New logic minimization techniques with applications to cryptology.

Joan Boyar*
joan@imada.sdu.dk

René Peralta†
rene.peralta@nist.gov

Abstract

A new technique for combinational circuit optimization is described in the context of S-boxes. The technique is a two-step process. In the first step, the non-linearity of the circuit – as measured by the number of non-linear gates it contains – is reduced. The second step reduces the number of gates in the linear components of the already reduced circuit. The technique can be applied to arbitrary circuits, and seems to yield improvements even on circuits that have already been optimized by standard methods. We apply our technique to the S-box of the Advanced Encryption Standard (AES). The result is, as far as we know, the smallest circuit yet constructed for this function.

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

1 Introduction

Constructing an optimal circuit for an arbitrary function is an intractable problem under almost any meaningful metric (gate count, depth, energy consumption, etc.). In practice, no known techniques can necessarily find optimal circuits for functions with as few as eight Boolean inputs and one Boolean output (there are 2^{256} such functions). For example, the multiplicative circuit complexity of the Boolean function E_4^8, which is true if and only if exactly four of its eight input bits are true, is unknown [1]. In practice, we build circuit implementations of functions using a variety of heuristics. Many of these heuristics have exponential time complexity and thus can only be applied to small components of a circuit being built. This works reasonably well for functions that naturally decompose into repeated use of small components. Such functions include arithmetic functions (which we often build using full adders), matrix

*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.

multiplication (which decomposes into multiplication of small submatrices), and more complex functions such as cryptographic functions based on substitution-permutation networks.

This work presents a new technique for circuit optimization. The technique can be applied to arbitrary circuits, and seems to yield improvements even on circuits that have already been optimized by standard methods. We apply our technique to the S-box of AES, which, in addition to being used in AES, has been used in several proposals for a new hash function standard[1]. The result is, as far as we know[2], the smallest circuit yet constructed for this function. The circuit contains 32 AND gates and 83 XOR/XNOR gates for a total of 115 gates.

Our circuits are over the basis $\{\oplus, \wedge, 1\}$. This basis is logically complete: any Boolean circuit can be transformed into this form using only local replacements. The circuit operations can be viewed either as performing Boolean logic or arithmetic modulo 2. The number of $\wedge$ gates is called the *multiplicative complexity* of the circuit. Connected components of the circuit containing $\wedge$ gates are called *non-linear*. Components free of $\wedge$ gates are called *linear*.

2 Combinational circuit optimization

The techniques described here would generally be applied to subcircuits of a larger circuit, such as an S-box in a cryptographic application, which have relatively few inputs and outputs connecting them to the remainder of the circuit. The key observation that led us to our techniques is that circuits with low multiplicative complexity will naturally have large sections which are purely linear (i.e. contain only $\oplus$ gates). Thus

> *it is plausible that a two-step process, which first reduces multiplicative complexity and then optimizes linear components, leads to small circuits.*

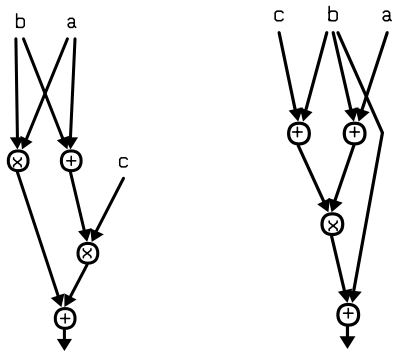
We have, of course, no way of proving this hypothesis. But the constructions in this paper support it.

First step

The first step of our technique consists of identifying non-linear components of the subcircuit to be optimized and reducing the number of $\wedge$ gates. This is not easy to do. For example, the two circuits below compute the same function. But it is not obvious how to algorithmically transform one into the other.

[1] See the first round candidates at:
http://csrc.nist.gov/groups/ST/hash/sha-3/Round1/submissions_rnd1.html

[2] We have been told that Intel has a new small implementation, but the details have not been made available.



Finding circuits with minimum multiplicative complexity is, in all likelihood, a highly intractable problem. However, recent work on multiplicative complexity contains an arsenal of reduction techniques that in practice yield circuits with small, and often optimal, multiplicative complexity [1]. That work focuses exclusively on symmetric functions (those whose value depends only on the Hamming weight of the input). In this paper, however, we use ad-hoc heuristics to construct a circuit with low multiplicative complexity for inversion in $GF(2^4)$. The technique is partially described in Section 3.

Second step

The second step of our technique consists of finding maximal linear components of the circuit and then minimizing the number of XOR gates needed to compute the target functions computed in these linear components. A new heuristic for this computationally intractable problem is described in Section 4.

