A depth-16 circuit for the AES S-box

Joan Boyar*
joan@imada.sdu.dk

René Peralta†
peralta@nist.gov

Abstract

New techniques for reducing the depth of circuits for cryptographic applications are described and applied to the AES S-box. These techniques also keep the number of gates quite small. The result, when applied to the AES S-box, is a circuit with depth 16 and only 128 gates. For the inverse, it is also depth 16 and has only 127 gates. There is a shared middle part, common to both the S-box and its inverse, consisting of 63 gates.

Keywords: AES; S-box; finite field inversion; circuit complexity; circuit depth.

1 Introduction

Constructing optimal combinational circuits is an intractable problem under almost any meaningful metric (gate count, depth, energy consumption, etc.). In practice, no known techniques can reliably find optimal circuits for functions with as few as eight Boolean inputs and one Boolean output (there are 2^{256} such functions). Thus, heuristic or specialized techniques are necessary in practice.

Recently, Nogami et.al. [10] presented a technique for reducing circuit depth in the forward direction of the AES S-box. Their technique was primarily to choose mixed bases for the tower-of-fields architecture in such a way that the matrices doing the linear transformations at the top and bottom had at most 4 ones in every row. In this way they were able to compute these transformations in depth 2 each, for a total of depth 4. Their technique cannot be simultaneously applied to the AES circuit in the reverse direction, as the inverse matrices do not have at most 4 ones in every row. We present more general techniques that are less dependent on the actual representation of the fields. These techniques allow us to produce circuits which are both smaller and shorter in both directions. Although the size of our construction is larger than the one reported in [1], which had only 115 gates, it is comparable to previous efforts to make a compact circuit (see [3, 8]). The latter two constructions have depths between 25 and 27. Nogami et. al. improved this, for the forward direction of the S-Box, to depth 22 at the cost of increasing size to 148. Our new circuits have depth 16 in both directions, size 128 in the forward direction, and size 127 in the reverse direction. The part that is shared between the forward and reverse directions is of size 63.

*Department of Mathematics and Computer Science, University of Southern Denmark. Partially supported by the Danish Natural Science Research Council (SNF). Some of this work was done while visiting the University of California, Irvine.

†Information Technology Laboratory, National Institute of Standards and Technology.

2 Combinational circuit optimization

Many different logically complete bases are possible for circuits. Since the operations in the basis (XOR, AND) are equivalent to addition and multiplication modulo 2 (i.e., in $GF(2)$), much work on circuits for cryptographic functions uses this basis. For logical completeness, the negation operation (or the constant 1) is needed as well. In $GF(2)$, negation corresponds to $x + 1$. For technical reasons, and for an accurate gate-count, we use XNOR gates ($XNOR(x, y) = x + y + 1$ in $GF(2)$) instead of negation. Components free of AND gates are called *linear*; our emphasis is on these components, though *nonlinear* components are also optimized.

Under the basis (XOR,XNOR,AND), classic results by Shannon [12] and Lupanov [7] show that almost all predicates on n bits have circuit complexity about $\frac{2^n}{n}$. The *multiplicative complexity* of a function is the number of AND gates necessary and sufficient to compute the function. Analogous to the Shannon-Lupanov bound, it was shown in [2] that almost all Boolean predicates on n bits have multiplicative complexity about $2^{\frac{n}{2}}$. Strictly speaking, these theorems say nothing about the class of functions with polynomial circuit complexity. However, it is reasonable to expect that, in practice, the multiplicative complexity of functions is significantly smaller than their Boolean complexity. This is one of the principles that guide our design strategy.

Circuits with few AND gates will naturally have large sections which are purely linear. Boyar and Peralta [1] have used this insight to construct circuits much smaller than previously known for a variety of applications (see `http://cs-www.cs.yale.edu/homes/peralta/CircuitStuff/CMT.html`). The heuristic is a two-step process which first reduces multiplicative complexity and then optimizes linear components.

The work presented in this paper is based on the idea that a short circuit may be obtained by starting from a small (i.e. optimized for size) circuit and performing three types of depth-decreasing optimizations:

- apply a greedy heuristic to re-synthesize linear components into lower-depth constructions;
- use techniques from automatic theorem proving to re-synthesize non-linear components into lower-depth constructions;
- do simple depth-shortening local replacement along critical paths.

