A Very Compact "Perfectly Masked" S-Box for AES (corrected)

D. Canright[1] and Lejla Batina[2]

[1] Applied Math., Naval Postgraduate School, Monterey CA 93943, USA,
dcanright@nps.edu
[2] K.U. Leuven ESAT/COSIC, Kasteelpark Arenberg 10,
B-3001 Leuven-Heverlee, Belgium

Abstract. Implementations of the Advanced Encryption Standard (AES), including hardware applications with limited resources (e.g., smart cards), may be vulnerable to "side-channel attacks" such as differential power analysis. One countermeasure against such attacks is adding a random mask to the data; this randomizes the statistics of the calculation at the cost of computing "mask corrections." The single nonlinear step in each AES round is the "S-box" (involving a Galois inversion), which incurs the majority of the cost for mask corrections. Oswald et al.[1] showed how the "tower field" representation allows maintaining an additive mask throughout the Galois inverse calculation. This work applies a similar masking strategy to the most compact (unmasked) S-box to date[2]. The result is the most compact masked S-box so far, with "perfect masking" (by the definition of Blömer[3]) giving suitable implementations immunity to first-order differential side-channel attacks.

keywords: AES, S-box, masking, DPA, composite Galois field

1 Introduction

In 2001 the National Institute of Standards and Technology adopted the Rijndael algorithm [4] as the Advanced Encryption Standard (AES)[5], to provide a standard algorithm for secure encryption, intended not only for U.S. government documents, but also for electronic commerce. Since then, applications of AES have become widespread.

Many different implementations of AES have appeared, to satisfy the varying criteria of different applications. Some approaches seek to maximize throughput, e.g., [6–9]; others minimize power consumption, e.g., [10]; and yet others minimize circuitry, e.g., [11–15]. For the latter goal, Rijmen[16] suggested using subfield arithmetic in the crucial step of computing an inverse in the Galois Field of 256 elements. This idea was further extended by Satoh et al.[12], using sub-subfields (the "tower field" representation of Paar[17], also called the "composite-field" approach), along with other innovative optimizations, which

revised November 28, 2008

resulted in the smallest AES circuit at that point. The S-box architecture of Satoh was refined somewhat by Canright[2], mainly through carefully chosen normal bases, resulting in the most compact S-box to date.

No attacks have yet been found on the AES algorithm itself that are more effective than exhaustive key search ("brute force"), although research continues, for example, on algebraic attacks[18, 19]. But specific implementations of cryptograpy in software or hardware, e.g. in smart cards, may be vulnerable to "side-channel attacks" such as differential power analysis[20, 21], that use statistical analysis of physical quantities such as power consumption, electromagnetic radiation, etc., to deduce information about the secret key.

One countermeasure against side-channel attacks is masking the data during calculation, through adding or multiplying by random values. All the steps in a round of AES are affine, except for the Galois field inversion substep of the S-box (*SubBytes*) step. For the other steps, calculation of the mask correction is linear, so an additive mask is most convenient. Some have suggested switching to a multiplicative mask for the Galois inverse step (e.g., [22]), but one inescapable weakness is that a zero data byte is unmasked by multiplication [23].

Applying the "tower field" representation, inversion in $GF(2^8)$ involves several multiplications and one inversion in the subfield $GF(2^4)$, which in turn involves multiplications and inversion in $GF(2^2)$. In the sub-subfield $GF(2^2)$, inversion is identical to squaring, and so is *linear* (over $GF(2)$). Oswald *et al.*[1] applied this idea to additive masking of the Galois inverse, and showed how to compute the mask corrections for the tower field approach. (Morioka and Akishita[24] apparently developed a similar masking scheme.) Many of the correction terms involve multiplication in subfields, and Oswald *et al.* showed how some of these multiplications can be eliminated through clever re-use of parts of the input mask for the output.

The present work incorporates this masking approach into the compact S-box of Canright[2], and also applies the optimization methods of [2] to the mask correction terms. At the level of operations in the subfield $GF(2^4)$, we simplify the approach somewhat, eliminating one multiplication and some additions, with further simplifications at lower levels. Even so, we show that our approach achieves the goal of "perfect masking," in the terminology of Blömer *et al.*[3]: each intermediate result has a statistical distribution that is independent of the plaintext and key (assuming a source of uniformly distributed truly random masks). This level of security (a strengthened version of that in [25]) ensures that no first-order differential side-channel attacks can succeed, at least at the algorithmic level. (Higher-order attacks are possible, but take much greater effort.)

However, Mangard *et al.*[26] showed that first-order DPA attacks can succeed against a masked S-box using standard CMOS technology, even when the intermediate results at the algorithmic level are provably secure. The attack exploits glitches in the gate transition timings. In later work, Mangard *et al.*[27] showed that the information leakage is caused by specific XOR gates in masked multipliers, and can be eliminated if those XORs are made to satisfy timing constraints, either through delay elements or by enable signals. Also, rather than CMOS,

other (more expensive) logic styles can be used to eliminate this potential problem. So masked S-boxes can be made secure from first-order DPA. At any rate, the current work only considers the intermediate results at the algorithmic level, and shows that they are provably secure, which is the best that can be expected.

We apply the same optimizations as in the unmasked S-box of [2], including efficient normal bases, elimination of common sub-expressions, and logic-gate substitutions, to the masked S-box calculation here, including mask correction terms. The result is, as far as we know, the most compact *masked* S-box to date (for the 0.13-μ CMOS standard cell library considered). But the cost for this security is that the S-box size is almost three times that of the unmasked S-box. Also, since the security requires certain calculations to proceed in sequence, the speed will be reduced. For applications with sufficient resources to unroll the round loop, we show how re-using masks *between rounds* can save a significant amount of the mask correction calculations; then the masked S-box is roughly twice the size of the unmasked. Our compact masked S-box design could be useful for securing some hardware AES applications, especially those with limited resources, against first-order differential attacks.

2 Algebraic description

The AES algorithm has been described thoroughly and frequently elsewhere[5]; here we give the barest outline before concentrating on the S-box. It is a symmetric block cipher (16 bytes, though the original Rijndael cipher supports other block sizes[4]) consisting of several rounds (10, 12, or 14, for a key size 16, 24, or 32 bytes, respectively). Each round involves the four steps called *SubBytes* (byte substitution, or S-box), *ShiftRows*, *MixColumns*, and *AddRoundKey* (the last round skips *MixColumns*, and there is a Round 0 consisting solely of *AddRoundKey*). The latter three steps are linear with respect to the data block, and provide "diffusion." *SubBytes* is the nonlinear step that provides "confusion."

