Attacking White-Box AES Constructions

Brendan McMillion
CloudFlare, Inc.
brendan@cloudflare.com

Nick Sullivan
CloudFlare, Inc.
nick@cloudflare.com

ABSTRACT

A white-box implementation of the *Advanced Encryption Standard* (AES) is a software implementation which aims to prevent recovery of the block cipher's master secret key. This paper refines the design criteria for white-box AES constructions by describing new attacks on past proposals which are conceptually very simple and introduces a new family of white-box AES constructions. Our attacks have a decomposition phase, followed by a disambiguation phase. The decomposition phase applies an SASAS-style cryptanalysis to convert the implementation into a simpler form, while the disambiguation phase converts the simpler form into a unique canonical form. It's then trivial to recover the master secret key of the implementation from its canonical form. We move on to discuss the hardness of SPN disambiguation as a problem on its own, and how to construct white-boxes from it. Implementations of all described attacks and constructions are provided on GitHub at

https://github.com/OpenWhiteBox/

CCS Concepts

•**Security and privacy** → *Block and stream ciphers; Cryptanalysis and other attacks;*

Keywords

AES, white-box, self-equivalences

1. INTRODUCTION

White-Box Cryptography is the study of securing block ciphers in the *white-box attack context*, where an adversary obtains an implementation of the algorithm and is allowed to observe/alter every step of its execution (with instantiated cryptographic keys). The goal is to prevent recovery of the cipher's master secret key, thus forcing the adversary to use an implementation which may be defective. Either by being prohibitively expensive to evaluate or giving invalid input/output pairs in a controlled way.

SPRO'16, October 28 2016, Vienna, Austria
© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ISBN 978-1-4503-4576-7/16/10…$15.00
DOI: http://dx.doi.org/10.1145/2995306.2995314

A system designer may wish to deploy a white-box implementation instead of a standard one to lower the risks associated with code lifting and key compromise. The white-box can be combined with other parts of the system in a way which is difficult to decouple, and its functionality may be heavily restricted. Most effort in the field has been dedicated to studying implementations of AES, for compatibility with legacy systems or the hardening thereof [7].

Unfortunately, efficient attacks have been described against all published constructions. These attacks were developed independently and are not immediately related. There has been one past effort to generalize these attacks by Michiels et al. in [9]. We continue their work by developing conceptually simpler attacks and discussing their implications to white-box design.

Section 2 contains mathematical preliminaries. Section 3 develops the generic SPN representation of AES which is well-known to cryptanalysts. Sections 4-6 describe attacks against the two most influential white-box AES constructions. Sections 7-9 investigate strategies for stronger white-box designs.

2. MATHEMATICAL PRELIMINARIES

An affine transformation from n-bits to n-bits is written as $A(x) = Mx + c$. M is the *linear part* and c is the *constant part*. Given an n-bit bijection f, and an m-bit bijection g, $f \mid\mid g$ denotes the $(n+m)$-bit bijection $(f \mid\mid g)(x \mid\mid y) = f(x) \mid\mid g(y)$. If $c \in \mathbb{F}_{2^8}$ is a constant, then $[c] \in \mathbb{F}_2^{8\times 8}$ denotes the matrix of multiplication by c. By $\mathbb{F}_{2^8}$, we will always mean Rijndael's field.

2.1 Parasitic Functions

In the expression $b^{-1} \circ f \circ a$, where b, f, a are n-bit bijections, we call a and b *parasites* of f. a and b are usually understood to be elements of a distinguished subgroup P of the bijections on n-bits, such as the invertible linear or affine transformations.

A pair of parasites $(a, b) \in P \times P$ such that $f \circ a = b \circ f$ is called a P *self-equivalence* of f. The set of all (a, b) pairs that satisfy this relation are called the P *group* of f (or the *parasitic group* when we speak generally), because they form the product group of two isomorphic subgroups of P. Alternatively, a pair (a, b) such that $f \circ a = b \circ g$ is called a P-*equivalence* of f and g, and information about the P group of f or g gives more P-equivalences, and information about the P group of the other function [8]. It is straightforward to generalize any of these ideas to non-bijective functions.

