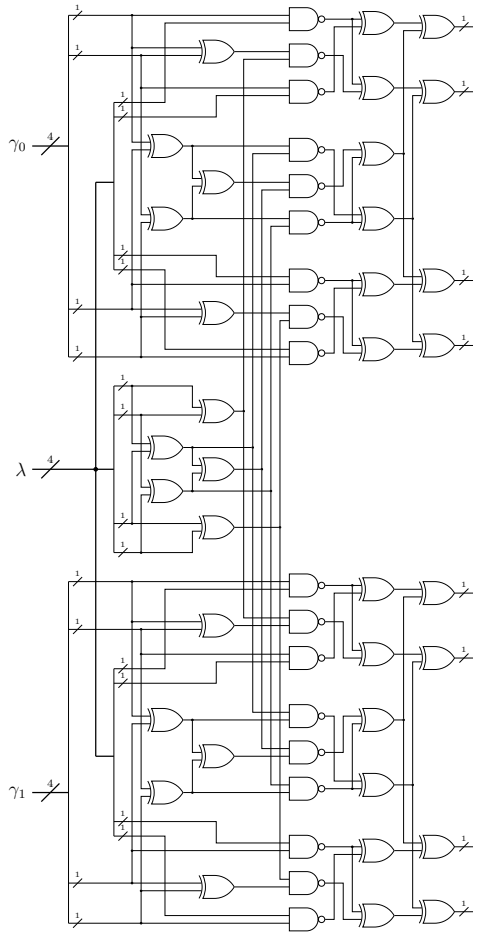


Fig. 14: The circuit for bottom part

Again, by applying the SAT based method for SLP [14] and taking $C_2$ into account, the affine transformation involving $B'$ and constant addition at the output end can be realized as follows, requiring 17 `XOR`/`XNOR` gates [4]:

$$
\begin{array}{lll}
t_1 = x_2 \oplus x_0 & t_2 = x_{10} \oplus t_1 & t_3 = x_8 \oplus t_2 \quad (y_7) \\
t_4 = x_5 \oplus x_1 & t_5 = x_5 \oplus x_3 & t_6 = x_7 \oplus t_5 \\
t_7 = x_8 \oplus x_6 & t_8 = x_2 \oplus t_3 & t_9 = x_4 \oplus t_8 \quad (y_4) \\
t_{10} = t_1 \oplus t_5 & t_{11} = x_9 \odot t_6 \quad (y_5) & t_{12} = t_4 \odot t_7 \quad (y_0) \\
t_{13} = t_3 \oplus t_6 & t_{14} = x_{11} \oplus t_{13} \quad (y_2) & t_{15} = t_1 \odot t_9 \quad (y_6) \\
t_{16} = t_{10} \oplus t_{12} \quad (y_1) & t_{17} = t_4 \oplus t_9 \quad (y_3) &
\end{array}
$$

[4] It costs about 30 days on a PC to produce this circuit.

where $x_i$'s are the input signals, $t_i$'s are intermediate signals, and $y_i$'s are output signals.

### 4.5 Overall Implementation Results and Comparison

We synthesis the optimized implementations of the S-boxes (AES, Camellia, SM4) using Synopsys Design Compiler 2014 (DC 2014) with four technology libraries, and the synthesized results [5] together with their technology-independent gate counts are listed in Table 2.

To make the full use of the libraries to save the circuit area, Reyhani-Masoleh et. al.'s implementations [26] exploit certain compound gates in specific libraries (e.g., `XOR3`, `NAND3`, `OAI21`, `AOI21`, `OAI32`), which are not universally available in all technology libraries. For example, the optimal implementation offered by [26] employs `XOR3` and `OAI32` gates, reaching 182.25 `GE` under the `STM` 65`nm` CMOS technology.

According to the results shown in Table 2, our implementation requires only 179 `GE`, which beats the record set by [26] even without using any compound gates. Moreover, when the area of one `XOR3` gate is smaller than the area of two `XOR` gates in underlying technology library, the compound gates `XOR3` can be applied in our design to take the place of some standard `XOR` gates. With this improvements, the area of our implementation of the AES S-box can be further reduced to 176.75 `GE`. For the S-boxes of Camellia and SM4, it can be seen from Table 2 that the improvements are even more obvious.

## 5 Conclusion

By applying state-of-the-art combinatorial logic minimization techniques to an exhaustive list of tower field representations of the AES, Camellia, and SM4 S-boxes with normal bases, we identify so far the most compact implementations of these S-boxes. The results obtained in this work can be used in compact and threshold implementations of AES, Camellia, and SM4. As a potential further work, it is interesting to see how to apply similar techniques to obtain compact implementations of combined S-box/inverse S-box designs for AES, Camellia, and SM4. Moreover, this work only focus on minimizing the circuit area, it is of equal importance to investigate how to reduce the depth of the circuit as in [26, 20].