The S-box, applied to each byte, consists of two substeps: (i) considering the byte an element of the Galois field $GF(2^8)$, find its inverse in that field (except a zero byte, which has no inverse, remains unchanged); (ii) considering the resulting byte a vector of bits in $(GF(2))^8$, multiply by a given bit matrix and add a given constant vector, i.e., an affine transformation.

In the particular Galois field of AES, a byte represents a polynomial where the bits are coefficients of corresponding powers of x, and multiplication is modulo the irreducible polynomial $q(x) = x^8 + x^4 + x^3 + x + 1$. Equivalently, one could consider a root, say θ, of this polynomial, so $q(\theta) = 0$ in this field; then the bits of a byte would correspond to coefficients of powers of θ, e.g., $2 = \theta, 3 = \theta + 1, 4 = \theta^2$, etc. Thus the bits form a vector with respect to what is called a polynomial basis. But there are computational advantages to considering a different (though isomorphic) representation of $GF(2^8)$. Instead of a vector of dimension eight over $GF(2)$, we consider a byte as a vector of dimension two over $GF(2^4)$, where each 4-bit element is in turn a vector of dimension two over $GF(2^2)$, and finally each 2-bit element is a vector of dimension two over $GF(2)$. This has been called a

composite field, or "tower field" representation[17]. In this way, the 8-bit inverse calculation comprises several 4-bit operations, each consisting of various simple 2-bit calculations. For each of these subfields, it has been shown[2] that a normal basis (consisting of a conjugate pair) is more efficient than a polynomial basis for the required inverse calculation; for each particular basis used, the trace (sum of the conjugate pair) is 1, and the norm (product of the conjugate pair) is a nonzero element of the subfield[2].

Converting between the standard AES representation and the composite field representation amounts to a change of basis, accomplished by multiplying the bit vector by a bit matrix. In converting back, this bit matrix can be combined with that of the affine transformation substep[12]. With regard to an additive mask, these matrix multiplications are simple linear calculations for the mask correction terms. Below we detail the mask corrections required for the nonlinear inverse calculation.

2.1 Inversion without masking

Here we employ the following convention: upper-case bold symbols represent elements of the main field (e.g. $\mathbf{A} \in GF(2^8)$); upper-case italic symbols are for elements of the subfield (e.g. $A \in GF(2^4)$); lower-case bold is used for the sub-subfield (e.g. $\mathbf{a} \in GF(2^2)$); and lower-case italic is for single bits (e.g. $a \in GF(2)$).

Without masking, inversion in $GF(2^8)/GF(2^4)$ (indicating the representation of $GF(2^8)$ as vectors over $GF(2^4)$) using a normal basis $[\mathbf{Y}^{16}, \mathbf{Y}]$, where $\mathbf{Y}$ and $\mathbf{Y}^{16}$ are the roots of $\mathbf{X}^2 + \mathbf{X} + N$ and $N \in GF(2^4)$ is the norm (product: $N = \mathbf{Y}^{17}$), is given by [2]:

$$\mathbf{A} = A_1\,\mathbf{Y}^{16} + A_0\,\mathbf{Y} \quad \text{(given)} \ , \tag{1}$$

$$B = N \otimes (A_1 \oplus A_0)^2 \ \oplus \ A_1 \otimes A_0 \ , \tag{2}$$

$$\mathbf{A}^{-1} = \left(A_0 \otimes B^{-1}\right)\,\mathbf{Y}^{16} + \left(A_1 \otimes B^{-1}\right)\,\mathbf{Y} \quad \text{(result)} \ . \tag{3}$$

(Note that $\otimes$ and $\oplus$ denote multiplication and addition calculations in a Galois field, while $A_1\,\mathbf{Y}^{16} + A_0\,\mathbf{Y}$ is just the algebraic expression for the vector $[A_1, A_0]$ in the normal basis.) This requires inversion, multiplication, and the combined "square-scale" operation ($N \otimes X^2$) in the subfield $GF(2^4)$. Similarly, the inversion in $GF(2^4)/GF(2^2)$ using a normal basis $[Z^4, Z]$, where Z and Z^4 are the roots of $X^2 + X + \mathbf{n}$ and $\mathbf{n} \in GF(2^2)$ is the norm ($\mathbf{n} = Z^5$), is given by:

$$B = \mathbf{b}_1\,Z^4 + \mathbf{b}_0\,Z \quad \text{(given)} \ , \tag{4}$$

$$\mathbf{c} = \mathbf{n} \otimes (\mathbf{b}_1 \oplus \mathbf{b}_0)^2 \ \oplus \ \mathbf{b}_1 \otimes \mathbf{b}_0 \ , \tag{5}$$

$$B^{-1} = \left(\mathbf{b}_0 \otimes \mathbf{c}^{-1}\right)\,Z^4 + \left(\mathbf{b}_1 \otimes \mathbf{c}^{-1}\right)\,Z \quad \text{(result)} \ . \tag{6}$$

But in the sub-subfield $GF(2^2)$, inversion is the same as squaring, equivalent to a bit swap:

$$\mathbf{c} = c_1\,\mathbf{w}^2 + c_0\,\mathbf{w} \quad \text{(given)} \ , \tag{7}$$

$$\mathbf{c}^{-1} = c_0\,\mathbf{w}^2 + c_1\,\mathbf{w} \quad \text{(result)} \ , \tag{8}$$

where $\mathbf{w}$ and $\mathbf{w}^2$ are the roots of $\mathbf{x}^2+\mathbf{x}+1$. (While this algebraic description uses the Galois inverse, the case of a zero element in any of these fields is correctly handled: the zero element is returned in lieu of an inverse.)

2.2 Masked Inversion

Now introduce additive masking. By adding a random mask, such that the statistical distribution of masks is uniform over the field, now our operands appear random as well, uncorrelated to either plaintext or key. Hence the statistical data available through side channels appears as noise, independent of the chosen sets of plaintexts, and the key is protected against first-order differential attacks. The cost is the computation of mask correction terms that are combined with the given masked input to correctly mask the output. Here we outline the algebraic steps involved; we later show that this masking scheme is secure in 2.5.