3. ADVANCED ENCRYPTION STANDARD

The typical algorithm for encrypting with AES is not always the best for studying the cipher because its subroutines, `AddRoundKey`, `SubBytes`, etc., appear atomic even though many of them are completely linear and therefore interact heavily.

In contrast to this, the study of structural cryptanalysis strips all semantic content and describes constructions only as the composition of functions from special families. The generalization offered for substitution-permutation networks (SPNs) is interleaved affine and S-box layers. An affine layer, denoted by an A, treats its input as an element of $\mathbb{F}_2^n$ and applies a fixed, invertible affine transformation over this space. An S-box layer, denoted by S, applies possibly independent m-bit S-boxes to consecutive chunks of its input; we assume S-boxes are arbitrary bijections throughout the paper. We can concisely represent a cipher's structure by concatenating these identifiers – for example, ASAS implies a function $E = A_2 \circ S_2 \circ A_1 \circ S_1$.

Let's find the generalized form of AES, starting with its subroutines:

`AddRoundKey` Add a round key from $\mathbb{F}_2^{128}$ to the block. (A)

`MixColumns` Apply an $\mathbb{F}_{2^8}$-linear transformation to each word of the block. (A)

`SubBytes` Invert each byte as an element of $\mathbb{F}_{2^8}$ and apply an affine transformation. (AS)

`ShiftRows` Permute the bytes of each block. (A)

Composing them, according to AES's specification, results in a large cascade of the shape $\ldots\mathsf{AASAA}$, but for clarity we collapse neighboring affine layers until the two kinds alternate. We're left with a function in $\mathsf{ASASA}\ldots\mathsf{ASASA}$ form, with 10 S-box and 11 affine layers. We call this AES's generic SPN representation. It's interesting to note that only the constant part of each affine layer is key dependent – everything else is public.

The first three images in Figure 1 visualize the linear part of each subroutine's affine layer – `SubBytes`[1] is dense but byte-wise independent, while `ShiftRows` is simply a permutation matrix. `MixColumns` is the $\mathbb{F}_2$-matrix representation of multiplication by elements of $\mathbb{F}_{2^8}$. The final matrix on the right is

$$A_i = \texttt{AddRoundKey}_i \circ \texttt{MixColumns} \circ \texttt{ShiftRows} \circ \texttt{SubBytes}^1$$

that is, the linear part of every intermediate affine layer of AES when written in generic form. The first affine layer of the cascade is simply $\texttt{AddRoundKey}_0$ and the last is

$$\texttt{AddRoundKey}_{10} \circ \texttt{ShiftRows} \circ \texttt{SubBytes}^1$$

[4].

4. PAST CONSTRUCTIONS

The white-box AES constructions of [7] and [11] compute AES encryption with a network of lookup tables. An arc from one table T to another T' means that (part of) the output of T is used as (part of) the input to table T'. The exact construction of the tables is very complicated and not very insightful so we describe it only as necessary.

The tables in the network are encoded, meaning that the output of each has been randomly masked somehow and the tables it inputs to have been adjusted so as not to break overall functionality. As constructed, this typically only hides low-level information such as the exact bits that are being computed on at a given point. It does not mask the high-level information of multiset properties and round boundaries. Because of this, we can pick out sections of the network which correspond to one or two consecutive rounds. This will necessarily have a small SPN structure which we can decompose into its individual layers with the techniques of Biryukov and Shamir from [4].

One round of AES has the structure AS. Chow et al. obfuscate the network by adding random *non-linear* maps to the tables' inputs and outputs, meaning the structure of one round in their construction is $\mathsf{S(AS)S} = \mathsf{SAS}$. Xiao and Lai take the complementary approach by adding *linear* maps to the input and output of all tables, making the round structure $\mathsf{A(AS)A} = \mathsf{ASA}$. These pose no problem, as [4] describes practical attacks against structures as large as SASAS.

This decomposed SPN is unlikely to be correct because generic SPNs have a large number of equivalent representations. We derive the canonical representation in a more ad-hoc way than we decomposed the network, but show that it is still easy.