3 AES's S-box

The non-linear operation in AES's S-box is to compute an inverse in the field $GF(2^8)$. A recursive method for building a circuit for inverses in $GF(2^{mn})$, given a circuit for inverses in $GF(2^m)$, is due to Itoh and Tsujii [4]. The circuits produced by this method are said to have a *tower fields architecture*. Since there are multiple possible representations for Galois fields, several authors have concentrated on finding representations that yield efficient circuits under the tower fields architecture. We use the same general technique for the reduction from inversion in $GF(2^8)$ to $GF(2^4)$ inversion, but we use a completely different technique for computing the inversion in $GF(2^4)$. We then place the optimized circuit for $GF(2^4)$ inversion in its appropriate place in AES's S-box and apply a novel optimization technique on the linear parts of the resulting circuit.

$GF(2^4)$ inversion – A non-linear component

The tower fields architecture for inversion in $GF(2^8)$ has (non-trivial) easily identifiable non-linear components corresponding to inversion in subfields. The first step in our method is to focus on one of these components and derive a circuit that uses few gates. The component for inversion in $GF(2^2)$ is too small for us to benefit significantly from optimizing it. Instead we focus on inversion in $GF(2^4)$.

Using the representation Canright [3] has concluded is best for his techniques, $GF(2^4)$ is represented using the normal basis (Z^2, Z^8) over $GF(2^2)$. An element $\delta \in GF(2^4)$ would be written as $\delta_1 Z^2 + \delta_2 Z^8$, where $\delta_1, \delta_2 \in GF(2^2)$. The four elements of $GF(2^2)$ are represented using the normal basis (W, W^2), so an element γ would be written as $\gamma_1 W + \gamma_2 W^2$, where $\gamma_1, \gamma_2 \in GF(2)$. Both Z^2 and Z^8 satisfy $z^2+z+W^2 = 0$, and W and W^2 satisfy $w^2+w+1 = 0$. From these two equations, one can calculate that $Z^4 = Z^2 + W$, $Z^8 = Z^2 + 1$, $Z^{10} = Z^4 + Z^2 = W$, $Z^{16} = Z^8 + W^2$, $W^3 = W^2 + W$, $W^4 = W$, and $W^5 = W^2$. These equations can be used to reduce expressions to check equalities.

Using this representation, an element of $GF(2^4)$ can be written as $\Delta = (x_1 W + x_2 W^2)Z^2 + (x_3 W + x_4 W^2)Z^8$, where $x_1, x_2, x_3, x_4 \in GF(2)$. The inverse of this element, $\Delta' = (y_1 W + y_2 W^2)Z^2 + (y_3 W + y_4 W^2)Z^8$, can then be calculated using 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 fact that Δ' is the inverse of Δ can be verified by multiplying the two elements together and reducing using the equations mentioned above (along with $x^2 = x$ and $x + x = 0$). The symbolic result is $(QW + QW^2)Z^2 + (QW + QW^2)Z^8$, where $Q = x_1x_2x_3x_4 + x_1x_2x_3 + x_1x_2x_4 + x_1x_3x_4 + x_2x_3x_4 + x_1x_2 + x_1x_3 + x_1x_4 + x_2x_3 + x_2x_4 + x_3x_4 + x_1 + x_2 + x_3 + x_4$. The fact that the value of Q is 1 unless all four variables have the value 0, when it is 0, can be seen by observing that it is the symmetric function $\Sigma_1^4 + \Sigma_2^4 + \Sigma_3^4 + \Sigma_4^4$. If exactly one variable is set, then the first term gives the value 1 (and the others 0); if exactly two are set, then only the second gives the value 1; if three are set, then the first, second and third terms give the value 1; and if all four are set, then only the last gives the value 1. Hence, the result is 1, except for the zero input.[3]

Thus the task at hand is to construct a circuit with four inputs and four outputs that calculates the above system of equations using as few $\wedge$ gates as possible. Currently, our heuristic search programs can handle functions with one output and up to eight inputs. This means that we can directly construct optimal circuits for each of

[3] A circuit for finite field inversion must have some output for the non-invertible zero element. In the following constructions we follow the AES convention that the output on input zero is zero.