As these steps are automatic heuristics that do not consider size, we also coded a final step that reduces size via depth-preserving local replacement. These techniques are explained below. They will often increase the size of the circuit, but we start with a small circuit and the techniques are designed to minimize the increase.

3 The tower field construction

There are many representations of $GF(2^8)$. We construct

- $GF(2^2)$ by adjoining a root W of $x^2 + x + 1$ over $GF(2)$;
- $GF(2^4)$ by adjoining a root Z of $x^2 + x + W^2$ over $GF(2^2)$.

- $GF(2^8)$ by adjoining a root Y of $x^2 + x + WZ$ over $GF(2^4)$.

As does Canright in [3], we represent $GF(2^2)$ using the basis (W, W^2), $GF(2^4)$ using the basis (Z^2, Z^8), and $GF(2^8)$ using the basis (Y, Y^{16}).

Let $A = a_0Y + a_1Y^{16}$ be an arbitrary element in $GF(2^8)$. Following [6], the inverse of A can be computed as follows:

$$\begin{aligned}
A^{-1} &= (AA^{16})^{-1}A^{16} \\
&= ((a_0Y + a_1Y^{16})(a_1Y + a_0Y^{16}))^{-1}(a_1Y + a_0Y^{16}) \\
&= ((a_0^2 + a_1^2)Y^{17} + a_0a_1(Y^2 + Y^{32}))^{-1}(a_1Y + a_0Y^{16}) \\
&= ((a_0 + a_1)^2Y^{17} + a_0a_1(Y + Y^{16})^2)^{-1}(a_1Y + a_0Y^{16}) \\
&= ((a_0 + a_1)^2WZ + a_0a_1)^{-1}(a_1Y + a_0Y^{16}).
\end{aligned}$$

Thus computation of the inverse in $GF(2^8)$ can be done using operations in $GF(2^4)$ as follows:

$$\begin{aligned}
T_1 &= (a_0 + a_1) \\
T_2 &= (WZ)T_1^2 \\
T_3 &= a_0a_1 \\
T_4 &= T_2 + T_3 \\
T_5 &= T_4^{-1} \\
T_6 &= T_5a_1 \\
T_7 &= T_5a_0
\end{aligned}$$

The result is $A^{-1} = T_6Y + T_7Y^{16}$.

The $GF(2^4)$ operations involved are addition, multiplication, square and scale by WZ, and inverse. Of these, only multiplication and inverse turn out to be non-linear. We derive a standard $GF(2^4)$ multiplication circuit by reduction to $GF(2^2)$ operations. The standard inversion circuit, however, has more gates and depth than necessary. Hence we derive a better circuit here.

Let $\Delta = (x_1W + x_2W^2)Z^2 + (x_3W + x_4W^2)Z^8$ be an arbitrary element in $GF(2^4)$. It is not hard to verify that its inverse is $\Delta^{-1} = (y_1W + y_2W^2)Z^2 + (y_3W + y_4W^2)Z^8$ where the y_i's satisfy the following polynomials over $GF(2)$:

- $y_1 = x_2x_3x_4 + x_1x_3 + x_2x_3 + x_3 + x_4$
- $y_2 = x_1x_3x_4 + x_1x_3 + x_2x_3 + x_2x_4 + x_4$
- $y_3 = x_1x_2x_4 + x_1x_3 + x_1x_4 + x_1 + x_2$
- $y_4 = x_1x_2x_3 + x_1x_3 + x_1x_4 + x_2x_4 + x_2$

The heuristic Boyar and Peralta used in [1] to compute the y_i's was inspired by methods from automatic theorem proving. Consider an arbitrary predicate f on n inputs. We refer to the last column of the truth table for f as the *signal* of f. The columns in the truth table corresponding to each of the inputs to f are *known* signals. A search for a circuit for f starts

with this set S of known signals. If u, v are known signals for functions g, h respectively, then the bit-wise XOR (AND) of u and v is the signal for the predicate $g \oplus h$ ($g \wedge h$). We can *grow* the set S by adding the XOR of randomly chosen signals. We call this step an *XOR round*. The analogous step where the AND of signals is added to S is called an *AND round*. Each round is parameterized by the number of new signals added and the maximum number of AND gates allowed. In either an XOR round or an AND round, two signals are not combined if doing so creates a circuit with more AND gates than is allowed. The heuristic alternates between XOR and AND rounds until the target signal is found or the set S becomes too large. In the latter case, since this is a randomized procedure, we start again.