We use the insight of Oswald *et al.* that in the sub-subfield $GF(2^2)$ inversion (squaring) is additive, so for data $\mathbf{a}$ and mask $\mathbf{m}$, then

$$(\mathbf{a} \oplus \mathbf{m})^{-1} = (\mathbf{a} \oplus \mathbf{m})^2 = \mathbf{a}^2 \oplus \mathbf{m}^2 = \mathbf{a}^{-1} \oplus \mathbf{m}^{-1} \ , \tag{9}$$

and finding the mask correction $\mathbf{m}^2$ is trivial. Hence the tower-field approach eliminates the need to remove the additive mask (or change it to a multiplicative one) before inversion.

In the larger fields, here is how the mask corrections can be calculated. We indicate the masked version of the input byte $\mathbf{A}$ with a tilde: $\tilde{\mathbf{A}}$, and similarly for the other masked quantities. So the input to the masked $GF(2^8)$ inverter is the data byte $\mathbf{A}$ already masked by the (known) mask $\mathbf{M}$

$$\tilde{\mathbf{A}} = (\mathbf{A} \oplus \mathbf{M}) = \tilde{A}_1 \, \mathbf{Y}^{16} + \tilde{A}_0 \, \mathbf{Y} \ , \tag{10}$$

$$\mathbf{M} = M_1 \, \mathbf{Y}^{16} + M_0 \, \mathbf{Y} \ . \tag{11}$$

Also, for later reference, let $M_1 = \mathbf{m}_{11} \, Z^4 + \mathbf{m}_{10} \, Z$.

To mask the first calculation, we need to introduce a fresh 4-bit mask Q that is independent of M. Let

$$\begin{aligned} \tilde{B} = Q \ \oplus \ N \otimes \Big(\tilde{A}_1 \oplus \tilde{A}_0 \Big)^2 \ \oplus \ N \otimes (M_1 \oplus M_0)^2 \\ \oplus \ \tilde{A}_1 \otimes \tilde{A}_0 \ \oplus \ \tilde{A}_1 \otimes M_0 \ \oplus \ \tilde{A}_0 \otimes M_1 \ \oplus \ M_1 \otimes M_0 \ , \end{aligned} \tag{12}$$

with the result $\tilde{B}$ being B above, masked by Q. Here (and below) the terms must be added in sequence, starting with the mask, to keep each intermediate result uniformly distributed, as discussed later in 2.5.

For the subfield inversion, say $\tilde{B} = \tilde{\mathbf{b}}_1 \, Z^4 + \tilde{\mathbf{b}}_0 \, Z$ and $Q = \mathbf{q}_1 \, Z^4 + \mathbf{q}_0 \, Z$. Then we need introduce a new 2-bit mask $\mathbf{r}$ that is independent of Q (but we are free to re-use bits from $\mathbf{M}$ for $\mathbf{r}$). Let

$$\begin{aligned} \tilde{\mathbf{c}} = \mathbf{r} \ \oplus \ \mathbf{n} \otimes \Big(\tilde{\mathbf{b}}_1 \oplus \tilde{\mathbf{b}}_0 \Big)^2 \ \oplus \ \mathbf{n} \otimes (\mathbf{q}_1 \oplus \mathbf{q}_0)^2 \\ \oplus \ \tilde{\mathbf{b}}_1 \otimes \tilde{\mathbf{b}}_0 \ \oplus \ \tilde{\mathbf{b}}_1 \otimes \mathbf{q}_0 \ \oplus \ \tilde{\mathbf{b}}_0 \otimes \mathbf{q}_1 \ \oplus \ \mathbf{q}_1 \otimes \mathbf{q}_0 \ , \end{aligned} \tag{13}$$

so $\tilde{\mathbf{c}}$ is $\mathbf{c}$ above, masked by $\mathbf{r}$.

In the sub-subfield, say $\tilde{\mathbf{c}} = \tilde{c}_1\,\mathbf{w}^2 + \tilde{c}_0\,\mathbf{w}$, $\mathbf{r} = r_1\,\mathbf{w}^2 + r_0\,\mathbf{w}$, and let

$$\tilde{\mathbf{c}}^{-1} = \tilde{c}_0\,\mathbf{w}^2 + \tilde{c}_1\,\mathbf{w} \quad \text{(bit swap)} \ , \tag{14}$$

$$\mathbf{r}^2 = r_0\,\mathbf{w}^2 + r_1\,\mathbf{w} \quad \text{(bit swap)} \ , \tag{15}$$

so $\tilde{\mathbf{c}}^{-1}$ is $\mathbf{c}^{-1}$ above masked the uniform mask $\mathbf{r}^2$ (the square of $\mathbf{r}$).

The next steps would involve only multiplications, and directly adding *any* of the correction terms would reveal a distribution that depends on the data. Again, the mask must be added *first*, then the other mask correction terms added individually to the sum, to maintain the uniform distribution for intermediate results.

To mask the next multiplication, we introduce another 4-bit mask $T = \mathbf{t}_1\,Z^4 + \mathbf{t}_0\,Z$ (as in Oswald *et al.*[1]), which must be independent of both Q and $\mathbf{r}$ for this step, and should also be independent of $\mathbf{M}$ for the next. (More precisely, $\mathbf{t}_1$ must be independent of $\mathbf{q}_0$, and $\mathbf{t}_0$ independent of $\mathbf{q}_1$, so we could just let $T = Q$.) Let

$$\tilde{\mathbf{b}}_1^{-1} = \mathbf{t}_1 \ \oplus \ \tilde{\mathbf{b}}_0 \otimes \tilde{\mathbf{c}}^{-1} \ \oplus \ \tilde{\mathbf{b}}_0 \otimes \mathbf{r}^2 \ \oplus \ \mathbf{q}_0 \otimes \tilde{\mathbf{c}}^{-1} \ \oplus \ \mathbf{q}_0 \otimes \mathbf{r}^2 \ , \tag{16}$$

$$\tilde{\mathbf{b}}_0^{-1} = \mathbf{t}_0 \ \oplus \ \tilde{\mathbf{b}}_1 \otimes \tilde{\mathbf{c}}^{-1} \ \oplus \ \tilde{\mathbf{b}}_1 \otimes \mathbf{r}^2 \ \oplus \ \mathbf{q}_1 \otimes \tilde{\mathbf{c}}^{-1} \ \oplus \ \mathbf{q}_1 \otimes \mathbf{r}^2 \ , \tag{17}$$

so that the result $\tilde{B}^{-1} = \tilde{\mathbf{b}}_1^{-1}\,Z^4 + \tilde{\mathbf{b}}_0^{-1}\,Z$ is B^{-1} above, masked by T (but is *not* the inverse of $\tilde{B}$).