**Acknowledgment.** The work is supported by the National Key R&D Program of China (Grant No. 2018YFB0804402), the Chinese Major Program of National Cryptography Development Foundation (Grant No. MMJJ20180102), the National Natural Science Foundation of China (61732021, 61802400, 61772519, 61802399), and the Youth Innovation Promotion Association of Chinese Academy of Sciences.

[5] We do not have access to the `STM` 65`nm` technology library. However, the authors of [26] provide sufficient area information for the gates involved in this particular library, based on which we can extrapolate the results without any difficulty.

## References


1. Abbasi, I., Afzal, M.: A compact S-Box design for SMS4 block cipher. IACR Cryptology ePrint Archive **2011**, 522 (2011)
2. Aoki, K., Ichikawa, T., Kanda, M., Matsui, M., Moriai, S., Nakajima, J., Tokita, T.: Camellia: A 128-bit block cipher suitable for multiple platforms - design and analysis. Lecture Notes in Computer Science pp. 39–56 (2000)
3. Bai, X., Xu, Y., Guo, L.: Securing SMS4 cipher against differential power analysis and its VLSI implementation. In: IEEE Singapore International Conference on Communication Systems, pp. 167–172 (2009)
4. Banik, S., Bogdanov, A., Minematsu, K.: Low-area hardware implementations of CLOC, SILC and AES-OTR. In: 2016 IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2016, McLean, VA, USA, May 3-5, 2016, pp. 71–74 (2016)
5. Banik, S., Bogdanov, A., Regazzoni, F.: Atomic-AES: A compact implementation of the AES encryption/decryption core. In: Progress in Cryptology - INDOCRYPT 2016 - 17th International Conference on Cryptology in India, Kolkata, India, December 11-14, 2016, Proceedings, pp. 173–190 (2016)
6. Beierle, C., Kranz, T., Leander, G.: Lightweight multiplication in GF($2^n$) with applications to MDS matrices. In: Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I, pp. 625–653 (2016)
7. Bilgin, B., Gierlichs, B., Nikova, S., Nikov, V., Rijmen, V.: A more efficient AES threshold implementation. In: Progress in Cryptology - AFRICACRYPT 2014 - 7th International Conference on Cryptology in Africa, Marrakesh, Morocco, May 28-30, 2014. Proceedings, pp. 267–284 (2014)
8. Bilgin, B., Gierlichs, B., Nikova, S., Nikov, V., Rijmen, V.: Trade-offs for threshold implementations illustrated on AES. IEEE Trans. on CAD of Integrated Circuits and Systems **34**(7), 1188–1200 (2015)
9. Boyar, J., Matthews, P., Peralta, R.: Logic minimization techniques with applications to cryptology. J. Cryptology **26**(2), 280–312 (2013)
10. Canright, D.: A very compact Rijndael S-box. Tech. rep., Naval Postgraduate School (2005). NPS-MA-05-001
11. Canright, D.: A very compact S-Box for AES. In: Cryptographic Hardware and Embedded Systems - CHES 2005, 7th International Workshop, Edinburgh, UK, August 29 - September 1, 2005, Proceedings, pp. 441–455 (2005)
12. Cnudde, T.D., Reparaz, O., Bilgin, B., Nikova, S., Nikov, V., Rijmen, V.: Masking AES with $d + 1$ shares in hardware. In: Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings, pp. 194–212 (2016)
13. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Information Security and Cryptography. Springer (2002)
14. Fuhs, C., Schneider-Kamp, P.: Synthesizing shortest linear straight-line programs over GF(2) using SAT. In: Theory and Applications of Satisfiability Testing - SAT 2010, 13th International Conference, SAT 2010, Edinburgh, UK, July 11-14, 2010. Proceedings, pp. 71–84 (2010)
15. Guajardo, J., Paar, C.: Efficient algorithms for elliptic curve cryptosystems. In: Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings, pp. 342–356 (1997)
16. Itoh, T., Tsujii, S.: A fast algorithm for computing multiplicative inverses in GF($2^m$)) using normal bases. Inf. Comput. **78**(3), 171–177 (1988)
17. Jean, J., Moradi, A., Peyrin, T., Sasdrich, P.: Bit-sliding: A generic technique for bit-serial implementations of SPN-based primitives - applications to AES, PRESENT and SKINNY. In: Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings, pp. 687–707 (2017)
18. Jean, J., Peyrin, T., Sim, S.M., Tourteaux, J.: Optimizing implementations of lightweight building blocks. IACR Trans. Symmetric Cryptol. **2017**(4), 130–168 (2017)
19. Kranz, T., Leander, G., Stoffelen, K., Wiemer, F.: Shorter linear straight-line programs for MDS matrices. IACR Trans. Symmetric Cryptol. **2017**(4), 188–211 (2017)
20. Li, S., Sun, S., Li, C., Wei, Z., Hu, L.: Constructing low-latency involutory MDS matrices with lightweight circuits. IACR Trans. Symmetric Cryptol. **2019**(1), 84–117 (2019)