$$
\begin{array}{llll}
t_1 = x_1 + x_2 & t_2 = x_1 \times x_3 & t_3 = x_4 + t_2 & \\
t_4 = t_1 \times t_3 & y_4 = x_2 + t_4 \quad (*) & t_5 = x_3 + x_4 & \\
t_6 = x_2 + t_2 & t_7 = t_6 \times t_5 & y_2 = x_4 + t_7 & (*) \\
t_8 = x_3 + y_2 & t_9 = t_3 + y_2 & t_{10} = x_4 \times t_9 & \\
y_1 = t_{10} + t_8 \quad (*) & t_{11} = t_3 + t_{10} & t_{12} = y_4 \times t_{11} & \\
y_3 = t_{12} + t_1 \quad (*) & & &
\end{array}
$$

Figure 1: Inversion in $GF(2^4)$.

the four equations individually, but not for the system itself. For the full system we took the following approach:

- pick an equation and construct an efficient circuit for it;[4]
- store intermediate functions computed in the previous steps for possible use in constructing a circuit for the next equation to be tackled;
- iterate until all equations have been computed.

We did this for each of the 24 orderings of $\{y_1, y_2, y_3, y_4\}$. The ordering (y_4, y_2, y_1, y_3) gave the best results. The resulting circuit, expressed as a straight-line program over GF(2), is shown in Figure 1 (outputs are indicated by an (*)).

This circuit contains 5 $\wedge$ gates and 11 $\oplus$ gates. It is a significant improvement over previous constructions, e.g. Paar's construction [5] has a gate count of 10 $\wedge$ gates and 15 $\oplus$ gates for the same function. It is harder to compare to Canright's construction [3]. In his original, he had 9 $\wedge$ gates (and NAND gates) and 14 $\oplus$ gates (and XNOR gates), but he optimized, allowing NOR gates. After this, he had 8 NAND gates, 2 NOR gates, and 9 XOR/XNOR gates.

The multiplicative complexity of a function is the number of GF(2) multiplications necessary and sufficient to compute it. Under the given representation for $GF(2^4)$, the multiplicative complexity of inversion is 5. This can be argued as follows: the upper bound is clearly given by the construction. The four outputs that have to be computed all have degree 3. One $\wedge$ is needed to compute a polynomial of degree 2. Then, an additional $\wedge$ is necessary to produce each of the four linearly independent polynomials, since each is of degree 3.

A view of the structure of AES's S-box

In the previous section, using the tower fields architecture, we identified and optimized (with respect to multiplicative complexity) a major non-linear component in an implementation of the AES S-box. That completes the first step of our technique

[4]This step is non-trivial for arbitrary functions. The heuristic we used is inspired by methods from automatic theorem proving. We omit its description here due to space constraints.

for circuit optimization, but in other circuits, one may be able to identify more non-linear components with few enough inputs that they can also be optimized before continuing. In the case of AES, after the non-linear portions of the circuit are optimized, as expected, a bird's-eye view of the resulting circuit reveals large linear connected components. In fact, from a cryptanalyst's point of view, the topology of the resulting circuit is potentially of interest: the S-box of AES consists of an initial linear expansion U from 8 to 22 bits, followed by a non-linear contraction F from 22 to 18 bits, and ending with a linear contraction B from 18 to 8 bits. The U and B matrices are given below. AES's S-box is $S(\mathbf{x}) = B \cdot F(U \cdot \mathbf{x}) + [11000110]^T$, where $\cdot$ is matrix multiplication and x is the 8-bit S-box input. We do not know if there are any cryptanalytic implications to the structure of these matrices. The first row and last columns of U should raise and eyebrow, as should the 12^{th} and last three columns of B. Note that the initial linear expansion and the linear contraction were defined to contain as much of the circuit as possible while still being linear. Thus, the portion of the circuit defined by U, for example, overlaps with the $GF(2^8)$ inversion. The next step was to minimize the circuits for computing U and B.

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

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

4 Minimizing linear components

Gate optimization of circuits for linear functions has been extensively studied. It has been shown that the problem of linear-circuit optimization is NP-hard [2]. That paper further shows that unless P=NP, this problem does not even have efficient ϵ-approximation schemes. Thus, our goal in this research is restricted to improving on known heuristics. As far as we know, the most successful heuristics are variations on a greedy algorithm due to Paar [6].

A linear straight-line program over a field F is a variation on a straight-line program which does not allow multiplication of variables. That is, every line of the program is of the form $u := \lambda v + \mu w$ where λ, μ are in F and v, w are variables. Constructing a linear circuit for a given function f is equivalent to constructing a linear straight-line program over GF(2) which computes f. (Note that over GF(2) λ and μ are always one and thus are never written explicitly.)

