A Small Depth-16 Circuit for the AES S-Box

Joan Boyar*[1] and René Peralta[2]

[1] Department of Mathematics and Computer Science
University of Southern Denmark, joan@imada.sdu.dk
[2] Information Technology Laboratory, NIST, rene.peralta@nist.gov

Abstract. New techniques for reducing the depth of circuits for cryptographic applications are described. 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. The best previous comparable design for the AES S-Box has depth 22 and size 148 [12].

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. The depth of a circuit is the number of gates on a longest path. Reducing depth generally leads to faster circuits. Reducing the number of gates is important for reducing area and power consumption. The aim is to reduce both simultaneously, but there is often a trade-off between these two goals.

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. This platform-independent basis leads to easy comparison with previous implementations. Note that changing AND gates to NANDs is not difficult.

The Advanced Encryption Standard (AES) is a widely used symmetric key encryption system that was adopted as a standard by the U.S. government in 2002. The nonlinear part of AES is the SubBytes step, which is often referred to as the S-Box. The original specification of AES [11] defined the S-Box as the multiplicative inverse in the field $GF(2^8)$, followed by an affine transformation.

* Partially supported by the Danish Council for Independent Research, Natural Sciences.

However, the S-Box can also be implemented using table look-up. This requires a ROM with 2048 bits. However, ROMs are not efficient in terms of size/power consumption. Satoh et al. [15], using a *tower-of-fields* approach to compute the S-Box, designed a combinational circuit that can be implemented with an area under 1/4 of previous ROM-based designs. Other work, for example [3, 10, 1], has improved on this size.

Recently, Nogami et al. [12] 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 8×8 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 than those in [12]. Although the size of our construction is larger than our size-optimized circuit [1] (which had only 115 gates but had depth 28) it is comparable to previous efforts to make a small circuit (the constructions in [3] and [10] have depths 25 and 27, respectively). The work of Nogami et al. improved on the forward direction of the S-Box to depth 22 at the cost of increasing the number of gates 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.

A general overview of the technique is described in Section 2, and the technique used for the nonlinear component, inversion in $GF(2^4)$, is described in Section 3. A greedy heuristic for linear components is described in Section 4. Further techniques for depth and size reduction are described in Section 5. The AES S-Box circuits are presented in Section 6 and conclusions in Section 7.

2 Combinational Circuit Optimization

Under the basis (XOR,XNOR,AND), classic results by Shannon [16] and Lupanov [9] 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. The authors [1] and Courtois et al. [5] have used this insight to construct circuits much smaller than previously known for a variety of applications. 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 optimizations with automatic heuristics:

- use techniques from automatic theorem proving to re-synthesize non-linear components into lower-depth constructions;
- apply a greedy heuristic to re-synthesize linear components into lower-depth constructions;
- perform depth-shortening and size-reducing local replacement.

These techniques are explained below. They will often increase the size of the circuit, but we start with our small circuit from [1], and the techniques are designed to minimize the size increase.

3 The Tower Field Construction: A Nonlinear Component

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 $GF(2^4)$.

Thus, there is a tower of fields, each one contained in the previous, and one can do multiplication and inverse operations in the larger fields efficiently by implementing the relevant operations in the smaller fields efficiently, as in [8].

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 [8], 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:

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

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 $\Delta = (x_0W + x_1W^2)Z^2 + (x_2W + x_3W^2)Z^8$ be an arbitrary element in $GF(2^4)$. It is not hard to verify that its inverse is $\Delta^{-1} = (y_0W + y_1W^2)Z^2 + (y_2W + y_3W^2)Z^8$ where the y_i's satisfy the following polynomials over $GF(2)$ (see [1]):

– $y_0 = x_1x_2x_3 + x_0x_2 + x_1x_2 + x_2 + x_3$
– $y_1 = x_0x_2x_3 + x_0x_2 + x_1x_2 + x_1x_3 + x_3$
– $y_2 = x_0x_1x_3 + x_0x_2 + x_0x_3 + x_0 + x_1$
– $y_4 = x_0x_1x_2 + x_0x_2 + x_0x_3 + x_1x_3 + x_1$

The heuristic we 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 parametrized 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.