21. Martínez-Herrera, A.F., Mex-Perera, J.C., Nolazco-Flores, J.A.: Some representations of the S-Box of Camellia in GF($((2^2)^2)^2$). In: Cryptology and Network Security, 11th International Conference, CANS 2012, Darmstadt, Germany, December 12-14, 2012. Proceedings, pp. 296–309 (2012)
22. Martínez-Herrera, A.F., Mex-Perera, J.C., Nolazco-Flores, J.A.: Merging the Camellia, SMS4 and AES S-Boxes in a single S-Box with composite bases. In: Information Security, 16th International Conference, ISC 2013, Dallas, Texas, USA, November 13-15, 2013, Proceedings, pp. 209–217 (2013)
23. Mentens, N., Batina, L., Preneel, B., Verbauwhede, I.: A systematic evaluation of compact hardware implementations for the Rijndael S-Box. In: Topics in Cryptology - CT-RSA 2005, The Cryptographers' Track at the RSA Conference 2005, San Francisco, CA, USA, February 14-18, 2005, Proceedings, pp. 323–333 (2005)
24. Moradi, A., Poschmann, A., Ling, S., Paar, C., Wang, H.: Pushing the limits: A very compact and a threshold implementation of AES. In: Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings, pp. 69–88 (2011)
25. Paar, C., Soria-Rodriguez, P.: Fast arithmetic architectures for public-key algorithms over Galois Fields GF($(2^n)^m$). In: Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding, pp. 363–378 (1997)
26. Reyhani-Masoleh, A., Taha, M.M.I., Ashmawy, D.: Smashing the implementation records of AES s-box. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2018**(2), 298–336 (2018)
27. Rudra, A., Dubey, P.K., Jutla, C.S., Kumar, V., Rao, J.R., Rohatgi, P.: Efficient Rijndael encryption implementation with composite field arithmetic. In: Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings, Generators, pp. 171–184 (2001)
28. Satoh, A., Morioka, S.: Unified hardware architecture for 128-bit block ciphers AES and Camellia. In: Cryptographic Hardware and Embedded Systems - CHES 2003, 5th International Workshop, Cologne, Germany, September 8-10, 2003, Proceedings, pp. 304–318 (2003)
29. Satoh, A., Morioka, S., Takano, K., Munetoh, S.: A compact Rijndael hardware architecture with S-Box optimization. In: Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings, pp. 239–254 (2001)
30. Stoffelen, K.: Optimizing S-Box implementations for several criteria using SAT solvers. In: Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers, pp. 140–160 (2016)
31. Team, C.M.: http://www.cs.yale.edu/homes/peralta/CircuitStuff/CMT.html
32. Wolkerstorfer, J., Oswald, E., Lamberger, M.: An ASIC implementation of the AES S-Boxes. In: Topics in Cryptology - CT-RSA 2002, The Cryptographer's Track at the RSA Conference, 2002, San Jose, CA, USA, February 18-22, 2002, Proceedings, pp. 67–78 (2002)