Similarly for the next multiplication, introduce a new 8-bit mask $\mathbf{S} = S_1\,\mathbf{Y}^{16} + S_0\,\mathbf{Y}$ for the output, where $\mathbf{S}$ must be independent of $\mathbf{M}$ and T. (More precisely, S_1 must be independent of M_0, and S_0 independent of M_1, so $\mathbf{S}$ may be $\mathbf{M}$.) Let

$$\tilde{A}_1^{-1} = S_1 \ \oplus \ \tilde{A}_0 \otimes \tilde{B}^{-1} \ \oplus \ \tilde{A}_0 \otimes T \ \oplus \ M_0 \otimes \tilde{B}^{-1} \ \oplus \ M_0 \otimes T \ , \tag{18}$$

$$\tilde{A}_0^{-1} = S_0 \ \oplus \ \tilde{A}_1 \otimes \tilde{B}^{-1} \ \oplus \ \tilde{A}_1 \otimes T \ \oplus \ M_1 \otimes \tilde{B}^{-1} \ \oplus \ M_1 \otimes T \ , \tag{19}$$

so that the result $\tilde{\mathbf{A}}^{-1} = \tilde{A}_1^{-1}\,\mathbf{Y}^{16} + \tilde{A}_0^{-1}\,\mathbf{Y}$ is the answer $\mathbf{A}^{-1}$ above, masked by the output mask $\mathbf{S}$:

$$\tilde{\mathbf{A}}^{-1} = \mathbf{A}^{-1} \oplus \mathbf{S} \ . \tag{20}$$

2.3 Re-using Masks

Oswald *et al.*[1] showed that through using parts of the input mask for the intermediate results, then several operations can be eliminated, notably multiplications (for the cost of a few additions). We will follow the same strategy below. (While re-using masks could make the implementation more vulnerable to higher-order differential side-channel analysis, it remains secure against first-order attacks.)

The first place (other than using part of $\mathbf{M}$ for $\mathbf{r}$) where re-using masks helps is in the masked intermediate result $\tilde{\mathbf{c}}^{-1}$, where for one subsequent calculation the mask $\mathbf{q}_1$ would be helpful but for another the preferred mask would be $\mathbf{q}_0$, so

we follow [1] and switch masks. Then from (14) above we modify the calculation as follows, starting by re-masking $\tilde{\mathbf{c}}^{-1}$:

$$\tilde{\mathbf{c}}^{-1} = \left[\tilde{c}_0\,\mathbf{w}^2 + \tilde{c}_1\,\mathbf{w}\right] \;\oplus\; \left(\mathbf{q}_1 \oplus \mathbf{r}^2\right) \;\;, \tag{21}$$

$$\tilde{\mathbf{b}}_1^{-1} = \mathbf{m}_{11} \;\oplus\; \tilde{\mathbf{b}}_0 \otimes \tilde{\mathbf{c}}^{-1} \;\oplus\; \underline{\tilde{\mathbf{b}}_0 \otimes \mathbf{q}_1} \;\oplus\; \mathbf{q}_0 \otimes \tilde{\mathbf{c}}^{-1} \;\oplus\; \underline{\mathbf{q}_0 \otimes \mathbf{q}_1} \;\;, \tag{22}$$

$$\tilde{\mathbf{c}}_2^{-1} = \tilde{\mathbf{c}}^{-1} \;\oplus\; (\mathbf{q}_0 \oplus \mathbf{q}_1) \;\;, \tag{23}$$

$$\tilde{\mathbf{b}}_0^{-1} = \mathbf{m}_{10} \;\oplus\; \tilde{\mathbf{b}}_1 \otimes \tilde{\mathbf{c}}_2^{-1} \;\oplus\; \underline{\tilde{\mathbf{b}}_1 \otimes \mathbf{q}_0} \;\oplus\; \mathbf{q}_1 \otimes \tilde{\mathbf{c}}_2^{-1} \;\oplus\; \underline{\mathbf{q}_1 \otimes \mathbf{q}_0} \;\;, \tag{24}$$

where the underlined products had already been computed previously and may be re-used. (Parens indicate the order of evaluation necessary to avoid unmasking operands. In the actual optimized code, some of the details of the specific bits added above are different, but with the same security properties.) The result $\tilde{B}^{-1} = \tilde{\mathbf{b}}_1^{-1}\,Z^4 + \tilde{\mathbf{b}}_0^{-1}\,Z$ is still B^{-1} above, but now masked by $M_1 = \mathbf{m}_{11}\,Z^4 + \mathbf{m}_{10}\,Z$, the upper half of the input mask.

Applying the same approach of switching masks at the next level depends on whether or not the output mask $\mathbf{S}$ is independent of the input mask $\mathbf{M}$. If $\mathbf{S}$ is independent (so Q above could be any 4 bits of $\mathbf{S}$), then

$$\tilde{A}_1^{-1} = S_1 \;\oplus\; \tilde{A}_0 \otimes \tilde{B}^{-1} \;\oplus\; \underline{\tilde{A}_0 \otimes M_1} \;\oplus\; M_0 \otimes \tilde{B}^{-1} \;\oplus\; \underline{M_0 \otimes M_1} \;\;, \tag{25}$$

$$\tilde{B}_2^{-1} = \tilde{B}^{-1} \oplus (M_0 \oplus M_1) \;\;, \tag{26}$$

$$\tilde{A}_0^{-1} = S_0 \;\oplus\; \tilde{A}_1 \otimes \tilde{B}_2^{-1} \;\oplus\; \underline{\tilde{A}_1 \otimes M_0} \;\oplus\; M_1 \otimes \tilde{B}_2^{-1} \;\oplus\; \underline{M_1 \otimes M_0} \;\;, \tag{27}$$

again allowing the underlined products to be re-used, and with the output $\tilde{\mathbf{A}}^{-1} = \tilde{A}_1^{-1}\,\mathbf{Y}^{16} + \tilde{A}_0^{-1}\,\mathbf{Y}$ having the output mask $\mathbf{S}$:

$$\tilde{\mathbf{A}}^{-1} = \mathbf{A}^{-1} \oplus \mathbf{S} \;\;. \tag{28}$$

However, if one chooses to use the same mask for output and input ($\mathbf{S} = \mathbf{M}$) then a temporary independent mask, such as the earlier 4-bit mask Q, is required to protect the additions:

$$\tilde{A}_1^{-1} = Q \oplus \tilde{A}_0 \otimes \tilde{B}^{-1} \oplus \underline{\tilde{A}_0 \otimes M_1} \oplus M_0 \otimes \tilde{B}^{-1} \oplus \underline{M_0 \otimes M_1} \oplus (M_1 \oplus Q) \;\;, \tag{29}$$

$$\tilde{B}_2^{-1} = \tilde{B}^{-1} \oplus (M_0 \oplus M_1) \;\;, \tag{30}$$

$$\tilde{A}_0^{-1} = Q \oplus \tilde{A}_1 \otimes \tilde{B}_2^{-1} \oplus \underline{\tilde{A}_1 \otimes M_0} \oplus M_1 \otimes \tilde{B}_2^{-1} \oplus \underline{M_1 \otimes M_0} \oplus (M_0 \oplus Q) \;\;, \tag{31}$$

with the terms added in sequence, securely gives

$$\tilde{\mathbf{A}}^{-1} = \mathbf{A}^{-1} \oplus \mathbf{M} \;\;. \tag{32}$$

2.4 Re-using Masks between Rounds

Many of the mask correction terms used in the masked inversion above involve *only* the random masks, independent of the masked data. This is also true of *all* the mask correction term calculations in the other steps of each round of encryption, as those other steps are all linear (with respect to the additive mask).

Then, if the original 128-bit mask for a block of data were to be re-used for every round, all those data-independent correction terms would be the same for each round. For implementations where the round loop is "unrolled," with S-boxes for each round, these terms would only need computing once, then could be passed along to all the other rounds. This would save the re-computation of all those mask terms, eliminating the associated circuitry, at the modest cost of the "wiring" required to pass along the correction terms. Of course, one would use a new random mask with each new block of data in Round 0, to ensure that over time the distribution of masks remains uniform.

More precisely, one way to do this starts by picking a random 128-bit mask that will be used as the *output* mask (whose bytes correspond to **S** above) from the inversion step. Then after each byte undergoes the basis change (from the tower field form) combined with affine transformation part of the S-box (excluding the additive constant), the *ShiftRows* step is applied to the whole mask; the result is the output mask after the last round of encryption (which lacks the *MixColumns* step). Then *MixColumns* is applied to that, giving the *input* mask to be added to the initial data before Round 0. Applying byte-wise the basis change (to the tower field form) gives the input mask (corresponding to **M** above) for the inversion step. (The *ShiftRows* and *MixColumns* recombine mask bits, ensuring that for each S-box **S** and **M** are independent.) Then choosing 4 bits of **S** for Q (and 2 bits of **M** for **r**), the data-independent terms such as $M_1 \otimes M_0$, $N \otimes (M_1 \oplus M_0)^2$, $\mathbf{q}_1 \otimes \mathbf{q}_0$, and $\mathbf{n} \otimes (\mathbf{q}_1 \oplus \mathbf{q}_0)^2$ above may be precomputed, to be re-used each round. Then the only correction terms that would need computing in each round are the data-dependent terms (e.g. $\tilde{A}_1 \otimes M_0$ above) of the inversion step.

But this only makes sense if the application has enough room to unroll (at least partly) the round loop. (While unrolling the rounds does not improve latency, this saving of correction terms may make unrolling preferable to simply duplicating more encryptors for increased throughput.) In cases where compactness is paramount the same few S-boxes would be employed for each round; using pre-computed correction terms from round to round would then require extra registers – a cost rather than a saving.

2.5 Security of Masked Operands

Here we show that the masked inversion operation outlined above is secure, by which we mean, assuming a source of truly random uniformly distributed masks, then the distribution of each intermediate result is independent of both the plaintext data and the key. This gives "perfect masking" in the terminology of [3], and hence protection from first-order differential side-channel attacks. We start with Lemmas 1 and 2 of [3] (paraphrased) and, to be thorough, add two more lemmas to cover all the operations in inversion, which are then examined in detail.

Lemma 1. *Given x uniformly distributed over a finite field $\mathbb{F}$, and any $y \in \mathbb{F}$ independent of x, then $z = x \oplus y$ is also uniformly distributed and independent of y.[3]*

Lemma 2. *Given x and y independent and both uniformly distributed over a finite field $\mathbb{F}_q$, then $z = x \otimes y$ is distributed according to*

$$Pr(z = i) = \begin{cases} (2q-1)/q^2 \ , \ i = 0 \\ (q-1)/q^2 \ \ , \ i \neq 0 \end{cases}$$

here called the random product distribution.[3]

Lemma 3. *Given x uniformly distributed over a finite set $\mathbb{F}$, and a one-to-one mapping $f : \mathbb{F} \to \mathbb{F}$, then $y = f(x)$ is also uniformly distributed.*

Proof: For a finite set, any one-to-one mapping is a bijection, i.e., just a permutation of the elements, so a uniform distribution is unchanged. (Note in particular that any isomorphism of a finite field is a bijection.)

Lemma 4. *Given $\boldsymbol{x} = [x_1, x_2, \cdots, x_{2n}]$ uniformly distributed over the set $\mathbb{F}^{2n}$ of ordered $2n$-tuples from a finite set $\mathbb{F}$, then the two halves $\boldsymbol{y}_1 = [x_1, x_2, \cdots, x_n]$ and $\boldsymbol{y}_2 = [x_{n+1}, x_{n+2}, \cdots, x_{2n}]$ of $\boldsymbol{x}$ are independent and uniformly distributed over $\mathbb{F}^n$.*

Proof: Since $\boldsymbol{x}$ is uniform if and only if each x_i is independently uniform over $\mathbb{F}$, then $\boldsymbol{y}_1$ and $\boldsymbol{y}_2$ are independent and uniform. (So given a uniform mask of n bits then any sub-mask is also uniform.)

