Masaryk University
Faculty of Informatics

Mobile cryptography

Diploma thesis

Dušan Klinec

Brno, 2013

Declaration

Hereby I declare, that this paper is my original authorial work, which I have worked out by my own. All sources, references and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Dušan Klinec

Advisor: RNDr. Petr Švenda, Ph.D.

Acknowledgement

Thanks here

Abstract

Abstract here

Keywords

white box attack resistant cryptography, look up tables form, AES

Table of contents

1 **Introduction** 2

2 **Area overview** 3

2.1 *Overview* 3

2.2 *Mobile cryptography* 3

2.3 *Homomorphic encryption* 3

3 **Whitebox cryptography** 4

3.1 *Introduction* 4

3.2 *History* 5

3.3 *Description of schemes* 5

3.3.1 Cipher invertibility 5

4 **WBCAR AES using dual ciphers** 7

4.1 *Scheme* 7

4.1.1 Generic AES 7

4.1.2 AES duality 7

4.1.3 Generic AES duality 8

4.1.4 Constructing Dual AES 8

4.1.5 Whitebox AES 10

4.1.6 Whitebox Dual AES 12

4.2 *Implementation of the cipher* 12

4.3 *Results* 12

4.4 *Attack* 12

4.5 *Attacking Dual AES scheme* 14

4.6 *Implementation of the attack* 16

4.7 *Attack results* 16

5 **Scheme improvement** 17

5.1 *Twofish S-boxes* 18

5.2 *Key schedule* 18

5.3 *Key bytes for S boxes* 20

5.4 *Diffusion layer modification* 20

5.5 *Analysis* 21

5.6 *Analysis of diffusion layer* 24

5.7 *Discussion* 24

6 **Future work** 25

A **Appendix A** 26

A.0.1 Affinity check 26

Bibliography 26

1 Introduction

Introduction here

2 Area overview

2.1 Overview

Overview, setting picture in cryptographic world

2.2 Mobile cryptography

Motivation for white box cryptography

- computation with encrypted data
- computation with encrypted function

2.3 Homomorphic encryption

Homomorphic encryption follows computation with encrypted data function. Motivation: cloud computation. Short history, recent state of the art...

1. security point of view - optimal
2. short description, computing with encrypted data - use slides from OwnTalk
3. practical usability
4. state of the art practical results

3 Whitebox cryptography

3.1 Introduction

In this part of cryptography we are studying cryptographic algorithms with a much stronger attacker model, saying it is executed in a whitebox context.

Whitebox context (also abbreviated as WBC) is itself defined by the attacker model, which was introduced by Chow *et al.* [1] in 2002. WBC attacker has full control over execution of particular algorithm. Namely attacker has the following abilities:

- can observe execution:
  - access to instructions processing at the moment of computation
  - trace algorithm flow
  - sees memory used
- controls execution environment - runtime modification:
  - tamper program memory
  - execute only specified part of the algorithm (one round of the cipher)
  - modify if-conditions
  - change cycle counters
  - fault induction

In contrast to *blackbox context* (also abreviated as BBC), standard cryptographic model, that has only access to output of the cryptographic algorithm. In BBC the cryptographic algorithm is considered as an oracle/blackbox performing some function (analogy to executing algorithm in secure environment). Depending on finer granularity of attacker model, one can have access only to algorithm output (ciphertext), or attacker can also query oracle (chosen plain-text attack) and so on, but has no access to computation itself.

Cryptographic algorithms (we are mainly interested in symmetric ciphers in this work) were extensively studied for attacks in BBC, they were originaly designed to resist attacks considering only BBC. But if the context is wrong, it can be possible entry point for an attacker. Typical example is DRM [1], where software of a vendor (representing the right owner) is executed in potentially hostile environment, where user can have motivation to extract protected content without restrictions added by DRM software. In this situation we cannot consider DRM software to be executed in BBC.

1. Digital rights management, <`http://en.wikipedia.org/wiki/Digital_rights_management`

Let's take some symmetric block cipher as an another example. Usually it is constructed as a keyed permutation (round function) that is repeated several times to add randomness and to improve statistical results of the cipher, increasing security. But if we can inspect such execution, it is very easy to extract encryption keys, since we can read memory during execution or trace algorithm flow.