In [1] we 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 (e.g. [13, 4]). All these constructions, however, were mostly concerned with size. To minimize depth without incurring a large increase in size, we use the search techniques described above, but add code to discard search paths that either violate given depth constraints or size constraints. With this modification, we found a circuit with depth 4 and size 17. The straight-line program for the circuit is in Figure 1 (arithmetic is over $GF(2)$).

4 A Greedy Heuristic for Linear Components

After this small circuit for inversion in $GF(2^4)$ was found, it was set into the original circuit for AES (we started with the circuit in [1]), and maximal linear components, connected components of the circuit not containing any AND gates, were found. The largest linear components in our circuit are the top linear and bottom linear components. These components contain more than the

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

Fig. 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).

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

Fig. 2. The transpose U^T of the top linear transformation U.

linear operations defined explicitly 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), meaning that 22 linear combinations of the 8 inputs are calculated. One can compute all 22 of the required outputs with only 23 XOR gates, and 23 are necessary [1, 7, 6]. 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 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

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

Fig. 3. The bottom linear transformation B.

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 [14], 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 existing 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. This new 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,*

```
**Low_Depth_Greedy**$(M, m, n, k)$:
{ $M$ is an $m \times n$ 0-1 matrix with Hamming weight at most $2^k$ in any row}
    $s := n + 1$ { index of the next column }
    **for** $i := 0$ to $k - 1$ **do**
        $ip := s - 1$  { keep track of which gates 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 $\ell$ in $M$ with weight 2
                had weight 2 at the beginning of the phase
                **then** let $j_1$ and $j_2$ be the columns in row $\ell$ with ones
                **else** find two columns $1 \leq j_1, j_2 \leq ip$
                    which maximize $|\{\ell \mid M[\ell, j_1] = M[\ell, j_2] = 1\}|$
            add an XOR gate with inputs from the variables for columns $j_1, j_2$
            the output variable produced will correspond to column $s$
            **for** $\ell = 1$ to $m$ **do**
                **if** $M[\ell, j_1] = M[\ell, j_2] = 1$
                    **then** $M[\ell, j_1] := 0; M[\ell, j_2] := 0; M[\ell, s] := 1$
                    **else** $M[\ell, s] := 0$
            $s := s + 1$
```

Fig. 4. Algorithm for creating a minimum depth circuit for linear components

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 $i+1$, 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 m, so the total running time is $O(mt^3)$. 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.

5 Local Optimizations

After applying the techniques discussed in the previous sections, highly localized optimizations may be possible.

We are 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 a longest 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 modifications of the Low_Depth_Greedy algorithm were not actually necessary to produce the circuits found.

Finally, 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. For example, suppose $g = g_1 \oplus g_2$, $g_1 = g_3 \oplus g_4$, and g_1 is at depth d. If the depths of g_2 and g_3 are at most $d-2$, then g is at depth $d+1$ and g_4 is at depth at $d-1$. Now calculating $h_1 = g_2 \oplus g_3$ and $h_2 = h_1 \oplus g_4$ results in h_2 computing the same result as g, but at depth one lower, d. 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) and is only used for the outputs. 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     | T8 = U7 + T6     | T15 = T5 + T11   | T22 = T7 + T21
T2 = U0 + U5     | T9 = U7 + T7     | T16 = T5 + T12   | T23 = T2 + T22
T3 = U0 + U6     | T10 = T6 + T7    | T17 = T9 + T16   | T24 = T2 + T10
T4 = U3 + U5     | T11 = U1 + U5    | T18 = U3 + U7    | T25 = T20 + T17
T5 = U4 + U6     | T12 = U2 + U5    | T19 = T7 + T18   | T26 = T3 + T16
T6 = T1 + T5     | T13 = T3 + T4    | T20 = T1 + T19   | T27 = T1 + T12
T7 = U1 + U2     | T14 = T6 + T11   | T21 = U6 + U7    |
```

Fig. 5. Top linear transform in forward direction. Inputs are U0...U7.

```
T23 = U0 + U3    | T19 = T22 + R5   | T17 = U2 # T19   | T6 = T22 + R17
T22 = U1 # U3    | T9 = U7 # T1     | T20 = T24 + R13  | T16 = R13 + R19
T2 = U0 # U1     | T10 = T2 + T24   | T4 = U4 + T8     | T27 = T1 + R18
T1 = U3 + U4     | T13 = T2 + R5    | R17 = U2 # U5    | T15 = T10 + T27
T24 = U4 # U7    | T3 = T1 + R5     | R18 = U5 # U6    | T14 = T10 + R18
R5 = U6 + U7     | T25 = U2 # T1    | R19 = U2 # U4    | T26 = T3 + T16
T8 = U1 # T23    | R13 = U1 + U6    | Y5 = U0 + R17    |
```

Fig. 6. Top linear transform in reverse direction.