Boyar and Peralta [1] used this heuristic to find a circuit with only 5 AND gates and 11 XOR gates, but depth 9. In terms of size, this was a significant improvement over previous constructions. None of these constructions, however, was concerned with depth. To minimize depth, we used a different parametrization of these techniques and found a circuit with depth 4 and size 17. The straight-line program for the circuit is in Figure 1 (arithmetic is over $GF(2)$).

t_1	=	$x_2 + x_3$	t_2	=	$x_2 \times x_0$	t_3	=	$x_1 + t_2$
t_4	=	$x_0 + x_1$	t_5	=	$x_3 + t_2$	t_6	=	$t_5 \times t_4$
t_7	=	$t_3 \times t_1$	t_8	=	$x_0 \times x_3$	t_9	=	$t_4 \times t_8$
t_{10}	=	$t_4 + t_9$	t_{11}	=	$x_1 \times x_2$	t_{12}	=	$t_1 \times t_{11}$
t_{13}	=	$t_1 + t_{12}$	y_0	=	$t_2 + t_{13}$	y_1	=	$x_3 + t_7$
y_2	=	$t_2 + t_{10}$	y_3	=	$x_1 + t_6$			

Figure 1: Inversion in $GF(2^4)$. Input is (x_0, x_1, x_2, x_3) and output is (y_0, y_1, y_2, y_3).

4 A greedy heuristic for linear components

The largest linear components in our circuit are the top linear and bottom linear components. These components contain more than the linear operations defined explicity in the definition of the AES S-box and the matrices to do the basis changes. This is because they include some of the finite field inversion operations. The top linear component is defined by the matrix U, a 22×8 matrix (Figure 2). One can compute all 22 of the required outputs with only 23 XOR gates, and 23 are necessary [1, 5, 4]. But these results do not attempt to minimize depth (the depth is 7). Since there are only 8 columns in this matrix, each of the 22 outputs could clearly be calculated independently using depth at most 3, simply by using a balanced binary tree with the inputs as leaves. The challenge is to achieve the low depth without increasing the number of XOR gates drastically. The algorithm below does this. (Note that although the linear transformation at the top of Nogami et.al.'s circuit only has depth 2, they have XOR gates at depth 3, so their top linear component also has depth at least 3.)

The bottom linear component is defined by the matrix B, an 8×18 matrix (Figure 3). The row with the largest Hamming weight (number of ones = number of variables added together) has 12 ones, so depth at most 4 is possible for this component.

The smallest circuits for these two matrices, U and B, use the concept of cancellation

$$U = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

Figure 2: The top linear transformation U.

$$B = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}$$

Figure 3: The bottom linear transformation B.

of variables. Note that in [1], the variable y_{11} is computed as $y_{20} \oplus y_9$. Since $y_{20} = x_0 \oplus x_1 \oplus x_3 \oplus x_4 \oplus x_5 \oplus x_6$ and $y_9 = x_0 \oplus x_3$, the result is $y_{11} = x_1 \oplus x_4 \oplus x_5 \oplus x_6$; the x_0 and x_3 are cancelled.

When attempting to find small, low-depth circuits for a linear component, one expects that cancellation of variables will be of limited help, since it would often require that something with a large Hamming weight has already been computed, before adding one to the depth at the gate where the cancellation occurs. Thus, it seems reasonable to start with a technique which does not allow cancellation, and then try to add cancellation afterwards where it helps.

We modify Paar's technique [11], a greedy approach which produces cancellation-free programs. Paar's technique keeps a list of variables computed, which is initially only the inputs. Then it repeatedly determines which two variables, XORed together, occur in most outputs. One such pair is selected and XORed together. This result is added as a new variable which appears in all outputs where both variables previously appeared. This is repeated until everything has been computed. Paar's technique is implemented by starting with the initial matrix and adding columns corresponding to the new variables which are added. When a new column is added, this corresponds to adding two variables, u and v. In all rows in the matrix which currently have a one in both of the columns corresponding to u and v, those two ones are changed to zeros, and a one is placed in the corresponding row of the new column. All other values in the new column are set to zero.