Consider the operations in the masked inversion above. Initially adding a uniform mask to the plaintext data gives a uniform result, such as $\tilde{\mathbf{A}}$ in (10) above, with independent uniform halves $\tilde{A}_1, \tilde{A}_0$. Then $\tilde{A}_1 \oplus \tilde{A}_0$ is uniform, so is its square (squaring is an isomorphism in a field of characteristic 2), and so is that square scaled by the norm N, since multiplication by a nonzero constant is a one-to-one mapping. Similarly $N \otimes (M_1 \oplus M_0)^2$ is uniform. Each of the other four products in (12) for $\tilde{B}$ has the random product distribution.

Now the order of adding these pieces together is crucial. Adding any pair of those four products would give a distribution that depends on the data. And while the original version[28] of the current work used the uniform term $N \otimes \left(\tilde{A}_1 \oplus \tilde{A}_0\right)^2$ to mask the sum of those four products, with a uniform result, *every* order in which those five terms might be added would reveal a data-dependent distribution at some point[29]. But if we *start* with the *independent* uniform mask Q and successively add terms to the previous independent uniform sum, then each resulting sum is uniform, including $\tilde{B}$, by Lemma 1.

Similar reasoning applies to $\tilde{\mathbf{c}}$ in (13), starting with the independent uniform mask $\mathbf{r}$. Then swapping the bits of $\tilde{\mathbf{c}}$ in (21) is also uniform, with the uniform mask $\mathbf{r}^2$; hence the parenthesized order of summation for $\tilde{\mathbf{c}}^{-1}$ in (21) is important to avoid unmasking (while remasking). Again, in (22) for $\tilde{\mathbf{b}}_1^{-1}$, each of the four products has the random product distribution, and the *first* sum must be with the independent uniform mask $\mathbf{m}_{11}$, then each other product added successively, so each result is uniform. And again, in switching masks for $\tilde{\mathbf{c}}_2^{-1}$ in (23) the order of summation is important to prevent unmasking. All the remaining steps are similar to those already discussed.

Therefore, the result of every calculation is either uniformly distributed or has the random product distribution (provided the summations are performed in the correct order), independent of the data and key: "perfect masking." This protects suitable implementations from attacks by first-order differential side-channel analysis. (Resistance against higher-order attacks would be improved by avoiding re-use of masks within the S-box, as well as between rounds.)

3 Optimizations and Results

The algebraic description above shows the most efficient masked inversion at that algebraic, hierarchical level. At the bit level, further optimization is possible due to certain bit combinations being useful in more than one place. And at the logic gate level, certain substitutions of types of logic gates give further savings in circuit size (for the standard cell library considered).

Here we briefly describe the main optimizations, which are essentially the same as in [2], but applied also to mask correction calculations. Each Galois multiplier involves bit sums, and since all factors are shared between two multipliers (using normal bases), these bit sums can be re-used; some of them are also useful as parts of the square-scale operation. Gate-level optimizations (minimizing the size for the 0.13-μ CMOS standard cell library[30] considered) include replacing AND by NAND, and where possible, replacing a NAND and some XORs by a single NOR. (We only consider two-input logic gates; using gates with more inputs may allow some improvement.)

But the S-box involves more than inversion in the tower-field representation. The affine transformation and conversion to/from the tower-field form are also required. (While one approach is to use the tower-field form for the entire encryption round[11], we use it only for the inversion, so that the *MixColumns* step remains simple.) For decryption, the inverse affine transformation is required. The conversion between the standard form and the tower-field form can be done through multiplying the byte by a bit matrix, and one of those two matrices can be combined with the bit matrix multiply of the affine transformation (or inverse). Satoh *et al.*[12] showed how, when both encryption and decryption are required, an architecture sharing a single Galois inverter between an S-box and an inverse S-box allows some further optimization in the four matrices involved. We use the particular normal bases of [2] and the corresponding optimally factored matrices to minimize this computation.

We wrote a complete implementation, with all optimizations, of the masked S-box and inverse S-box using the merged architecture of [12], with a shared Galois inverter. The complete implementation, written in Verilog, is given in [31]. Here both the input mask and the output mask are parameters, along with the masked data byte.

Tables 1 and 2 give the resulting numbers of logic gates separately for the masked Galois inverter and the basis change (bit matrices). Results are shown by number and type of specific logic operations, and also by total "gates," where the number refers to the equivalent number of NAND gates (rounded

to whole numbers), using our standard cell library. We use the equivalencies 1 XOR/XNOR = $\frac{7}{4}$ NAND gates, 1 NOR = 1 NAND gate, 1 NOT = $\frac{3}{4}$ NAND gate, and 1 MUX21I = $\frac{7}{4}$ NAND gates[30]. Table 3 gives the total size of each resulting S-box.

Two cases are considered for masking: using two independent 8-bit masks for input and output; and using a single 8-bit mask for both input and output, which also requires providing an independent 4-bit mask (Q above) for temporary use. (The latter case is called "1 mask" although perhaps "1.5 masks" would be more accurate.) The 1-mask approach needs a bigger inverter but smaller basis change.

Table 1. Inverter Size. Here we compare the masked inverter with the unmasked version, where total gates is in NAND equivalents.

Inverter	gate counts	total gates
2 masks	231 XOR, 94 NAND, 6 NOR	504
1 mask	247 XOR, 94 NAND, 6 NOR	532
unmasked	56 XOR, 34 NAND, 6 NOR	138

Table 2. Basis Change Sizes. Here we compare gates needed in the basis change bit matrices (including the affine transformation but excluding the Galois inverter) for a merged S-box & inverse, S-box alone, and inverse S-box alone, using different input and output masks, same mask for both, or no mask. Both individual gate counts and NAND equivalents are given.

Basis Change	merged	S-box	$(\text{S-box})^{-1}$
2 masks	78 XOR, 4 NOT, 32 MUX = 196	49 XOR = 86	50 XOR = 88
1 mask	58 XOR, 3 NOT, 32 MUX = 160	44 XOR = 77	45 XOR = 79
unmasked	38 XOR, 2 NOT, 16 MUX = 96	24 XOR = 42	25 XOR = 44

Table 3. S-box Sizes. Here we give the sizes, in NAND equivalents, for the different complete S-box implementations, including both the inverter and the basis change.