## A Verilog Code for the AES S-box

```
/* The AES S-box */
module AES (b, Sb);
        input [7:0] b;
        output [7:0] Sb;
        wire [7:0] g;
        wire [9:0] m;
        wire [3:0] p, l;
        wire [17:0] e;

        Input M1(b, g, m);
        Top M2(g, m, p);
        Middle M3(p, l);
        Bottom M4(g, m, l, e);
```

```
        Output M5(e, Sb);
endmodule

/* Gates implemented as modules to prevent
 unintentional optimization of the DC */
module XOR (t, a, b);
        output t;
        input a, b;
        xor(t, a, b);
endmodule

module XNOR (t, a, b);
        output t;
        input a, b;
        xnor(t, a, b);
endmodule

module NAND (t, a, b);
        output t;
        input a, b;
        nand(t, a, b);
endmodule

module NOR (t, a, b);
        output t;
        input a, b;
        nor(t, a, b);
endmodule

/* input matrix */
module Input (b, g, m);
        input [7:0] b;
        output [7:0] g;
        output [9:0] m;
        wire t1, t2, t3, t4, t5, t6, t7, t8, t9, t10,
             t11, t12, t13, t14, t15, t16, t17, t18, t19;

        XOR m1(t1, b[7], b[4]);
        XOR m2(t2, b[3], b[1]);
        XOR m3(t3, b[2], t2);
        XOR m4(t4, b[6], t3);
        XOR m5(t5, b[0], t4);
        XOR m6(t6, b[5], t3);
        XOR m7(t7, t5, t6);
        XOR m8(t8, b[4], t7);
        XOR m9(t9, t1, t2);
        XOR m10(t10, t1, t8);
        XOR m11(t11, b[0], t9);
        XOR m12(t12, b[4], b[2]);
        XOR m13(t13, b[1], t7);
        XOR m14(t14, b[7], b[2]);
        XOR m15(t15, t13, t14);
        XOR m16(t16, b[7], b[1]);
        XOR m17(t17, t12, t16);
        XOR m18(t18, t6, t9);
        XOR m19(t19, t7, t11);

        assign g = {b[0], t11, t5, t7, t8, t15, t10, t13};
        assign m = {t9, t17, t4, t1, t19, t14, t18, t12, t6, t16};
endmodule
```

```
/* top part: GF(2^4) multiplier and GF(2^4) square-scaler */
module Top (g, m, p);
        input [7:0] g;
        input [9:0] m;
        output [3:0] p;
        wire t1, t2, t3, t4, t5, t6, t7, t8, t9, t10, t11, t12, t13,
             t14, t15, t16, t17, t18, t19, t20, t21, t22, t23, t24;

        NAND m1(t1, g[6], g[2]);
        NAND m2(t2, m[9], m[8]);
        NAND m3(t3, g[7], g[3]);
        XOR m4(t13, t1, t2);
        XOR m5(t14, t3, t1);

        NAND m6(t4, m[7], m[6]);
        NOR m7(t5, m[7], m[6]);
        NAND m8(t6, m[3], m[2]);
        NOR m9(t7, m[3], m[2]);
        NAND m10(t8, m[5], m[4]);
        XOR m11(t15, t5, t6);
        XOR m12(t16, t8, t7);
        XOR m13(t17, t4, t6);
        XOR m14(t18, t8, t6);

        NAND m15(t9, g[4], g[0]);
        NOR m16(t10, m[1], m[0]);
        NAND m17(t11, g[5], g[1]);
        NOR m18(t12, g[4], g[0]);
        XOR m19(t19, t9, t10);
        XOR m20(t20, t11, t12);

        XOR m21(t21, t13, t15);
        XOR m22(t22, t14, t16);
        XOR m23(t23, t19, t17);
        XOR m24(t24, t20, t18);

        assign p[3] = t21;
        assign p[2] = t22;
        assign p[1] = t23;
        assign p[0] = t24;
endmodule

/* middle part: GF(2^4) inverse */