5. CRYPTANALYSIS OF CHOW ET AL.

For each round, the construction of Chow et al. produces a network which computes

$$R_i := Q_i \circ \texttt{MixColumns} \circ \texttt{SubBytes} \circ \texttt{AddRoundKey}_i \circ P_i$$

for $0 \leq i \leq 10$, where the input encoding P_i and the output encoding Q_i apply 32 independent, random 4-bit S-boxes to the state array; to allow transparent composition, each round's input encoding inverts the previous output encoding, $P_i = Q_{i-1}^{-1}$. P_0 and Q_{10} are called *external encodings* and are arbitrary (normally affine) transformations.

Note that the round structure is slightly out of order. Before, `ShiftRows` would've been between `SubBytes` and `MixColumns`. We've pushed `ShiftRows` to the front of each round and adjusted the the round key appropriately.

For the decomposition phase of the attack, we decompose two consecutive rounds into their component layers with an SAS cryptanalysis, assuming 8-bit S-boxes. The trailing S-boxes of the first round are composed with the leading S-boxes of the second round to form a middle S-box layer.

We now move to the disambiguation phase of the attack. The linear part of the remnant affine layers looks something like Figure 2, assuming we choose `MixColumns` to be the 'correct' affine layer (instead of combining `MixColumns` with the affine part of `SubBytes`). A trick from [1] can partially recover the left and right parasites of each 32×32 block along the diagonal. The trick consists of observing that these blocks are equal to

$$\begin{pmatrix} b_1 & & & \\ & b_2 & & \\ & & b_3 & \\ & & & b_4 \end{pmatrix} \circ \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \circ \begin{pmatrix} a_1 & & & \\ & a_2 & & \\ & & a_3 & \\ & & & a_4 \end{pmatrix}$$

where the parasites a_i and b_i are random, invertible $8 \times$

[1]The affine layer of `SubBytes` only.

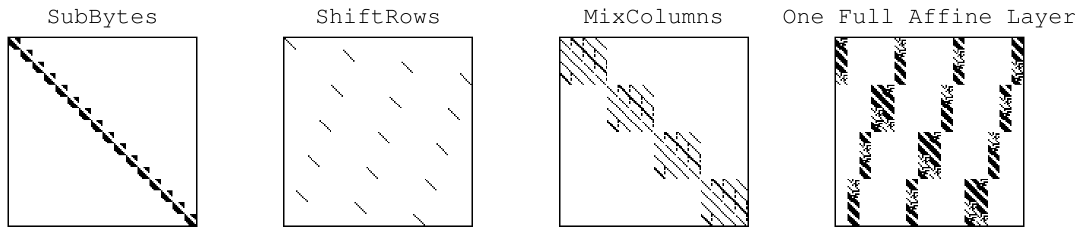

Figure 1: The linear part of AES's subroutines and the linear part of their composition. They're shown as 128×128 $\mathbb{F}_2$-matrices, where a white pixel is a zero and a black pixel is a one. For reference: the `ShiftRows` matrix is sixteen 8×8 identities, shuffled.

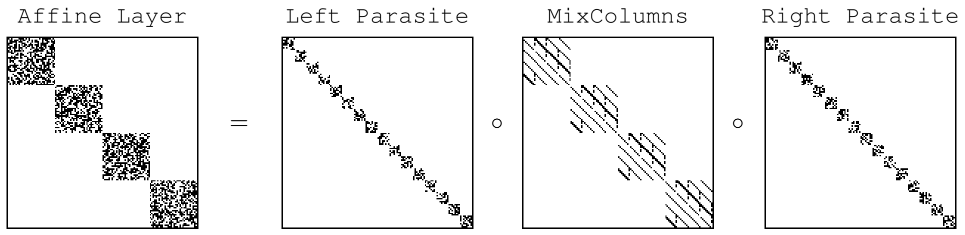

Figure 2: The affine layer recovered from the decomposition phase of attacking Chow et al.'s white-box AES construction and its proper disambiguation.