S-Box	merged	S-box	$(\text{S-box})^{-1}$
2 masks	700	590	592
1 mask	692	609	611
unmasked	234	180	182

Note that for the merged architecture, using different masks on input and output requires slightly more resources resources than using a single mask, but

not for dedicated encryption (or decryption) only. Since the difference is slight, from here on we only consider using two masks. In this case, the size for the merged architecture where encryption and decryption share an inverter is 700 NAND equivalents, three times the size of the unmasked version (234 gates). (For encryption only, not merged, the masked S-box is 590 NAND equivalents, compared with 180 for unmasked; here masked is over three times larger.)

However, if the current approach were used in an application where the loop of rounds was "unrolled" (requiring enough room for at least 160 S-boxes for complete unrolling), the masks could be re-used from round to round, as discussed above in 2.4. This would require passing along the extra bits of pre-computed corrections between rounds. For one S-box, the total number of mask-term bits would be 46, as compared to 16 bits for masks alone. These 30 extra wires would replace 43 XORs and 12 NANDs in the inverter, and *all* of the mask basis change calculation (so the basis change would be as if unmasked). The gains from this re-use between rounds is shown in Table 4; then a masked merged S-box is 513 NAND equivalents, rather than the 700 above. In addition to this saving per S-box (after the first round), the *MixColumns* operation on the mask block would also be eliminated (again, after the first round).

Table 4. Gains from re-using masks *between* rounds: the complete S-box, in NAND equivalents. (This re-use requires unrolling the round loop with many copies of the S-box.)

masking	merged	S-box	$(\text{S-box})^{-1}$
2 masks	700	590	592
re-use	513	459	461

Table 5. High-level comparison of masking schemes: $GF(2^4)$ operations, from Table 1 of [1] with a new row for the current work, and a new column for the Square-Scale optimized combination [addition and inverse operations not shown].

method	Mult	MultConst	Square	Square-Scale
S-Akkar	18	6	4	0
S-Blömer	12	1	2	0
MS-IAIK	9	2	2	0
this work	8	0	0	2

Direct comparison with Oswald *et al.*[1] is difficult at the level of optimization employed here. Their terms of comparison are operations in $GF(2^4)$ and their Table 1 is reproduced here as Table 5, augmented to compare with the present work. They compare their approach (MS-IAIK) to their implementations of two

previous methods, and do not include addition and inversion operations, presumably because addition is relatively small (4 XORs) and one subfield inversion is assumed. The column for multipliers in $GF(2^4)$ is the most significant (since each $GF(2^4)$ multiplier includes 3 multipliers and 4 additions in $GF(2^2)$) and our approach saves one multiplier at this level. It is not clear how they implemented squaring, and multiplication by a constant, called "scaling" here; in our approach with normal bases, squaring is always followed with scaling by the norm, so we treat square-scale as a single operation, which we have optimized down to only 3 XORs, *less* than a 4-bit addition, which are not counted in [1].

Some rough comparison is possible in the algebraic description at the $GF(2^4)$ level: our (12) is approximately comparable to their (19), while our (25), (27) are like their (15), (17) respectively. In particular, their (19) (which includes 4 squares, though only 2 appear in [1, Table 1]) shows more terms than our(12). So we save at least some $GF(2^4)$ additions, and again at the $GF(2^2)$ level. But while detailed comparison is difficult in the lack of specifics, we are confident that ours is the smallest masked S-box to date, because it uses the same optimizations as the smallest unmasked S-box to date.

4 Conclusion

Side-channel attacks can be a major concern for certain applications of AES, including hardware applications with limited resources, such as smart cards. Adding random masks can be an effective countermeasure, though at the cost of computing mask corrections in the S-box (the rest of each round being linear). Here we show how to compute the Galois inverse (the nonlinear part of the S-box) with "perfect masking," in that the distributions of *all* the masked operands are independent of the chosen plaintext and key; hence suitable implementations employing this method are secure against first-order differential side-channel attacks. (Though, as discussed in the Introduction, CMOS implementations might be vulnerable to DPA attacks, due to glitches[26], unless specific timing constraints are met[27].) While our approach is similar to [1], we have reduced the number of operations at every level. We have optimized this masked S-box for minimal chip area, giving the smallest masked S-box of which we are aware.

The overhead for masking triples the size of the S-box, from 234 gates (NAND equivalents) to 700 gates for the merged version. In applications with sufficient resources to unroll the round loop (where the compactness of our S-box allows more copies for a given area), this overhead may be reduced through re-using the block mask between rounds. Then (after the first round) each S-box would require only 513 gates, a little over twice the size of the unmasked version, and the mask correction for the rest of each round would also be eliminated.

Acknowledgements

We would like to thank the reviewers for several helpful suggestions.