module Middle (p, l);
        input [3:0] p;
        output [3:0] l;
        wire t1, t2, t3, t4, t5, t6, t7, t8, t9, t10,
             t11, t12, t13, t14, t15;

        NAND m1(t1, p[3], p[0]);
        NOR m2(t2, t1, p[2]);
        NAND m3(t3, p[2], p[0]);
        XOR m4(t4, p[1], t3);
        NOR m5(t5, p[2], t4);
        NAND m6(t6, p[1], t4);
        NOR m7(t7, p[3], t4);
        NOR m8(t8, t7, t2);
        XNOR m9(t9, t5, t7);
        XNOR m10(t10, t9, p[3]);
```

```
        NAND m11(t11, t6, t8);
        NAND m12(t12, t8, p[1]);
        XNOR m13(t13, p[0], t12);
        NAND m14(t14, t1, p[2]);
        NAND m15(t15, t9, t14);

        assign l[3] = t13;
        assign l[2] = t11;
        assign l[1] = t15;
        assign l[0] = t10;
endmodule

/* bottom part: GF(2^4) multipliers */
module Bottom (g, m, l, e);
        input [7:0] g;
        input [9:0] m;
        input [3:0] l;
        output [17:0] e;
        wire k4, k3, k2, k1, k0;

        XOR m1(k4, l[3], l[2]);
        XOR m2(k3, l[3], l[1]);
        XOR m3(k2, l[2], l[0]);
        XOR m4(k1, k3, k2);
        XOR m5(k0, l[1], l[0]);

        NAND m6(e[17], g[2], l[2]);
        NAND m7(e[16], g[3], l[3]);
        NAND m8(e[15], m[8], k4);

        NAND m9(e[14], m[2], k1);
        NAND m10(e[13], m[4], k2);
        NAND m11(e[12], m[6], k3);

        NAND m12(e[11], g[0], l[0]);
        NAND m13(e[10], g[1], l[1]);
        NAND m14(e[9], m[0], k0);

        NAND m15(e[8], g[6], l[2]);
        NAND m16(e[7], g[7], l[3]);
        NAND m17(e[6], m[9], k4);

        NAND m18(e[5], m[3], k1);
        NAND m19(e[4], m[5], k2);
        NAND m20(e[3], m[7], k3);

        NAND m21(e[2], g[4], l[0]);
        NAND m22(e[1], g[5], l[1]);
        NAND m23(e[0], m[1], k0);
endmodule

/* output matrix */
module Output (e, Sb);
        input [17:0] e;
        output [7:0] Sb;
        wire E11, E10, E9, E8, E7, E6, E5, E4, E3, E2, E1, E0;
        wire t1, t2, t3, t4, t5, t6, t7, t8, t9, t10,
             t11, t12, t13, t14, t15, t16, t17;

        XOR m1(E11, e[17], e[16]);
```

```
        XOR m2(E10, e[15], e[16]);
        XOR m3(E9, e[14], e[13]);
        XOR m4(E8, e[12], e[13]);
        XOR m5(E7, e[11], e[10]);
        XOR m6(E6, e[9], e[10]);
        XOR m7(E5, e[8], e[7]);
        XOR m8(E4, e[6], e[7]);
        XOR m9(E3, e[5], e[4]);
        XOR m10(E2, e[3], e[4]);
        XOR m11(E1, e[2], e[1]);
        XOR m12(E0, e[0], e[1]);

        XOR m13(t1, E2, E0);
        XOR m14(t2, E10, t1);
        XOR m15(t3, E8, t2);
        XOR m16(t4, E5, E1);
        XOR m17(t5, E5, E3);
        XOR m18(t6, E7, t5);
        XOR m19(t7, E8, E6);
        XOR m20(t8, E2, t3);
        XOR m21(t9, E4, t8);
        XOR m22(t10, t1, t5);
        XNOR m23(t11, E9, t6);
        XNOR m24(t12, t4, t7);
        XOR m25(t13, t3, t6);
        XOR m26(t14, E11, t13);
        XNOR m27(t15, t1, t9);
        XOR m28(t16, t10, t12);
        XOR m29(t17, t4, t9);

        assign Sb = {t3, t15, t11, t9, t17, t14, t16, t12};
endmodule
```