8 matrices, making each 8×8 sub-block of the composed matrix of the form $b_i \circ \mathtt{u} \circ a_j$. From this, we can compute

$$(b_n \circ \mathtt{v} \circ a_i) \circ (b_m \circ \mathtt{w} \circ a_i)^{-1} \circ (b_m \circ \mathtt{x} \circ a_j) \circ (b_n \circ \mathtt{y} \circ a_j)^{-1}$$
$$= b_n \circ \mathtt{z} \circ b_n^{-1} = L$$

as long as $i \neq j$ and $n \neq m$ (a slight modification will find an L for a given a_i). The equation $L \circ b'_n = b'_n \circ \mathtt{z}$ will give a singular system of 64 linear equations in 64 unknowns. We then derive a basis of the system's nullspace and search it for an invertible solution b'_n which will be correct up to an unknown scalar in each 32×32 block along the diagonal. For both affine layers, we push their constant part and the recovered parasites into the S-box layers on either side.

Both affine layers are now correct; the S-box layers are determined up to P_i, Q_{i+1}, 8 unknown scalars from $\mathbb{F}_{2^8}$, and two 128-bit constants. The j^{th} S-box in the middle layer equals $a_{\lfloor j/4 \rfloor} \cdot \mathtt{SubByte}(c_{\lfloor j/4 \rfloor} \cdot x + d_j) + b_j$. These constants are unique and can be found either by brute force or with a variant of the affine equivalence algorithm. Once found, we remove them from the middle S-box layer and their counterparts on the first and last S-box layers.

Both affine layers and the middle S-box layer are correct; the first and last S-box layers are determined up to P_i, Q_{i+1}, and two 128-bit constants. The j^{th} S-box in the first layer equals $k'_j + \mathtt{SubByte}(P_{i,j}(x))$. To verify a guess for k'_j, we recall that the $P_{i,j}$ also have an AS structure, with two 4-bit S-boxes and an 8×8 affine layer. Therefore, it must always satisfy

$$\bigoplus_{x}^{\{0,1\}^4} P_{i,j}(0^4 \mid\mid x) \stackrel{?}{=} 0 \quad \text{and} \quad \bigoplus_{x}^{\{0,1\}^4} P_{i,j}(x \mid\mid 0^4) \stackrel{?}{=} 0$$

because the $\oplus$ of the codebook of any 4-bit bijection must be zero and the outer affine layer does not affect this property. If stripping the S-box of k'_j and `SubByte` results in a function which satisfies this relation, we consider k'_j a solution. This is just a heuristic, but it is reasonably efficient and accurate enough to uniquely identify a round key.

Remark 1. *Given only an ensemble of rounds $\{R_i\}_{i \in I}$ such that there are no consecutive rounds ($i \in I \implies i+1 \notin I$), no adversary can recover the key.*

Remark 2. *Our implementation takes around a second to generate an instance of this construction (800KB in size) and less than thirty seconds to recover the master key from an instance, running on consumer hardware.*

6. CRYPTANALYSIS OF XIAO AND LAI

For each round, the construction of Xiao and Lai produces a network which computes

$$R_i := Q_i \circ \mathtt{MixColumns} \circ \mathtt{SubBytes} \circ \mathtt{AddRoundKey}_i \circ P_i$$

for $0 \leq i \leq 10$, where P_i and Q_i are 128×128 invertible matrices, P_i with random 16×16 blocks along the diagonal and Q_i with random 32×32 blocks along the diagonal. Each round also has a matrix $M_i = P_i^{-1} \circ \mathtt{ShiftRows} \circ Q_{i-1}^{-1}$ which simultaneously re-encodes the state array and permutes it. Once again, the round structure is slightly out of order–this is for the same reason as before.

The structure of the tables for each round is ASA. $Q_i \circ \mathtt{MixColumns}$ is the trailing affine layer, $\mathtt{AddRoundKey}_i \circ P_i$ is the leading affine layer, and `SubBytes` is the S-box layer. We can apply a variant of the previous attack–that is, decompose two consecutive rounds and disambiguate the space between them. There is a more interesting way, though:

The decomposition phase of the attack splits one round into its component layers, $R_i = A_i^2 \circ S_i \circ A_i^1$.