A linear straight-line program over GF(2) is said to be *cancellation-free* if, for every line of the program $u := v + w$, none of the variables in the expression for v are also present in the expression for w, i.e., there is no cancellation of variables in the computation.

Previous work on circuit minimization for AES S-boxes (e.g. [5, 7, 3]) only consider cancellation-free straight-line programs for producing a set of linear forms over GF(2). Some authors appear to make the incorrect assumption that there always exists a cancellation-free optimal linear program over GF(2). A small counter-example showing this is not the case over GF(2) is the following:

$$x_1 + x_2; \quad x_1 + x_2 + x_3; \quad x_1 + x_2 + x_3 + x_4; \quad x_2 + x_3 + x_4.$$

It is not hard to see that the optimum cancellation-free straight-line program has length 5. A solution of length 4 which allows cancellations is

$$v_1 = x_1 + x_2 \quad v_2 = v_1 + x_3 \quad v_3 = v_2 + x_4 \quad v_4 = v_3 + x_1.$$

More generally, it was shown in [2] that any algorithm for computing linear programs, which only produces cancellation-free programs, is at most $\frac{3}{2}$-approximating. Thus, even optimal cancellation-free circuits can be far from optimal in the unrestricted model. The heuristic we present below is not restricted to producing cancellation-free circuits.

A new heuristic

Let S be a set of linear functions. For any linear predicate f, we define the distance $\delta(S, f)$ as the minimum number of additions of elements from S necessary to obtain f.

The problem is to find a short linear program that computes $f(\mathbf{x}) = M\mathbf{x}$ where M is an $m \times n$ matrix over GF(2). The heuristic is as follows. We keep a "base" S of "known" functions. Initially S is just the set of variables $x_1, \ldots, x_n$. We maintain the vector $Dist[]$ of distances from S to the functions given by the rows of M. That is, $Dist[i] = \delta(S, f_i)$ where f_i is the i^{th} row of M multiplied by the input vector $\mathbf{x}$. Initially, $Dist[i]$ is just one less than the Hamming weight of row i. We then perform the following loop

- pick a new basis element by adding two existing basis elements;
- update $Dist[]$;

until $Dist[i] = 0$ for all i.

The current criterion for picking the new basis element is

- pick one that minimizes the sum of new distances;
- resolve ties by maximizing the Euclidean norm of the vector of new distances.

The tie resolution criterion may seem counter-intuitive. The basic idea is that we prefer a distance vector like 0,0,3,1 to one like 1,1,1,1. In the latter case, we would need 4 more gates to finish. In the former, 3 might do it.

The bulk of the time of the heuristic is spent on picking the new basis element. Our experiments show that the following "pre-emptive" choice usually improves running time without increasing the size of the output circuit:

- if any two bases $S[i], S[j]$ are such that $S[i] \oplus S[j]$ is a row in M then pick this sum as the new basis element.

The tie resolution criterion is a critical part of the heuristic. It does well on most matrices we have tried, but we have found specific matrices for which other decision rules do better. Intuitively, no one simple rule should work for all matrices. The effectiveness of the heuristic most likely depends on the topology of the digraph represented by the input matrix. We have not pursued this line of inquiry. We have, however tested our heuristic against standard methods. On random matrices, our heuristic gives significant improvements (see Appendix A).

The tie resolution step is also a good randomization point. If this step fails to resolve a tie between two choices, we can simply flip a coin. We can perform several runs of the algorithm and pick the best solution found.

The distance vector is computed by exhaustive search. The reason the heuristic is practical for moderate-size matrices is that the distance can only decrease. In fact, it can only decrease by 1. So when a new base is being considered, if a distance is d, then only combinations of exactly $d-1$ old basis elements and the new basis element need to be considered.

A small example using the heuristic

Suppose we need a circuit that computes the following system of equations.

$$\begin{aligned} y_0 &= x_0 + x_1 + x_2 \\ y_1 &= x_1 + x_3 + x_4 \\ y_2 &= x_0 + x_2 + x_3 + x_4 \\ y_3 &= x_1 + x_2 + x_3 \\ y_4 &= x_0 + x_1 + x_3 \\ y_5 &= x_1 + x_2 + x_3 + x_4 \end{aligned}$$