The Low_Depth_Greedy algorithm maintains the greedy approach of Paar's technique, but only allows this greediness as long as it does not increase the circuit's depth unnecessarily. Assume that k is the depth we are aiming for, i.e. $k = \lceil \log_2(w) \rceil$, where w is the largest Hamming weight of any row. The Low_Depth_Greedy algorithm has k phases, starting with 0. At the beginning of a new phase, we check if any row has Hamming weight two. Since there must be an additional gate to produce that output, we produce it at the beginning of the phase so that it affects all counting in the current phase. During phase $i \geq 0$, no row in the current matrix has Hamming weight more than 2^{k-i} and only inputs or gates already produced at depth i or less are considered as possible inputs to gates in phase i. Thus, the depth of gates in phase i is at most $i + 1$. When choosing two possible inputs for gates, one chooses a pair which occurs most frequently in the current rows, with the restriction, of course, that both inputs are at level i or less. Pseudo-code for this algorithm is given in Figure 4.

This algorithm produces a minimum depth (optimal depth) circuit.

Theorem 1 *When given an $m \times n$ 0-1 matrix, M, with maximum Hamming weight at most 2^k in any row, Algorithm Low_Depth_Greedy, produces a correct, depth-k circuit for computing the linear component defined by the matrix. The running time is $O(mt^3)$, where t is the final number of columns and is at most $mn + n - m$.*

Proof. If one considers the inputs as being produced at depth zero, in phase i of the algorithm, only variables which have been produced at depth at most i are considered as possible inputs to XOR gates, so the XOR gates produced have depth at most $i + 1$. This is maintained inductively by only considering columns between 1 and ip, and ip is reset at the end of each phase to the last column produced in that phase. Since the algorithm maintains that at the beginning of phase i, no more than 2^{k-i} of the current variables have to be XORed to produce any output, the algorithm terminates in phase $k - 1$, giving maximum depth k. Note that it will always be possible to proceed from phase i to phase

```
Low_Depth_Greedy(M, m, n, k):
{ M is an m × n 0-1 matrix with Hamming weight at most 2^k in any row}
    s := n + 1 { index of the next column }
    i := 0
    ip := n   { columns up to n had depth at most i }
    while there is some row in M with Hamming weight > 2^{k-i-1} do
    { Phase i }
        if some row ℓ in M with weight 2
            had weight 2 at the beginning of the phase
            then let j1 and j2 be the columns in row ℓ with ones
            else find two columns 1 ≤ j1, j2 ≤ ip
                                  which maximize |{ℓ | M[ℓ, j1] = M[ℓ, j2] = 1}|
        add an XOR gate with inputs from the variables for columns j1, j2
        the output variable produced will correspond to column s
        for ℓ = 1 to m do
            if M[ℓ, j1] = M[ℓ, j2] = 1
                then M[ℓ, j1] := 0; M[ℓ, j2] := 0; M[ℓ, s] := 1
                else M[ℓ, s] := 0
        s := s + 1
    ip := s − 1   { keep track of which gates had depth at most i }
    i := i + 1
```

Figure 4: Algorithm for creating a minimum depth circuit for linear components

$i + 1$, since combining the at most 2^{k-i} ones any row by pairs will reduce the number of ones to at most half as many, at most 2^{k-i-1}.

For each XOR gate added, the algorithm checks every pair of columns between 1 and $ip < s$, where s is the new column being added. For each of these pairs of columns, one checks for each row if both entries corresponding to these columns are one and then does some updating. The number of rows is n, so the total running time is $O(nt^3)$. Since there are at most n ones in every row, each row will be computed using at most $n-1$ XORs, and all m rows will be computed with at most $m(n-1)$ XORs. There are n columns initially, so in all $t \leq mn + n - m$. □

Another possibility for an algorithm to produce optimal depth circuits for linear components would have been to finish with all pairs of inputs before continuing to pairs involving gates at depth one, and then to finish with all pairs at depth one (or involving the possibly one remaining input which has not been paired), etc. However, the method chosen here allows more flexibility in choosing gates, thus allowing more possibilities to create gates which can be used more than once.