We now move to the disambiguation phase of the attack. A_i^2 should be exactly linear, as it is in the canonical decomposition – if the structural cryptanalysis leaves A_i^2 with a non-zero constant part, we merge into S_i. The structural cryptanalysis may have also permuted the S-boxes unnecessarily; we can undo most of this by permuting S-boxes until A_i^1 has 16×16 blocks along the diagonal and A_i^2 has 32×32 blocks along the diagonal.

Each S-box is equal to $b^{-1} \circ \texttt{SubByte} \circ a$, where a is an 8×8 affine transformation, and b is an 8×8 linear transformation. We can whiten SubByte by redefining it as $\texttt{SubByte}' = \texttt{SubByte} \circ \oplus_{52}$ so that $\texttt{SubByte}'(00) = 00$. We whiten the S-boxes similarly, by finding the input that maps to zero and setting it as an $\oplus$-mask on the input. This makes each S-box equal to $d^{-1} \circ \texttt{SubByte}' \circ c$, where c and d are 8×8 linear transformations. We use the linear equivalence algorithm from [3] to derive the set of 8 candidate solutions for c, d (which is much more efficient than applying the affine equivalence algorithm of the same paper to $b^{-1} \circ \texttt{SubByte} \circ a$). However, the large random blocks of Q_i will hide which of the 8 solutions is correct, so we need to redefine A_i^2,

$$A_i^{2\prime} := M_{i+1} \circ A_i^2 = P_{i+1}^{-1} \circ \texttt{ShiftRows} \circ \texttt{MixColumns} \circ \ldots$$

so that the smaller blocks of P_{i+1}^{-1} (spread out by ShiftRows) will leak the information we need; see Figure 3.

Once the correct c, d for each S-box has been recovered and moved into the affine layers, the construction will be completely disambiguated. The round key is the constant part of A_i^1.

Remark 3. *Given only an ensemble of rounds $\{R_i\}_{i \in I}$ and an ensemble of ShiftRows matrices $\{M_j\}_{j \in J}$ such that $j \in J \implies j - 1 \notin I$, no adversary can recover the key.*

Remark 4. *Our implementation takes less than a minute to generate an instance of this construction (21MB in size) and less than a minute to recover the master key from an instance, running on consumer hardware.*

7. NEW CONSTRUCTIONS

Structural cryptanalysis has been very successful at decomposing small SPN structures, and in general it doesn't seem possible to represent an arbitrary SPN structure as anything other than an SPN. For example, even the large ASASA white-box described in [2] suffers from very efficient decomposition attacks. A natural research direction, then, is to explore how to hide a white-box's canonical secret key inside a functionally-equivalent SPN representation where the problem of *disambiguation* is designed to be hard, rather than *decomposition*.

8. TOY CONSTRUCTION

As an initial attempt in the new design space, we describe a white-box construction which is an 'optimally hard' instance of the disambiguation problem for SPNs computing AES encryption. We then show that this construction is as weak as any of its predecessors.

Take an SPN computing AES encryption as input. We choose to leave the S-box layer of the SPN as the function $\zeta(x) = x^{-1} \in \mathbb{F}_{2^8}$ (the nonlinear part of SubBytes) concatenated sixteen times. There is no gain in generating S-boxes which are affine-equivalent to ζ and compensating for the error in the affine layer of the SPN. The algorithms of [3] can simply be used to convert them back, and the affine-equivalence will be recovered up to the group of ζ. This implies that our white-box generation algorithm must consist of teleporting information from one affine layer to the next, in the form of self-equivalences of the S-box layer.

In the same paper, Biryukov et al. find that the affine group of ζ contains 2,040 elements and derive a closed-form representation of them:

$$\zeta \circ ([c] \circ Q^i) = ([c^{-1}] \circ Q^{-i}) \circ \zeta$$

where $c \in \mathbb{F}_{2^8}^*$ is an arbitrary constant and Q^i is the Frobenius endomorphism (squaring) applied $0 \leq i < 8$ times[2].