## B Verilog Code for the Camellia S-box

```
/* The Camellia S-box */
module Camellia (b, Sb);
        input [7:0] b;
        output [7:0] Sb;
        wire [7:0] g;
        wire [9:0] m;
        wire [3:0] p, l;
        wire [17:0] e;

        Input M1(b, g, m);
        Top M2(g, m, p);
        Middle M3(p, l);
        Bottom M4(g, m, l, e);
        Output M5(e, Sb);
endmodule

/* Gates implemented as modules to prevent
 unintentional optimization of the DC */
module XOR (t, a, b);
        output t;
        input a, b;
        xor(t, a, b);
endmodule

module XNOR (t, a, b);
        output t;
        input a, b;
        xnor(t, a, b);
endmodule

module NAND (t, a, b);
        output t;
        input a, b;
        nand(t, a, b);
endmodule

module NOR (t, a, b);
        output t;
        input a, b;
        nor(t, a, b);
endmodule

/* input matrix */
module Input (b, g, m);
        input [7:0] b;
        output [7:0] g;
        output [9:0] m;
        wire t1, t2, t3, t4, t5, t6, t7, t8, t9, t10,
             t11, t12, t13, t14, t15, t16, t17, t18, t19;

        XNOR m1(t1, b[6], b[3]);
        XNOR m2(t2, b[4], b[2]);
        XOR m3(t3, b[4], b[1]);
        XOR m4(t4, t1, t3);
        XOR m5(t5, t2, t4);
        XOR m6(t6, b[5], t2);
        XNOR m7(t7, b[0], t6);
        XOR m8(t8, b[7], b[0]);
```

```
        XOR m9(t9, t1, t8);
        XOR m10(t10, t5, t9);
        XOR m11(t11, b[1], t9);
        XNOR m12(t12, b[0], t11);
        XOR m13(t13, b[4], t11);
        XNOR m14(t14, b[2], t11);
        XOR m15(t15, b[6], b[5]);
        XNOR m16(t16, t9, t15);
        XOR m17(t17, t7, t16);
        XNOR m18(t18, b[1], t15);
        XOR m19(t19, t6, t18);

        assign g = {t5, t4, t10, t1, t16, t17, t11, t12};
        assign m = {t2, t7, t9, t18, t3, t19, t13, t6, t14, ~b[0]};
endmodule

/* top part: GF(2^4) multiplier and GF(2^4) square-scaler */
module Top (g, m, d);
        input [7:0] g;
        input [9:0] m;
        output [3:0] d;
        wire t1, t2, t3, t4, t5, t6, t7, t8, t9, t10, t11, t12, t13,
             t14, t15, t16, t17, t18, t19, t20, t21, t22, t23, t24;

        NAND m1(t1, g[7], g[3]);
        NOR m2(t2, g[6], g[2]);
        NOR m3(t3, g[7], g[3]);
        NAND m4(t4, m[9], m[8]);
        XOR m5(t13, t1, t2);
        XOR m6(t14, t3, t4);

        NAND m7(t5, m[5], m[4]);
        NAND m8(t6, m[3], m[2]);
        NAND m9(t7, m[7], m[6]);
        NOR m10(t8, m[3], m[2]);
        NOR m11(t9, m[5], m[4]);
        XOR m12(t15, t5, t6);
        XOR m13(t16, t7, t5);
        XOR m14(t17, t5, t8);
        XOR m15(t18, t7, t9);

        NAND m16(t10, g[5], g[1]);
        NAND m17(t11, g[4], g[0]);
        NAND m18(t12, m[1], m[0]);
        XOR m19(t19, t10, t11);
        XOR m20(t20, t10, t12);

        XOR m21(t21, t13, t15);
        XOR m22(t22, t14, t16);
        XOR m23(t23, t19, t17);
        XOR m24(t24, t20, t18);

        assign d[3] = t21;
        assign d[2] = t22;
        assign d[1] = t23;
        assign d[0] = t24;
endmodule