After an initial attempt at minimizing depth and size in the entire circuit, we may be able to further decrease the number of gates in the top linear component since not all the XOR gates at level three (an output of the top linear component) would necessarily increase the total depth if they were at level four or more (for the AES S-box, k and $k + 1$ more generally). Or, on the other hand, one might be able to reduce the depth even more

by calculating some outputs of the top linear component at lower depth than the depth indicated by the matrix row with largest Hamming weight, if these "outputs" are on the critical path.

It is easy to determine which outputs of the top linear component could be allowed to be at a larger depth or should be at a lower depth if possible, using a program which calculates the depth and height of every gate. If all of the outputs of the top linear component which have depth and height values adding up to exactly the total depth of the circuit are such that they could have been calculated at lower depth than their current depth, then one can probably reduce the depth of the circuit. On the other hand, when these values add up to less than the total depth of the circuit, there is some *slack* at that gate. For XOR gates at depth 3 (in an AES S-box circuit) which have slack, one can check if they are the sum of any two of the other outputs of the top-linear part. If they are, these other outputs were computed at depth 3, so adding them together only gives depth 4, which is acceptable when the output was originally created at a gate with slack. Note that cancellation of variables should be allowed here.

The Low_Depth_Greedy algorithm can be modified to take advantage of slackness. In this case, an extra array *Factor* is initialized for each input to the linear transformation. Rows with no slack are given the value 1, and rows that could be at j levels further down than the minimum are given the value 2^j in *Factor*. Then, when checking if one should proceed to the next phase, rather than check if all rows have Hamming weight at most 2^{k-i} for phase i, one checks if its Hamming weight divided by its value in *Factor* is at most 2^{k-i}. This allows the possibility of choosing inputs required for these outputs at a larger depth. These techniques were not actually necessary to produce the circuits found.

5 Reducing depth in linear components

There are straight-forward techniques for reducing depth in linear components via local replacement. Consider any gate in such a component. The output produced there is the XOR of several values (either inputs or outputs from other gates). These values can be XORed in any order to get this result. Thus, for example, suppose $g = g_1 \oplus g_2$, g_1 is at depth d_1 and $g_1 = g_3 \oplus g_4$, g_2 is at depth d_2, and g_3 is at depth d_3. If d_2 and d_3 are at depth at most $d_1 - 2$, then calculating $h_1 = g_2 \oplus g_3$ and $h_2 = h_1 \oplus g_4$ results in h_1 computing the same result as g, but at depth one lower. If the result computed at g_1 was not used anywhere else in the circuit, then this does not increase the total number of gates. However, if g_1 is used elsewhere, it would still need to be computed, and the number of gates would increase by one.

6 The circuits

The depth-16 circuits are shown in Figures 5, 6, 7, 8, and 9. Note that the addition and multiplication operations are modulo 2, so they are XOR and AND operations. The # operation is an XNOR (adding modulo 2 and then complementing the result). We used Algorithm Low_Depth_Greedy for the four linear transformations (here, we do not include the binary matrices corresponding to the transformations in the reverse direction of the AES S-box). The circuits are divided into three components: top linear transformations (Figures 5 and 6), shared non-linear component (Figure 7), and bottom linear transformations

(Figures 8 and 9).

```
T1 = U0 + U3
T2 = U0 + U5
T3 = U0 + U6
T4 = U3 + U5
T5 = U4 + U6
T6 = T1 + T5
T7 = U1 + U2
```

```
T8 = U7 + T6
T9 = U7 + T7
T10 = T6 + T7
T11 = U1 + U5
T12 = U2 + U5
T13 = T3 + T4
T14 = T6 + T11
```

```
T15 = T5 + T11
T16 = T5 + T12
T17 = T9 + T16
T18 = U3 + U7
T19 = T7 + T18
T20 = T1 + T19
T21 = U6 + U7
```

```
T22 = T7 + T21
T23 = T2 + T22
T24 = T2 + T10
T25 = T20 + T17
T26 = T3 + T16
T27 = T1 + T12
```