```
M1 = T13 x T6      | M17 = M5 + T24    | M33 = M27 + M25   | M49 = M43 x T16
M2 = T23 x T8      | M18 = M8 + M7     | M34 = M21 x M22   | M50 = M38 x T9
M3 = T14 + M1      | M19 = M10 + M15   | M35 = M24 x M34   | M51 = M37 x T17
M4 = T19 x D       | M20 = M16 + M13   | M36 = M24 + M25   | M52 = M42 x T15
M5 = M4 + M1       | M21 = M17 + M15   | M37 = M21 + M29   | M53 = M45 x T27
M6 = T3 x T16      | M22 = M18 + M13   | M38 = M32 + M33   | M54 = M41 x T10
M7 = T22 x T9      | M23 = M19 + T25   | M39 = M23 + M30   | M55 = M44 x T13
M8 = T26 + M6      | M24 = M22 + M23   | M40 = M35 + M36   | M56 = M40 x T23
M9 = T20 x T17     | M25 = M22 x M20   | M41 = M38 + M40   | M57 = M39 x T19
M10 = M9 + M6      | M26 = M21 + M25   | M42 = M37 + M39   | M58 = M43 x T3
M11 = T1 x T15     | M27 = M20 + M21   | M43 = M37 + M38   | M59 = M38 x T22
M12 = T4 x T27     | M28 = M23 + M25   | M44 = M39 + M40   | M60 = M37 x T20
M13 = M12 + M11    | M29 = M28 x M27   | M45 = M42 + M41   | M61 = M42 x T1
M14 = T2 x T10     | M30 = M26 x M24   | M46 = M44 x T6    | M62 = M45 x T4
M15 = M14 + M11    | M31 = M20 x M23   | M47 = M40 x T8    | M63 = M41 x T2
M16 = M3 + M2      | M32 = M27 x M31   | M48 = M39 x D     |
```

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

```
L0 = M61 + M62     | L10 = M53 + L4    | L20 = L0 + L1     | S0 = L6 + L24
L1 = M50 + M56     | L11 = M60 + L2    | L21 = L1 + L7     | S1 = L16 # L26
L2 = M46 + M48     | L12 = M48 + M51   | L22 = L3 + L12    | S2 = L19 # L28
L3 = M47 + M55     | L13 = M50 + L0    | L23 = L18 + L2    | S3 = L6 + L21
L4 = M54 + M58     | L14 = M52 + M61   | L24 = L15 + L9    | S4 = L20 + L22
L5 = M49 + M61     | L15 = M55 + L1    | L25 = L6 + L10    | S5 = L25 + L29
L6 = M62 + L5      | L16 = M56 + L0    | L26 = L7 + L9     | S6 = L13 # L27
L7 = M46 + L3      | L17 = M57 + L1    | L27 = L8 + L10    | S7 = L6 # L23
L8 = M51 + M59     | L18 = M58 + L8    | L28 = L11 + L14   |
L9 = M52 + M53     | L19 = M63 + L4    | L29 = L11 + L17   |
```

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