/* middle part: GF(2^4) inverse */
module Middle (p, l);
```

```
        input [3:0] p;
        output [3:0] l;
        wire t1, t2, t3, t4, t5, t6, t7, t8, t9, t10,
             t11, t12, t13, t14, t15;

        NAND m1(t1, p[3], p[0]);
        NOR m2(t2, t1, p[2]);
        NAND m3(t3, p[2], p[0]);
        XOR m4(t4, p[1], t3);
        NOR m5(t5, p[2], t4);
        NAND m6(t6, p[1], t4);
        NOR m7(t7, p[3], t4);
        NOR m8(t8, t7, t2);
        XNOR m9(t9, t5, t7);
        XNOR m10(t10, t9, p[3]);
        NAND m11(t11, t6, t8);
        NAND m12(t12, t8, p[1]);
        XNOR m13(t13, p[0], t12);
        NAND m14(t14, t1, p[2]);
        NAND m15(t15, t9, t14);

        assign l[3] = t13;
        assign l[2] = t11;
        assign l[1] = t15;
        assign l[0] = t10;
endmodule

/* bottom part: GF(2^4) multipliers */
module Bottom (g, m, l, e);
        input [7:0] g;
        input [9:0] m;
        input [3:0] l;
        output [17:0] e;
        wire k4, k3, k2, k1, k0;

        XOR m1(k4, l[3], l[2]);
        XOR m2(k3, l[3], l[1]);
        XOR m3(k2, l[2], l[0]);
        XOR m4(k1, k3, k2);
        XOR m5(k0, l[1], l[0]);

        NAND m6(e[17], g[2], l[2]);
        NAND m7(e[16], g[3], l[3]);
        NAND m8(e[15], m[8], k4);

        NAND m9(e[14], m[2], k1);
        NAND m10(e[13], m[4], k2);
        NAND m11(e[12], m[6], k3);

        NAND m12(e[11], g[0], l[0]);
        NAND m13(e[10], g[1], l[1]);
        NAND m14(e[9], m[0], k0);

        NAND m15(e[8], g[6], l[2]);
        NAND m16(e[7], g[7], l[3]);
        NAND m17(e[6], m[9], k4);

        NAND m18(e[5], m[3], k1);
        NAND m19(e[4], m[5], k2);
        NAND m20(e[3], m[7], k3);
```

```
        NAND m21(e[2], g[4], l[0]);
        NAND m22(e[1], g[5], l[1]);
        NAND m23(e[0], m[1], k0);
endmodule

/* output matrix */
module Output (e, Sb);
        input [17:0] e;
        output [7:0] Sb;
        wire E11, E10, E9, E8, E7, E6, E5, E4, E3, E2, E1, E0;
        wire t1, t2, t3, t4, t5, t6, t7, t8, t9, t10,
             t11, t12, t13, t14, t15, t16;

        XOR m1(E11, e[17], e[16]);
        XOR m2(E10, e[15], e[16]);
        XOR m3(E9, e[14], e[13]);
        XOR m4(E8, e[12], e[13]);
        XOR m5(E7, e[11], e[10]);
        XOR m6(E6, e[9], e[10]);
        XOR m7(E5, e[8], e[7]);
        XOR m8(E4, e[6], e[7]);
        XOR m9(E3, e[5], e[4]);
        XOR m10(E2, e[3], e[4]);
        XOR m11(E1, e[2], e[1]);
        XOR m12(E0, e[0], e[1]);

        XNOR m13(t1, E2, E0);
        XNOR m14(t2, E10, t1);
        XOR m15(t3, E6, t2);
        XOR m16(t4, E11, t3);
        XOR m17(t5, E9, t4);
        XOR m18(t6, E11, E7);
        XOR m19(t7, E5, t6);
        XNOR m20(t8, E4, E0);
        XNOR m21(t9, t5, t8);
        XOR m22(t10, t2, t9);
        XNOR m23(t11, E1, t1);
        XOR m24(t12, t7, t9);
        XNOR m25(t13, E5, t11);
        XNOR m26(t14, E8, t10);
        XNOR m27(t15, E3, t12);
        XOR m28(t16, t7, t11);

        assign Sb = {t3, t1, t15, t5, t13, t14, t8, t16};
endmodule
```