Equivalently, we need a circuit for multiplication by the following 6×5 matrix

$$M = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

The *target signals* to be computed are simply the rows of M. The initial base is $\{x_0, x_1, x_2, x_3, x_4\}$, which corresponds to

$$S = \{\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \end{bmatrix},$$

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \end{bmatrix}\}$$

The initial distance vector is

$$D = \begin{bmatrix} 2 & 2 & 3 & 2 & 2 & 3 \end{bmatrix}$$

The heuristic must find two basis vectors whose sum, when added to the basis, minimizes the sum of the new distances. It turns out the right choice is to calculate $x_1 + x_3$. So the new basis S is expanded to contain the signal

$$\begin{bmatrix} 0 & 1 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

The new distance vector is

$$D = \begin{bmatrix} 2 & 1 & 3 & 1 & 1 & 2 \end{bmatrix}$$

The full run of the program is below. The tie breaking criteria is used in Step 2. If one had chosen $x_1 + x_2$ instead of $x_0 + t_5$, the new distance vector would be $[\ 1\ \ 1\ \ 3\ \ 1\ \ 1\ \ 2\ ]$, which has norm 17, while the one found has norm 19. Note that there is cancellation in the last step.

Step 1 : $t_5 = x_1 + x_3$. New D : [2 1 3 1 1 2].

Step 2 : $t_6 = x_0 + t_5$ (found target signal $y_4 = [1\ 1\ 0\ 1\ 0]$). New D : [2 1 3 1 0 2].
Step 3 : $t_7 = x_2 + t_5$ (found target signal $y_3 = [0\ 1\ 1\ 1\ 0]$). New D : [2 1 3 0 0 1].
Step 4 : $t_8 = x_4 + t_5$ (found target signal $y_1 = [0\ 1\ 0\ 1\ 1]$). New D : [2 0 3 0 0 1].
Step 5 : $t_9 = x_2 + t_8$ (found target signal $y_5 = [0\ 1\ 1\ 1\ 1]$). New D : [2 0 2 0 0 0].
Step 6 : $t_{10} = x_0 + x_1$. New D : [1 0 1 0 0 0].
Step 7 : $t_{11} = x_2 + t_{10}$ (found target signal $y_0 = [1\ 1\ 1\ 0\ 0]$) . New D : [0 0 1 0 0 0].
Step 8 : $t_{12} = t_8 + t_{11}$ (found target signal $y_2 = [1\ 0\ 1\ 1\ 1]$). New D : [0 0 0 0 0 0].
(DONE!)

As mentioned earlier, when choosing a base pair to create a new base, the heuristics we used computed for each possible pair, the sum of the new distances to the targets and chose the pair giving the minimum. When there was more than one, a tie-breaking rule was used. We have also tried other tie-breaking rules, but rules which involve trying to maximize the Euclidean norm appear to do best. In Appendix A we present some of our results comparing tie-breakers based partially on maximizing the Euclidean norm. One of these tie-breakers also involved minimizing the largest distance, since this largest distance is a lower bound on the number of extra gates still needed. Another tie-breaker involved minimizing the difference between the two largest distances, since if one distance is much larger than another, there is a limit to how many gates can be shared in computing those two targets. In both cases, we used as the tie-breaking rule, to maximize the square of the Euclidean norm minus the value which we are trying to minimize. There are clearly other possibilities. Among these is using that Euclidean norm, instead of its square in the objective functions where the largest distance, for example, is subtracted. This appears to give poorer results.

5 A circuit for the S-box of AES

Our techniques yield a circuit for the AES S-box composed of three parts: a "top" linear transformation; a middle non-linear part; and a "bottom" linear transformation. The linear transformations are defined by the matrices U and B of section 3.

For matrix U, the smallest circuits we found had 23 $\oplus$ gates. Among the many such circuits, the shortest ones have depth 7. It is worthwhile to note that if 24 $\oplus$ gates are allowed, circuits with depth 4 exist for U. Figure 2 shows a circuit of size 23 and depth 7. The circuit maps inputs $x_0 \ldots x_7$ to outputs $x_7, y_1 \ldots y_{21}$.

Figure 3 shows the non-linear middle part of the S-box circuit. It is a function from 22 to 18 bits. The circuit contains 32 $\wedge$ gates and 32 $\oplus$ gates. It maps inputs $x_7, y_1 \ldots y_{21}$ to outputs $z_0 \ldots z_{17}$.