For each S-box layer S_i of the SPN, sample a 128×128 binary matrix P which permutes the bytes of its input, and sixteen pairs $(c, i),\ (d, j),\ (e, k),\ \ldots \in \mathbb{F}_{2^8}^* \times \mathbb{Z}_8$. Set

$$a := P \circ \text{diagonal}([c] \circ Q^i,\ [d] \circ Q^j,\ [e] \circ Q^k,\ \ldots)$$
$$b := P \circ \text{diagonal}([c^{-1}] \circ Q^{-i},\ [d^{-1}] \circ Q^{-j},\ [e^{-1}] \circ Q^{-k},\ \ldots)$$

Because (a, b) is a self-equivalence of S_i, we get a non-trivial cancellation law $S_i = b^{-1} \circ S_i \circ a$ for linear functions a, b. Recalling that the SPN is written as $A_{11} \circ \ldots \circ A_2 \circ S_1 \circ A_1$, we redefine $A_{i+i} := A_{i+1} \circ b^{-1}$ and $A_i := a \circ A_i$.

After doing this to all S-box layers, the overall effect is that the S-box layers are left unaltered while the affine layers have their input and output mixed (which includes mixing of the key material, in the constant part of the affine layers):

$$(A_{11} \circ b_{10}^{-1}) \circ S_{10} \circ \ldots \circ (a_2 \circ A_2 \circ b_1^{-1}) \circ S_1 \circ (a_1 \circ A_1)$$

To prevent a trivial key-recovery attack, we set b_0^{-1} and a_{11} to be random affine transformations. The set of affine layers $\{a_i \circ A_i \circ b_{i-1}^{-1}\}_{i=1,\ldots,11}$ is the public output of our white-box generation algorithm and b_0^{-1} and a_{11} are the private outputs. This idea was originally described in [3] as a method for generating dual Rijndaels.

8.1 Cryptanalysis of the Toy Construction

First, we need to identify and remove the self-equivalences of ζ from an affine layer. The affine layers of AES have only four distinct 8×8 blocks, written in closed-form as $[c] \circ \texttt{SubBytes}^1$ for $c \in \{00, 01, 02, 03\} \subset \mathbb{F}_{2^8}$. However, the affine layers of the toy construction can have a large number of distinct 8×8 blocks:

$$[b] \circ Q^j \circ \texttt{SubBytes}^1 \circ Q^i \circ [a]$$

for any $a, b \in \mathbb{F}_{2^8}$ and $0 \leq i, j < 8$. The values a, b, i, j are uniquely determined and can be quickly found by brute-force. Even though $b = b'c$ masks the correct value of $c \in \{00, 01, 02, 03\}$ for a block, if we can find the value of a in the subsequent affine layer then this will be equal to b'.

Once the self-equivalences of ζ are gone, the 8×8 blocks of the affine layer will still be permuted arbitrarily. We fix this with a simple bounded search algorithm, which leaves the linear part of the affine layer the same as in Figure 1.

What we've described so far is a specialized way of solving the linear equivalence problem for matrices, as studied in [9]. This is not enough though, because the constant part of the recovered affine layer is not going to be the round key. The affine layer of AES has a certain group under permutation, and the constant part of the affine layer will be the round key permuted by a random element of this group.

Say that we have been disambiguating the ith affine layer of the white-box, $A_i' = p_i^{-1} \circ A_i \circ p_{i-1}$, where we've manipulated the original parasites a_i, b_{i-1}^{-1} into an element (p_{i-1}, p_i)

[2]Since $Q^8 = I$, $Q^{-i} = Q^{8-i}$.