Figure 5: Top linear transform in forward direction.

```
T23 = U0 + U3
T22 = U1 # U3
T2 = U0 # U1
T1 = U3 + U4
T24 = U4 # U7
R5 = U6 + U7
T8 = U1 # T23
```

```
T19 = T22 + R5
T9 = U7 # T1
T10 = T2 + T24
T13 = T2 + R5
T3 = T1 + R5
T25 = U2 # T1
R13 = U1 + U6
```

```
T17 = U2 # T19
T20 = T24 + R13
T4 = U4 + T8
R17 = U2 # U5
R18 = U5 # U6
R19 = U2 # U4
Y5 = U0 + R17
```

```
T6 = T22 + R17
T16 = R13 + R19
T27 = T1 + R18
T15 = T10 + T27
T14 = T10 + R18
T26 = T3 + T16
```

Figure 6: Top linear transform in reverse direction.

```
M1 = T13 x T6
M2 = T23 x T8
M3 = T14 + M1
M4 = T19 x D
M5 = M4 + M1
M6 = T3 x T16
M7 = T22 x T9
M8 = T26 + M6
M9 = T20 x T17
M10 = M9 + M6
M11 = T1 x T15
M12 = T4 x T27
M13 = M12 + M11
M14 = T2 x T10
M15 = M14 + M11
M16 = M3 + M2
```

```
M17 = M5 + T24
M18 = M8 + M7
M19 = M10 + M15
M20 = M16 + M13
M21 = M17 + M15
M22 = M18 + M13
M23 = M19 + T25
M24 = M22 + M23
M25 = M22 x M20
M26 = M21 + M25
M27 = M20 + M21
M28 = M23 + M25
M29 = M28 x M27
M30 = M26 x M24
M31 = M20 x M23
M32 = M27 x M31
```

```
M33 = M27 + M25
M34 = M21 x M22
M35 = M24 x M34
M36 = M24 + M25
M37 = M21 + M29
M38 = M32 + M33
M39 = M23 + M30
M40 = M35 + M36
M41 = M38 + M40
M42 = M37 + M39
M43 = M37 + M38
M44 = M39 + M40
M45 = M42 + M41
M46 = M44 x T6
M47 = M40 x T8
M48 = M39 x D
```

```
M49 = M43 x T16
M50 = M38 x T9
M51 = M37 x T17
M52 = M42 x T15
M53 = M45 x T27
M54 = M41 x T10
M55 = M44 x T13
M56 = M40 x T23
M57 = M39 x T19
M58 = M43 x T3
M59 = M38 x T22
M60 = M37 x T20
M61 = M42 x T1
M62 = M45 x T4
M63 = M41 x T2
```

Figure 7: Shared part of AES S-box circuit ($D = U7$ in the forward direction and $D = Y5$ in the reverse direction).

The circuits were generated automatically using randomization for tie-resolution. Different runs of our code yield depth 16 consistently. However, size can vary by a few gates. As

```
L0 = M61 + M62
L1 = M50 + M56
L2 = M46 + M48
L3 = M47 + M55
L4 = M54 + M58
L5 = M49 + M61
L6 = M62 + L5
L7 = M46 + L3
L8 = M51 + M59
L9 = M52 + M53
```

```
L10 = M53 + L4
L11 = M60 + L2
L12 = M48 + M51
L13 = M50 + L0
L14 = M52 + M61
L15 = M55 + L1
L16 = M56 + L0
L17 = M57 + L1
L18 = M58 + L8
L19 = M63 + L4
```

```
L20 = L0 + L1
L21 = L1 + L7
L22 = L3 + L12
L23 = L18 + L2
L24 = L15 + L9
L25 = L6 + L10
L26 = L7 + L9
L27 = L8 + L10
L28 = L11 + L14
L29 = L11 + L17
```

```
S0 = L6 + L24
S1 = L16 # L26
S2 = L19 # L28
S3 = L6 + L21
S4 = L20 + L22
S5 = L25 + L29
S6 = L13 # L27
S7 = L6 # L23
```