For matrix B, the randomized version of our heuristic yields many circuits with 30 $\oplus$ gates. The heuristic is fast enough that we are able to pick a circuit which is both small and short. Figure 4 shows a circuit of depth 6. The circuit maps inputs $z_0 \ldots z_{17}$ to outputs $s_0 \ldots s_7$.

$$
\begin{array}{lcllcllcl}
y_{14} & = & x_3 + x_5 & y_{13} & = & x_0 + x_6 & y_9 & = & x_0 + x_3 \\
y_8 & = & x_0 + x_5 & t_0 & = & x_1 + x_2 & y_1 & = & t_0 + x_7 \\
y_4 & = & y_1 + x_3 & y_{12} & = & y_{13} + y_{14} & y_2 & = & y_1 + x_0 \\
y_5 & = & y_1 + x_6 & y_3 & = & y_5 + y_8 & t_1 & = & x_4 + y_{12} \\
y_{15} & = & t_1 + x_5 & y_{20} & = & t_1 + x_1 & y_6 & = & y_{15} + x_7 \\
y_{10} & = & y_{15} + t_0 & y_{11} & = & y_{20} + y_9 & y_7 & = & x_7 + y_{11} \\
y_{17} & = & y_{10} + y_{11} & y_{19} & = & y_{10} + y_8 & y_{16} & = & t_0 + y_{11} \\
y_{21} & = & y_{13} + y_{16} & y_{18} & = & x_0 + y_{16} & & &
\end{array}
$$

Figure 2: Top linear transformation: Inputs are $x_0, x_1, ..., x_7$. Outputs to the next level are $x_7, y_1, y_2, ..., y_{21}$.

$$
\begin{array}{lcllcllcl}
t_2 & = & y_{12} \times y_{15} & t_3 & = & y_3 \times y_6 & t_4 & = & t_3 + t_2 \\
t_5 & = & y_4 \times x_7 & t_6 & = & t_5 + t_2 & t_7 & = & y_{13} \times y_{16} \\
t_8 & = & y_5 \times y_1 & t_9 & = & t_8 + t_7 & t_{10} & = & y_2 \times y_7 \\
t_{11} & = & t_{10} + t_7 & t_{12} & = & y_9 \times y_{11} & t_{13} & = & y_{14} \times y_{17} \\
t_{14} & = & t_{13} + t_{12} & t_{15} & = & y_8 \times y_{10} & t_{16} & = & t_{15} + t_{12} \\
t_{17} & = & t_4 + t_{14} & t_{18} & = & t_6 + t_{16} & t_{19} & = & t_9 + t_{14} \\
t_{20} & = & t_{11} + t_{16} & t_{21} & = & t_{17} + y_{20} & t_{22} & = & t_{18} + y_{19} \\
t_{23} & = & t_{19} + y_{21} & t_{24} & = & t_{20} + y_{18} & & & \\
 & & & & & & & & \\
t_{25} & = & t_{21} + t_{22} & t_{26} & = & t_{21} \times t_{23} & t_{27} & = & t_{24} + t_{26} \\
t_{28} & = & t_{25} \times t_{27} & t_{29} & = & t_{28} + t_{22} & t_{30} & = & t_{23} + t_{24} \\
t_{31} & = & t_{22} + t_{26} & t_{32} & = & t_{31} \times t_{30} & t_{33} & = & t_{32} + t_{24} \\
t_{34} & = & t_{23} + t_{33} & t_{35} & = & t_{27} + t_{33} & t_{36} & = & t_{24} \times t_{35} \\
t_{37} & = & t_{36} + t_{34} & t_{38} & = & t_{27} + t_{36} & t_{39} & = & t_{29} \times t_{38} \\
t_{40} & = & t_{25} + t_{39} & & & & & & \\
 & & & & & & & & \\
t_{41} & = & t_{40} + t_{37} & t_{42} & = & t_{29} + t_{33} & t_{43} & = & t_{29} + t_{40} \\
t_{44} & = & t_{33} + t_{37} & t_{45} & = & t_{42} + t_{41} & z_0 & = & t_{44} \times y_{15} \\
z_1 & = & t_{37} \times y_6 & z_2 & = & t_{33} \times x_7 & z_3 & = & t_{43} \times y_{16} \\
z_4 & = & t_{40} \times y_1 & z_5 & = & t_{29} \times y_7 & z_6 & = & t_{42} \times y_{11} \\
z_7 & = & t_{45} \times y_{17} & z_8 & = & t_{41} \times y_{10} & z_9 & = & t_{44} \times y_{12} \\
z_{10} & = & t_{37} \times y_3 & z_{11} & = & t_{33} \times y_4 & z_{12} & = & t_{43} \times y_{13} \\
z_{13} & = & t_{40} \times y_5 & z_{14} & = & t_{29} \times y_2 & z_{15} & = & t_{42} \times y_9 \\
z_{16} & = & t_{45} \times y_{14} & z_{17} & = & t_{41} \times y_8 & & &
\end{array}
$$