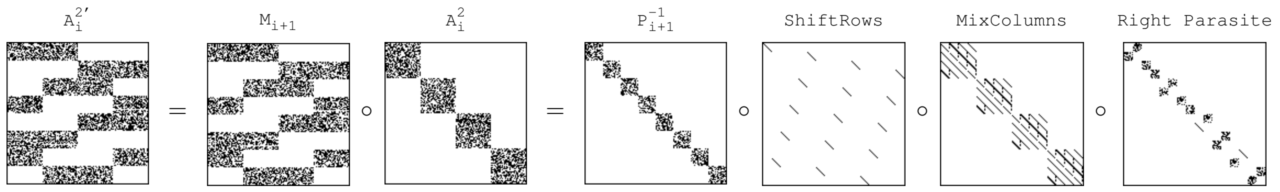

Figure 3: An affine layer encountered in the cryptanalysis of Xiao and Lai's construction and its proper dismabiguation. The right parasite transposes neighboring S-boxes randomly; its 8×8 blocks are random self-equivalences of the S-boxes.

of the permutation group of A_i. The constant part of A_i' is $p_i^{-1}(k_i \oplus d)$ where k_i is the i^{th} round key and d is a known constant, and the constant part of $A_{i+1}' = a_{i+1} \circ A_{i+1} \circ p_i$ is $a_{i+1}(k_{i+1} \oplus d)$.

By guessing p_i, we define k_i as well as a_{i+1}, and therefore k_{i+1}. This guess is considered consistent if k_{i+1} is the result of applying a key schedule to k_i. This determines the correct pair (k_i, k_{i+1}) and therefore the correct master key k_0. The authors are currently unaware of a faster way to achieve this.

The permutation group of AES's affine layer has $4^4 \times 4! \approx 2^{12}$ elements which need to be considered, corresponding to a cyclic rotation of each 8×8 block of `MixColumns` and any permutation of the four 32×32 blocks along the diagonal.

Remark 5. *Our implementation takes less than a second to generate an instance of this construction (23KB in size) and less than a minute to recover the master key from an instance, running on consumer hardware.*

9. DECOMPOSITION OF ζ

The failure of the toy construction shows that the common SPN representation of AES is not at all amenable to white-box representations. To get around this, we propose to replace the S-box layer with a decomposition, or a (possibly multi-layer) SPN with smaller S-boxes which computes the same function as the original S-box layer. This preserves functionality and the smaller, simpler S-boxes will necessarily have richer affine groups.

We can then go through each S-box layer and rewrite the affine layers on either side with random self-equivalences as we did in the toy construction. Once this is done, it doesn't seem easy to convert the S-boxes back to their original form because they are more heavily mixed with the neighboring affine layers and even their nearest S-boxes. One would also hope that the richer affine group would mask the algebraic structure of the affine layer which was useful in cryptanalyzing the toy construction.

We are aware of only one suitable decomposition, published by Canright [6]. It is motivated as follows. Elements of the field $\mathbb{F}_{2^8}$ are typically represented as seventh-degree polynomials modulo an eight-degree polynomial, with coefficients from $\mathbb{F}_2$. However, they can also be represented as first-degree polynomials modulo a second-degree polynomial, with coefficients from $\mathbb{F}_{2^4}$.

More generally, if we take an element of $c \in \mathbb{F}_{2^n}$ with a polynomial basis defined by the irreducible $f(x) = x^2 + \tau x + \nu$ with $\tau, \nu \in \mathbb{F}_{2^{n/2}}$, then c is written as $c_1 x + c_0$. The inverse of c is given by

$$(c_1 x + c_0)^{-1} = [\theta^{-1} c_1]x + [\theta^{-1}(c_0 + c_1 \tau)]$$
$$\text{where } \theta = c_1^2 \nu + c_1 c_0 \tau + c_0^2$$

This can be represented as an SPN with (non-invertible) affine layers computing multiplications by constants and additions, and (non-invertible) S-box layers computing multiplications between two variables and inversions. We decompose inversion in $\mathbb{F}_{2^8}$ recursively until all S-boxes are either identities or multiplication by elements of $\mathbb{F}_2$, i.e. `AND` gates. The final result is an SPN with five A layers and four S layers.

Affine Group.

We want the equation $\texttt{AND} \circ a = b \circ \texttt{AND}$ to hold when a, b are affine transformations. b must always be the 1×1 identity. The linear part of a can be any 2×2 invertible matrix. The constant part of a must be a 1-bit at the i^{th} position if the i^{th} row of the linear part has two 1-bits and 0 otherwise. This implies the following equations and their commuted versions:

$$xy = xy \qquad (x + y + 1)y = xy \qquad x(x + y + 1) = xy$$

.