```
P0 = M52 + M61     | P10 = M57 + P4    | P20 = P4 + P6     | W1 = P26 + P29
P1 = M58 + M59     | P11 = P0 + P3     | P22 = P2 + P7     | W2 = P17 + P28
P2 = M54 + M62     | P12 = M46 + M48   | P23 = P7 + P8     | W3 = P12 + P22
P3 = M47 + M50     | P13 = M49 + M51   | P24 = P5 + P7     | W4 = P23 + P27
P4 = M48 + M56     | P14 = M49 + M62   | P25 = P6 + P10    | W5 = P19 + P24
P5 = M46 + M51     | P15 = M54 + M59   | P26 = P9 + P11    | W6 = P14 + P23
P6 = M49 + M60     | P16 = M57 + M61   | P27 = P10 + P18   | W7 = P9 + P16
P7 = P0 + P1       | P17 = M58 + P2    | P28 = P11 + P25   |
P8 = M50 + M53     | P18 = M63 + P5    | P29 = P15 + P20   |
P9 = M55 + M63     | P19 = P2 + P3     | W0 = P13 + P22    |
```

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

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 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 (see appendix). Since there are only 256 possible inputs, we verified the circuits fully against the specifications in [11].

7 Conclusion

The techniques used in [1, 5] to obtain smaller AND/XOR circuits were modified and extended here to consider the depth of the circuits produced. The resulting techniques were successfully applied to the AES S-Box, significantly improving, both in size and depth, over another recent attempt at a low depth S-Box [12].

The techniques presented in this paper appear, not surprisingly, to lead to a trade-off between size and depth. The modification of our search technique [1] to find circuits with few AND gates, which rejected candidates with too large depth, decreased the depth of the $GF(2^4)$ inversion from 9 to 4 while only increasing the number of gates from 16 to 17, changing 5 AND gates and 11 XOR gates to 7 AND gates and 10 XOR gates. Thus, most of the trade-off was due to the techniques for handling large linear components, especially the Low_Depth_Greedy algorithm. To illustrate this trade-off, we list the depths and sizes of some of the circuits for the AES S-Box which were obtained in the course of this research:

– depth 28; size 115: original circuit
– depth 23; size 116
– depth 22; size 117
– depth 21; size 118
– depth 20; size 122
– depth 16; size 128: final result

References

1. Boyar, J., Peralta, R.: A new combinational logic minimization technique with applications to cryptology. In: Festa, P. (ed.) SEA 2010. LNCS, vol. 6049, pp. 178–189. Springer (2010)
2. Boyar, J., Peralta, R., Pochuev, D.: On the multiplicative complexity of Boolean functions over the basis $(\wedge, \oplus, 1)$. Theoretical Computer Science 235, 43–57 (2000)
3. Canright, D.: A very compact Rijndael S-box. In: Rao, J., Sunar, B. (eds.) CHES 2005. LNCS, vol. 3659, pp. 441–455. Springer (2005)

4. Canright, D.: A very compact Rijndael S-box. Tech. Rep. NPS-MA-05-001, Naval Postgraduate School (2005)
5. Courtois, N., Hulme, D., Mourouzis, T.: Solving circuit optimisation problems in cryptography and cryptanalysis. IACR Cryptology ePrint Archive 2011, 475 (2011)
6. Fuhs, C., Schneider-Kamp, P.: Optimizing the AES S-Box using SAT. In: Proceedings of the 8th International Workshop on the Implementation of Logics (2010)
7. Fuhs, C., Schneider-Kamp, P.: Synthesizing shortest linear straight-line programs over GF(2) using SAT. In: Strichman, O., Szeider, S. (eds.) SAT '10. LNCS, vol. 6175, pp. 71–84. Springer (2010)
8. 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)
9. Lupanov, O.B.: A method of circuit synthesis. Izvestia V.U.Z. Radiofizika 1, 120–140 (1958)
10. Morioka, S., Satoh, A.: An optimized S-Box circuit architecture for low power AES design. In: Kaliski, B., Ç.K. Koç, Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 172–186. Springer (2003)
11. NIST: Advanced Encryption Standard (AES) (FIPS PUB 197). National Institute of Standards and Technology (Nov 2001)
12. Nogami, Y., Nekado, K., Toyota, T., Hongo, N., Morikawa, Y.: Mixed bases for efficient inversion in $\mathbb{F}(((2^2)^2)^2)$ and conversion matrices of subbytes of AES. In: Mangard, S., Standaert, F.X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 234–247. Springer (2010)
13. Paar, C.: Some remarks on efficient inversion in finite fields. In: 1995 IEEE International Symposium on Information Theory. p. 58 (1995)
14. Paar, C.: Optimized arithmetic for Reed-Solomon encoders. In: 1997 IEEE International Symposium on Information Theory. p. 250 (1997)
15. Satoh, A., Morioka, S., Takano, K., Munetoh, S.: A compact Rijndael hardware architecture with S-Box optimization. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 239–254. Springer (2001)
16. Shannon, C.E.: The synthesis of two-terminal switching circuits. Bell System Tech. J. 28, 59–98 (1949)

A Appendix

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

Fig. 10. Inversion in $GF(2^4)$ using NAND gates. Input is (x_0, x_1, x_2, x_3) and output is (y_0, y_1, y_2, y_3).