## C Verilog Code for the SM4 S-box

```
/* The SM4 S-box */
module SM4 (b, Sb);
        input [7:0] b;
        output [7:0] Sb;
        wire [7:0] g;
        wire [9:0] m;
        wire [3:0] p, l;
        wire [17:0] e;

        Input M1(b, g, m);
        Top M2(g, m, p);
        Middle M3(p, l);
        Bottom M4(g, m, l, e);
        Output M5(e, Sb);
endmodule

/* Gates implemented as modules to prevent
 unintentional optimization of the DC */
module XOR (t, a, b);
        output t;
        input a, b;
        xor(t, a, b);
endmodule

module XNOR (t, a, b);
        output t;
        input a, b;
        xnor(t, a, b);
endmodule

module NAND (t, a, b);
        output t;
        input a, b;
        nand(t, a, b);
endmodule

module NOR (t, a, b);
        output t;
        input a, b;
        nor(t, a, b);
endmodule

/* input matrix */
module Input (b, g, m);
        input [7:0] b;
        output [7:0] g;
        output [9:0] m;
        wire t1, t2, t3, t4, t5, t6, t7, t8, t9, t10,
             t11, t12, t13, t14, t15, t16, t17;

        XOR m1(t1, b[7], b[5]);
        XNOR m2(t2, b[5], b[1]);
        XNOR m3(t3, b[0], t2);
        XOR m4(t4, b[6], b[2]);
        XOR m5(t5, b[3], t3);
        XOR m6(t6, b[4], t1);
        XOR m7(t7, b[1], t5);
        XOR m8(t8, b[1], t4);
```

```
        XOR m9(t9, t6, t8);
        XOR m10(t10, t6, t7);
        XNOR m11(t11, b[3], t1);
        XNOR m12(t12, b[6], t9);
        XOR m13(t13, t4, t10);
        XOR m14(t14, t2, t11);
        XOR m15(t15, t12, t14);
        XOR m16(t16, t3, t12);
        XOR m17(t17, t11, t16);

        assign g = {t15, t14, ~b[0], t2, t5, t13, t7, t10};
        assign m = {t12, t9, t17, b[1], t11, t4, t16, t8, t3, t6};
endmodule

/* top part: GF(2^4) multiplier and GF(2^4) square-scaler */
module Top (g, m, d);
        input [7:0] g;
        input [9:0] m;
        output [3:0] d;
        wire t1, t2, t3, t4, t5, t6, t7, t8, t9, t10, t11, t12, t13,
             t14, t15, t16, t17, t18, t19, t20, t21, t22, t23, t24;

        NAND m1(t1, g[5], g[1]);
        NAND m2(t2, m[1], m[0]);
        NAND m3(t3, g[4], g[0]);
        NAND m4(t4, g[7], g[3]);
        NAND m5(t5, m[9], m[8]);
        NOR m6(t6, g[6], g[2]);
        NOR m7(t7, g[7], g[3]);
        NOR m8(t8, m[9], m[8]);
        NOR m9(t9, m[7], m[6]);
        NAND m10(t10, m[3], m[2]);
        NAND m11(t11, m[5], m[4]);
        NOR m12(t12, m[3], m[2]);

        XOR m13(t13, t1, t2);
        XOR m14(t14, t3, t2);
        XOR m15(t15, t4, t13);
        XOR m16(t16, t5, t14);
        XOR m17(t17, t9, t10);
        XOR m18(t18, t11, t12);
        XOR m19(t19, t6, t15);
        XOR m20(t20, t7, t16);

        XOR m21(t21, t19, t17);
        XOR m22(t22, t20, t18);
        XOR m23(t23, t8, t15);
        XOR m24(t24, t6, t16);

        assign d[3] = t21;
        assign d[2] = t22;
        assign d[1] = t23;
        assign d[0] = t24;
endmodule

/* middle part: GF(2^4) inverse */
module Middle (p, l);
        input [3:0] p;
        output [3:0] l;
        wire t1, t2, t3, t4, t5, t6, t7, t8, t9, t10,
```

```
            t11, t12, t13, t14, t15;

        NAND m1(t1, p[3], p[0]);
        NOR m2(t2, t1, p[2]);
        NAND m3(t3, p[2], p[0]);
        XOR m4(t4, p[1], t3);
        NOR m5(t5, p[2], t4);
        NAND m6(t6, p[1], t4);
        NOR m7(t7, p[3], t4);
        NOR m8(t8, t7, t2);
        XNOR m9(t9, t5, t7);
        XNOR m10(t10, t9, p[3]);
        NAND m11(t11, t6, t8);
        NAND m12(t12, t8, p[1]);
        XNOR m13(t13, p[0], t12);
        NAND m14(t14, t1, p[2]);
        NAND m15(t15, t9, t14);

        assign l[3] = t13;
        assign l[2] = t11;
        assign l[1] = t15;
        assign l[0] = t10;
endmodule

/* bottom part: GF(2^4) multipliers */
module Bottom (g, m, l, e);
        input [7:0] g;
        input [9:0] m;
        input [3:0] l;
        output [17:0] e;
        wire k4, k3, k2, k1, k0;

        XOR m1(k4, l[3], l[2]);
        XOR m2(k3, l[3], l[1]);
        XOR m3(k2, l[2], l[0]);
        XOR m4(k1, k3, k2);
        XOR m5(k0, l[1], l[0]);

        NAND m6(e[17], g[2], l[2]);
        NAND m7(e[16], g[3], l[3]);
        NAND m8(e[15], m[8], k4);

        NAND m9(e[14], m[2], k1);
        NAND m10(e[13], m[4], k2);
        NAND m11(e[12], m[6], k3);

        NAND m12(e[11], g[0], l[0]);
        NAND m13(e[10], g[1], l[1]);
        NAND m14(e[9], m[0], k0);

        NAND m15(e[8], g[6], l[2]);
        NAND m16(e[7], g[7], l[3]);
        NAND m17(e[6], m[9], k4);

        NAND m18(e[5], m[3], k1);
        NAND m19(e[4], m[5], k2);
        NAND m20(e[3], m[7], k3);

        NAND m21(e[2], g[4], l[0]);
        NAND m22(e[1], g[5], l[1]);
```

```
        NAND m23(e[0], m[1], k0);
endmodule

/* output matrix */
module Output (e, Sb);
        input [17:0] e;
        output [7:0] Sb;
        wire E11, E10, E9, E8, E7, E6, E5, E4, E3, E2, E1, E0;
        wire t1, t2, t3, t4, t5, t6, t7, t8, t9, t10,
             t11, t12, t13, t14, t15, t16;

        XOR m1(E11, e[17], e[16]);
        XOR m2(E10, e[15], e[16]);
        XOR m3(E9, e[14], e[13]);
        XOR m4(E8, e[12], e[13]);
        XOR m5(E7, e[11], e[10]);
        XOR m6(E6, e[9], e[10]);
        XOR m7(E5, e[8], e[7]);
        XOR m8(E4, e[6], e[7]);
        XOR m9(E3, e[5], e[4]);
        XOR m10(E2, e[3], e[4]);
        XOR m11(E1, e[2], e[1]);
        XOR m12(E0, e[0], e[1]);

        XOR m13(t1, E9, E7);
        XOR m14(t2, E1, t1);
        XOR m15(t3, E3, t2);
        XOR m16(t4, E5, E3);
        XOR m17(t5, E4, t4);
        XOR m18(t6, E4, E0);
        XOR m19(t7, E11, E7);
        XOR m20(t8, t1, t4);
        XOR m21(t9, t1, t6);
        XOR m22(t10, E2, t5);
        XOR m23(t11, E10, E8);
        XNOR m24(t12, t3, t11);
        XOR m25(t13, t10, t12);
        XNOR m26(t14, t3, t7);
        XNOR m27(t15, E10, E6);
        XOR m28(t16, t6, t14);

        assign Sb = {t15, t13, t8, t14, t11, t9, t12, t16};
endmodule
```