Remark 6. *Our implementation takes less than a second to generate an instance of this construction (100KB in size), running on consumer hardware.*

9.1 Security Against Various Attacks

Decomposition attacks – including differential and linear cryptanalysis – don't seem to be applicable to the cryptanalysis of this construction because they can only output another instance of the dismabiguation problem. It should almost always be easier to directly inspect the affine layers.

Along these lines, the linear equivalence algorithm for matrices presented by Michiels et al. in [9] only constitutes a partial cryptanalysis. This algorithm quickly exhausts the amount of information that can be learned by inspecting individual affine layers but doesn't allow round keys to be recovered because the *self-equivalences* of the affine layer continue to mask it.

Lastly, we note that the addition of random affine transformations b_0^{-1} and a_{11} protect the white-box against simple attacks based on its structure as well as against the DCA and DFA attacks of [5]. To successfully attack a white-box with DCA or DFA, the attacker must know the 'real' input or output of the white-box, not the masked values. This ends up being a significant constraint in practice, because it means that ciphertexts must be encoded (and plaintexts decoded) somewhere the the adversary does not control and he also can't have partial information about the plaintexts and ciphertexts that might allow him to learn a_{11} and b_0^{-1}. A viable remaining application of white-boxes is key distribution architecture, where keys are generated uniform randomly and wrapped by a trusted party and distributed/unwrapped throughout an untrusted network. [10]

10. CONCLUSION

We presented two new attacks on past white-box AES constructions, which we believe are conceptually simple and give actionable insight into the proper design of white-box constructions. This leads us to propose one construction which is easily broken, and another whose security level is unknown. Further research may look for more fruitful decompositions of the S-box layer or ciphers such as BES, which contain AES but have more symmetry.

11. REFERENCES

[1] O. Billet, H. Gilbert, and C. Ech-Chatbi. Cryptanalysis of a white box AES implementation. In H. Handschuh and M. Hasan, editors, *Selected Areas in Cryptography*, volume 3357 of *Lecture Notes in Computer Science*, pages 227–240. Springer Berlin Heidelberg, 2005.

[2] A. Biryukov, C. Bouillaguet, and D. Khovratovich. *Cryptographic Schemes Based on the ASASA Structure: Black-Box, White-Box, and Public-Key (Extended Abstract)*, pages 63–84. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[3] A. Biryukov, C. De Cannière, A. Braeken, and B. Preneel. *A Toolbox for Cryptanalysis: Linear and Affine Equivalence Algorithms*, pages 33–50. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[4] A. Biryukov and A. Shamir. Structural cryptanalysis of SASAS. *Journal of Cryptology*, 23(4):505–518, 2010.

[5] J. W. Bos, C. Hubain, W. Michiels, and P. Teuwen. Differential computation analysis: Hiding your white-box designs is not enough. Cryptology ePrint Archive, Report 2015/753, 2015. http://eprint.iacr.org/2015/753.

[6] D. Canright. A very compact S-box for AES. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 441–455. Springer, 2005.

[7] S. Chow, P. Eisen, H. Johnson, and P. C. Van Oorschot. White-box cryptography and an AES implementation. In K. Nyberg and H. Heys, editors, *Selected Areas in Cryptography*, volume 2595 of *Lecture Notes in Computer Science*, pages 250–270. Springer Berlin Heidelberg, 2003.

[8] C. Lorens. Invertible boolean functions. *Electronic Computers, IEEE Transactions on*, EC-13(5):529–541, Oct 1964.

[9] W. Michiels, P. Gorissen, and H. D. Hollmann. Cryptanalysis of a generic class of white-box implementations. In R. M. Avanzi, L. Keliher, and F. Sica, editors, *Selected Areas in Cryptography*, pages 414–428. Springer-Verlag, Berlin, Heidelberg, 2009.

[10] P. Teuwen. private communication, 2016.

[11] Y. Xiao and X. Lai. A secure implementation of white-box AES, 2009.