Figure 3: The middle non-linear section: Inputs are $x_7, y_1, y_2, ..., y_{21}$. Outputs to the next level are $z_0, z_1, ..., z_{17}$. Note that the computation of t_{25} through t_{40} is the inversion in $GF(2^4)$.

t_{46}	=	$z_{15} + z_{16}$	t_{47}	=	$z_{10} + z_{11}$	t_{48}	=	$z_5 + z_{13}$
t_{49}	=	$z_9 + z_{10}$	t_{50}	=	$z_2 + z_{12}$	t_{51}	=	$z_2 + z_5$
t_{52}	=	$z_7 + z_8$	t_{53}	=	$z_0 + z_3$	t_{54}	=	$z_6 + z_7$
t_{55}	=	$z_{16} + z_{17}$	t_{56}	=	$z_{12} + t_{48}$	t_{57}	=	$t_{50} + t_{53}$
t_{58}	=	$z_4 + t_{46}$	t_{59}	=	$z_3 + t_{54}$	t_{60}	=	$t_{46} + t_{57}$
t_{61}	=	$z_{14} + t_{57}$	t_{62}	=	$t_{52} + t_{58}$	t_{63}	=	$t_{49} + t_{58}$
t_{64}	=	$z_4 + t_{59}$	t_{65}	=	$t_{61} + t_{62}$	t_{66}	=	$z_1 + t_{63}$
s_0	=	$t_{59} + t_{63}$	s_6	=	t_{56} XNOR t_{62}	s_7	=	t_{48} XNOR t_{60}
t_67	=	$t_{64} + t_{65}$	s_3	=	$t_{53} + t_{66}$	s_4	=	$t_{51} + t_{66}$
s_5	=	$t_{47} + t_{65}$	s_1	=	t_{64} XNOR s_3	s_2	=	t_{55} XNOR t_{67}

Figure 4: Bottom linear transformation: Inputs are $z_0, z_1, ..., z_{17}$. Outputs are $s_0, s_1, ..., s_7$.

References

[1] J. Boyar and R. Peralta. Tight bounds for the multiplicative complexity of symmetric functions. *Theoretical Computer Science*, 396(1-3):223–246, 2008.

[2] Joan Boyar, Philip Matthews, and René Peralta. On the shortest linear straight-line program for computing linear forms. In *MFCS*, pages 168–179, 2008.

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

[4] 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.

[5] C. Paar. Some remarks on efficient inversion in finite fields, 1995. In 1995 IEEE International Symposium on Information Theory, page 58, Whistler, B.C. Canada.

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

[7] A. Satoh, S. Morioka, K. Takano, and S. Munetoh. A compact Rijndael hardware architecture with S-Box optimization. In *Advances in Cryptology - Proceedings of ASIACRYPT 01*, volume 2248 of *Lecture Notes in Computer Science*, pages 239–254. Springer-Verlag, 2001.

Appendix A: Experimental results for the linear circuit optimization heuristic

In order to compare the effects of using different tie-breakers, we created random matrices defining which sets of linear forms should be computed by the circuits produced. Each row of the matrix corresponds to a linear form in the natural way. We generated random matrices as follows

- we first chose a size (for example, 10×20 matrices, which represent 10 linear forms on 20 distinct variables);
- we then picked a *bias* ρ between 0 and 1;
- for each entry of the matrix, we set the bit to 1 with probability ρ and to 0 with probability $1 - \rho$. Thus ρ is the expected fraction of variables that appears in each linear form. The "srand()" pseudorandom number generator in C++ (using the g++ compiler) was used to choose bits with the given probability.
- Matrices with rows which are all zeros (representing the form 0) were discarded, as were duplicate rows (producing the same form twice).

We compared the different heuristics on one hundred 15×15 matrices, with each of the biases 1/4, 1/2, 3/4, and 9/10, on ten 20×20 matrices with bias 3/4, on twenty 20×10 matrices with bias 3/4, on twenty 10×20 matrices with bias 3/4. The experience with 15×15 matrices showed that the heuristics were all slower when the bias was larger. This was expected, since the initial "distances" (number of operations on the basis vectors to obtain the target vectors) were then larger on average when there were more ones in the matrices. Note that the heuristics were slower for bias 9/10 than for bias 3/4, even though fewer gates were used in the resulting circuits. For the first three heuristics, all one hundred 15×15 matrices were processed within a few minutes for small biases and within half an hour for the larger biases. In contrast, the ten 20×20 matrices took several hours, though less than a day to complete. The running times on 20×10 matrices was closer to that of the 15×15 matrices, while the running times on 10×20 matrices was closer to that of the 20×20 matrices, despite the fact that the total number of ones in these matrices was similar. This again reflects the fact that the average (or possibly maximum) initial "distances" determine the running time.