Figure 8: Bottom linear transform in forward direction. Outputs are $S0 \ldots S7$.

```
P0 = M52 + M61
P1 = M58 + M59
P2 = M54 + M62
P3 = M47 + M50
P4 = M48 + M56
P5 = M46 + M51
P6 = M49 + M60
P7 = P0 + P1
P8 = M50 + M53
P9 = M55 + M63
```

```
P10 = M57 + P4
P11 = P0 + P3
P12 = M46 + M48
P13 = M49 + M51
P14 = M49 + M62
P15 = M54 + M59
P16 = M57 + M61
P17 = M58 + P2
P18 = M63 + P5
P19 = P2 + P3
```

```
P20 = P4 + P6
P22 = P2 + P7
P23 = P7 + P8
P24 = P5 + P7
P25 = P6 + P10
P26 = P9 + P11
P27 = P10 + P18
P28 = P11 + P25
P29 = P15 + P20
W0 = P13 + P22
```

```
W1 = P26 + P29
W2 = P17 + P28
W3 = P12 + P22
W4 = P23 + P27
W5 = P19 + P24
W6 = P14 + P23
W7 = P9 + P16
```

Figure 9: Bottom linear transform in reverse direction. Outputs are $W0 \ldots W7$.

long as the topology derived from the tower-of-fields method is maintained, we conjecture that it is unlikely that the size of the circuits can be significantly reduced without increasing the depth. We also conjecture that it is unlikely that the depth can be reduced without significantly increasing size. Of course, if the logical base is expanded, we may be able to do better. For example, if NAND gates are used in the circuit for inversion in $GF(2^4)$, it is not hard to reduce the number of gates by two without increasing the depth. Since there are only 256 possible inputs, we verified the circuits fully against the specifications in [9].

References

[1] J. Boyar and R. Peralta. A new combinational logic minimization technique with applications to cryptology. In P. Festa, editor, *SEA*, volume 6049 of *Lecture Notes in Computer Science*, pages 178–189. Springer, 2010.

[2] J. Boyar, R. Peralta, and D. Pochuev. On the multiplicative complexity of Boolean functions over the basis $(\wedge, \oplus, 1)$. *Theoretical Computer Science*, 235:43–57, 2000.

[3] D. Canright. A very compact Rijndael S-box. Technical Report NPS-MA-05-001, Naval Postgraduate School, 2005.

[4] C. Fuhs and P. Schneider-Kamp. Optimizing the AES S-Box using SAT. In *Proceedings of the 8th International Workshop on the Implementation of Logics*, 2010.

[5] C. Fuhs and P. Schneider-Kamp. Synthesizing shortest linear straight-line programs over GF(2) using SAT. In O. Strichman and S. Szeider, editors, *SAT*, volume 6175 of *Lecture Notes in Computer Science*, pages 71–84. Springer, 2010.

[6] T. Itoh and S. Tsujii. A fast algorithm for computing multiplicative inverses in $GF(2^m)$ using normal bases. *Inf. Comput.*, 78(3):171–177, 1988.

[7] O. B. Lupanov. A method of circuit synthesis. *Izvestia V.U.Z. Radiofizika*, 1:120–140, 1958.

[8] S. Morioka and A. Satoh. An optimized S-Box circuit architecture for low power AES design. In *Revised Papers from the 4th International Workshop on Cryptographic Hardware and Embedded Systems*, pages 172–186, London, UK, 2003. Springer-Verlag.

[9] NIST. *Advanced Encryption Standard (AES) (FIPS PUB 197)*. National Institute of Standards and Technology, November 2001.

[10] Y. Nogami, K. Nekado, T. Toyota, N. Hongo, and Y. Morikawa. Mixed bases for efficient inversion in $f(((2^2)^2)^2)$ and conversion matrices of subbytes of AES. In S. Mangard and F.-X. Standaert, editors, *CHES*, volume 6225 of *Lecture Notes in Computer Science*, pages 234–247. Springer, 2010.

[11] C. Paar. Optimized arithmetic for Reed-Solomon encoders. In *IEEE International Symposium on Information Theory*, page 250, 1997.

[12] C. E. Shannon. The synthesis of two-terminal switching circuits. *Bell System Tech. J.*, 28:59–98, 1949.