References

1. Oswald, E., Mangard, S., Pramstaller, N., Rijmen, V.: A side-channel analysis resistant description of the AES S-box. In Gilbert, H., Handschuh, H., eds.: Fast Software Encryption: 12th International Workshop, FSE 2005. Volume 3557 of Lecture Notes in Computer Science., Springer-Verlag (2005) 413–23
2. Canright, D.: A very compact S-box for AES. In Rao, J.R., Sunar, B., eds.: Proceedings of 7th International Workshop on Cryptograpic Hardware and Embedded Systems (CHES). Volume 3659 of Lecture Notes in Computer Science., Springer-Verlag (2005) 441–455
3. Blömer, J., Guajardo, J., Krummel, V.: Provably secure masking of AES. In Handschuh, H., Hasan, M.A., eds.: Selected Areas in Cryptography, 11th International Workshop, Springer-Verlag (2004) 69–83
4. Daemen, J., Rijmen, V.: The Design of Rijndael, AES - The Advanced Encryption Standard. Springer-Verlag (2002)
5. NIST: Specification for the ADVANCED ENCRYPTION STANDARD (AES). Technical Report FIPS PUB 197, National Institute of Standards and Technology (NIST) (November 2001)
6. Morioka, S., Satoh, A.: A 10 Gbps full-AES crypto design with a twisted-BDD S-box architecture. In: IEEE International Conference on Computer Design, IEEE (2002) 98–103
7. Weaver, N., Wawrzynek, J.: High performance, compact AES implementations in Xilinx FPGAs. available at `http://www.cs.berkeley.edu/~nweaver/papers/AES_in_FPGAs.pdf` (September 2002)
8. Jarvinen, K.U., Tommiska, M.T., Skytta, J.O.: A fully pipelined memoryless 17.8 Gbps AES128 encryptor. In: FPGA03, ACM (2003)
9. Hodjat, A., Hwang, D., Lai, B.C., Tiri, K., Verbauwhede, I.: A 3.84 Gbits/s AES crypto coprocessor with modes of operation in a 0.18-μm CMOS technology. In: ACM Great Lakes Symposium on VLSI. (2005) 60–63
10. Morioka, S., Satoh, A.: An optimized S-box circuit arthitecture for low power AES design. In Jr., B.S.K., Çetin Kaya Koç, Paar, C., eds.: Proceedings of 4th International Workshop on Cryptograpic Hardware and Embedded Systems (CHES). Volume 2523 of Lecture Notes in Computer Science., Springer-Verlag (2003) 172–186
11. Rudra, A., Dubey, P., Jutla, C., Kumar, V., Rao, J., Rohatgi, P.: Efficient Rijndael encryption implementation with composite field arithmetic. In Ç.K. Koç, Naccache, D., Paar, C., eds.: Proceedings of 3rd International Workshop on Cryptograpic Hardware and Embedded Systems (CHES). Number 2162 in Lecture Notes in Computer Science, Springer-Verlag (2001) 171–184
12. Satoh, A., Morioka, S., Takano, K., Munetoh, S.: A compact Rijndael hardware architecture with s-box optimization. In Boyd, C., ed.: Proceedings of Advances in Cryptology - ASIACRYPT: 7th International Conference on the Theory and Application of Cryptology and Information Security. Number 2248 in Lecture Notes in Computer Science, Springer-Verlag (2001) 239–254
13. Wolkerstorfer, J., Oswald, E., Lamberger, M.: An ASIC implementation of the AES S-boxes. In Preneel, B., ed.: In Topics in Cryptology - CT-RSA - The Cryptographers' Track at the RSA Conference. Number 2271 in Lecture Notes in Computer Science, Springer-Verlag (2002) 67–78
14. Chodowiec, P., Gaj, K.: Very compact FPGA implementation of the AES algorithm. In Walter, C., Ç.K. Koç, Paar, C., eds.: Proceedings of 5th International

Workshop on Cryptograpic Hardware and Embedded Systems (CHES). Number 2779 in Lecture Notes in Computer Science, Springer-Verlag (2003) 319–333
15. Feldhofer, M., Wolkerstorfer, J., Rijmen, V.: AES implementation on a grain of sand. In: IEE Proceedings on Information Security. Volume 152., IEE (2005) 13–20
16. Rijmen, V.: Efficient implementation of the Rijndael S-box. available at `http://www.esat.kuleuven.ac.be/~rijmen/rijndael/sbox.pdf` (2001)
17. Paar, C.: Efficient VLSI Architectures for Bit-Parallel Computation in Galois Fields. PhD thesis, Institute for Experimental Mathematics, University of Essen, Germany (1994)
18. Courtois, N., Pieprzyk, J.: Cryptanalysis of block ciphers with overdefined systems of equations. In Zheng, Y., ed.: Advances in Cryptology - ASIACRYPT 2002, Proceedings. Volume 2501 of Lecture Notes in Computer Science., Springer-Verlag (2002) 267–287
19. Murphy, S., Robshaw, M.J.B.: Essential algebraic structure within the AES. In Yung, M., ed.: Advances in Cryptology: Proceedings of CRYPTO 2002, Springer-Verlag (2002) 1–16
20. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In Wiener, M.J., ed.: Advances in Cryptology: Proceedings of CRYPTO'99. Number 1666 in Lecture Notes in Computer Science, Springer-Verlag (1999) 388–397
21. Akkar, M.L., Bévan, R., Dischamp, P., Moyart, D.: Power Analysis, What is Now Possible... In Okamoto, T., ed.: Advances in Cryptology, Proceedings of ASIACRYPT 2000. Number 1976 in Lecture Notes in Computer Science, Springer-Verlag (2000) 489–502
22. Akkar, M.L., Giraud, C.: An implementation of DES and AES, secure against some attacks. In Ç.K. Koç, Naccache, D., Paar, C., eds.: Proceedings of 3rd International Workshop on Cryptograpic Hardware and Embedded Systems (CHES). Number 2162 in Lecture Notes in Computer Science, Springer-Verlag (2001) 309–18
23. Golić, J., Tymen, C.: Multiplicative masking and power anaylsis of AES. In Kaliski Jr., B., Ç.K. Koç, Paar, C., eds.: Proceedings of 4th International Workshop on Cryptographic Hardware and Embedded Systems (CHES). Number 2535 in Lecture Notes in Computer Science, Springer-Verlag (2002) 198–212
24. Morioka, S., Akishita, T.: DPA attack to AES S-box circuits over composite fields. Joho Shori Gakkai Shinpojiumu Ronbunshu **2004**(11) (2004) 9C–2 (in Japanese).
25. Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards sound approaches to counteract power-analysis attacks. In Wiener, M.J., ed.: Advances in Cryptology: Proceedings of CRYPTO'99. Number 1666 in Lecture Notes in Computer Science, Springer-Verlag (1999) 398–412
26. Mangard, S., Pramstaller, N., Oswald, E.: Successfully attacking masked AES hardware implementations. In Rao, J.R., Sunar, B., eds.: Proceedings of 7th International Workshop on Cryptograpic Hardware and Embedded Systems (CHES). Volume 3659 of Lecture Notes in Computer Science., Springer-Verlag (2005) 157–171
27. Mangard, S., Schramm, K.: Pinpointing the side-channel leakage of masked AES hardware implementations. In Goubin, L., Matsui, M., eds.: Proceedings of 8th International Workshop on Cryptograpic Hardware and Embedded Systems (CHES). Volume 4249 of Lecture Notes in Computer Science., Springer-Verlag (2006) 76–90
28. Canright, D., Batina, L.: A very compact "perfectly masked" S-box for AES. In Bellovin, S.M., et al., eds.: Proceedings of 6th International Workshop on Applied Cryptography and Network Security (ACNS). Volume 5037 of Lecture Notes in Computer Science., Springer-Verlag (2008) 446–459

29. Canright, D.: Avoid mask re-use in masked Galois multipliers. http://web.nps.navy.mil/~dcanrig/pub/mask_reuse.pdf (November 2008)
30. Satoh, A. personal communication (July 2004)
31. Canright, D.: Masked S-box implementation (Verilog). http://web.nps.navy.mil/~dcanrig/pub/sboxmaskcorr.v (November 2008)