The tie-breakers we compared were the following:

- **Norm:** maximizing the Euclidean norm
- **Norm-largest:** maximizing the square of the Euclidean norm minus the largest distance
- **Norm-diff:** maximizing the square of the Euclidean norm minus the difference of the largest two distances

- **Random:** If there is a tie with respect to sum of distances then flip an unbiased coin. If heads then keep current choice. If tails then apply the Euclidean norm criterion for tie-breaking. This heuristic may end up choosing a pair with non-maximum Euclidean norm. On the other hand, it allows substitution of one optimum (by sum-of-distances and Euclidean norm) pair by another found later in the search.

In all cases, except the "Random" one, when there were still ties after applying the "tie-breaker", the first pair, with both the minimum sum of distances and the optimal value for the tie-breaker, was chosen. This was the basis pair with lexicographical minimum indices (i, j). The exception to this is when there is a target with distance 1, meaning that using one extra gate will produce a target. Since it can never be wrong to use such a gate, a check is made for this case, by scanning the distances and choosing the first with distance 1 when such exists. This check is efficient, and when there is a target of distance one, it saves lengthy computations of new distances for each possible pair of bases. Note that this optimization gives a different ordering from what is shown in the example in Section 4 (though one still gets a circuit with eight gates).

Randomized tie-breaking allows running the heuristic several times and picking the best result. In our tests we ran the heuristic with "Random" tie-breaking three times.

We also compared these heuristics to Paar's heuristic [6] on the same matrices. Paar's heuristic is significantly faster than our heuristic, but it produces only cancellation-free circuits. Its performance, relative to the heuristics proposed here, decreases as the bias increases, using more than 30% extra gates when the bias is 3/4 and 40% extra when the bias is 9/10.

Among the biases tried, the number of gates in the circuits found by our heuristics is largest with bias 3/4. It is not a strictly increasing function of the bias, since when nearly all of the variables are used in nearly all of the forms, the outputs from many of the gates can be reused for many targets.

All of the heuristics do fairly similarly, with Random apparently doing slightly better, presumably because it tries three different circuits and uses the best. It also runs for about three times as long as the others.

In the table below, the column headings specify the matrix size, the bias, and the number of matrices of that size and bias tested.

For each tie-breaker rule and Paar's heuristic, for each matrix size and bias, the total number of gates used (the sum over all the matrices having those characteristics) and the number of matrices where that heuristic did not obtain the minimum value of all of the heuristics is given. Note that this means the Paar heuristic was beaten by at least one of the other heuristics on all matrices except for 16 of the 100 with bias 1/4. In fact, for the tests with bias larger than 1/4, Paar's heuristic did worse than any of the other heuristic on every one of the matrices; usually the values obtained for the newer heuristics were similar, with Random possibly being marginally better, but with the value for Paar's heuristic being significantly larger.

The last row shows the sums of the values which are the minimum of those calculated by the different heuristics for each matrix. This shows that for each of the tie-breakers, there are cases where it gets a better result than the others.

	15×15 (100) Bias=$\frac{1}{4}$		15×15 (100) Bias=$\frac{1}{2}$	
Heuristic	Total	Not min	Total	Not min
Norm	2965	16	4421	48
Norm-largest	2963	14	4423	49
Norm-diff	2965	15	4428	51
Random	2965	10	4428	23
Paar	3107	83	5170	100
Minimum	2948	0	4350	0

	15×15 (100) Bias=$\frac{3}{4}$		15×15 (100) Bias=$\frac{9}{10}$	
Heuristic	Total	Not min	Total	Not min
Norm	4082	47	3028	31
Norm-largest	4082	46	3028	31
Norm-diff	4082	46	3029	32
Random	4082	23	3029	14
Paar	5327	100	4311	100
Minimum	4011	0	2986	0

	20×20 (10) Bias=$\frac{3}{4}$		20×10 (20) Bias=$\frac{3}{4}$		10×20 (20) Bias=$\frac{3}{4}$	
Heuristic	Total	Not min	Total	Not min	Total	Not min
Norm	684	7	618	6	839	6
Norm-largest	684	7	617	5	848	10
Norm-diff	684	7	617	5	850	10
Random	676	3	612	1	840	5
Paar	926	10	864	20	995	20
Minimum	673	0	611	0	832	0