One such whitebox attack is *Key Whitening Attack* [2]. Key whitening is technique intended to increase the security of the iterated block cipher. It is typically implemented as adding a key material to the data (usually by simple operation, such as XOR) in the first and the last round. Such key whitening uses Twofish [3] and in modified version (only adding the key material in the last round) also AES [4]. In Key Whitening Attacking is modified cipher binary (we are in whitebox context) in such a way that the output of the cipher will be the key material itself.

The definition of whitebox cryptography could be: "The challenge that white-box cryptography aims to address is to implement a cryptographic algorithm in software in such a way that cryptographic assets remain secure even when subject to white-box attacks. Software implementations that resist such white-box attacks are denoted white-box implementations." [5].

3.2 History

History overview, oorschot, billet, imposibility of obfuscation, generic attacks on AES, DES

3.3 Description of schemes

Schemes used with AES, DES. Some techniques used in whiteboxing the cipher (input output encodings, mixing bijections - diffusion layers).

3.3.1 Cipher invertibility

One of the requirements on whitebox cipher implementation is usually a *non-invertibility.* It means that given a encryption part of the cipher with embedded key one should not be able to use it also for decryption and vice versa. This property is especially useful if one want's to use symmetric cipher to simulate an asymmetric. But it is important to realize that this goal is difficult to achieve in whitebox context.

As an example take AES whitebox implementation. Inverting cipher is blackbox context is rather computationally difficult. Using brute-force one would need 2^{128} operations to invert the cipher. The whitebox context is in contrast to blackbox rather in advantage. One of the problems here is that ShiftRows operation can be very easily canceled in whitebox context and that attacker can execute only particular round of the cipher. I propose some improvement addressing this problem in section 5.4.

There are 4 columns of state array within one round independent on each other. Thus cipher can be easily inverted running the cipher backwards and finding inversion for each column separately. Thus the main task is to find inversion of 32-bit wide function representing one round of the cipher on one column of the state array by running through the space GF $\left(2^8\right)^4$, evaluating the round function and comparing with wanted result.

Computational complexity to invert the cipher is $\underbrace{10}_{\text{rounds}} \cdot \underbrace{4}_{\text{columns}} \cdot \underbrace{2^{32}}_{\text{column function space}}$ operations.

One can also pre-compute tables for inverted cipher, that would occupy $10 \cdot 4 \cdot (2^{32} \cdot 4)$ B $\approx$ 69 GB. We have implemented inverting WB AES, in non-optimized version it takes 13 hours on my hardware to invert WB AES with negligible memory requirements.

4 WBCAR AES using dual ciphers

WBCAR stands for whitebox context attack resistant, meaning cipher implementation should resist attacks like key-extraction, cipher-invertion and others against attacker in whitebox context.

In this chapter I describe whitebox scheme proposed in [6] that make use of AES dual ciphers. It is supposed that using dual AES, different in each round, will increase security of whitebox implementation of the cipher. Paper says that this modification results in raising Billet's attack complexity to 2^{91} computational steps, making it unfeasible to perform it in practice.

4.1 Scheme

In defining paper [6] is not explained how to obtain dual AES ciphers and how to construct mapping from one to another. This is important part since it plays crucial role in proof that this scheme is vulnerable. At first is described generalization of AES and how to construct mappings between them.

4.1.1 Generic AES

It is possible to generalize AES by changing its irreducible polynomial and generator to obtain generic form of AES.

Generic AES can be represented as a $\{R(x), \beta\}$, where $R(x) \in (\mathbb{Z}/p\mathbb{Z})[x]$ is irreducible polynomial of degree 8, $\beta \in GF(2^8)$ is a generator of the field $GF(2^8)$.

Default AES (as in NIST standard) in this notation is represented as $\{\{11B\}_x, \{03\}_x\}$. Polynomial is expressed in hexadecimal notation, each bit corresponds to polynomial coefficient, LSB corresponds to constant term.
Thus $0x11B_{16} = 1\ 0001\ 1011_2 \Rightarrow \{11B\}_x \sim x^8 + x^4 + x^3 + x + 1$.

It is known that there are 30 irreducible polynomials over GF(2^8). For each of them there are 8 possible generators that can be used to generate field and to preserve duality mentioned in the next section.

4.1.2 AES duality

Definition 1. *Two ciphers E and E' are called Dual Ciphers, if they are isomorphic, i.e., if there exist invertible transformations $f()$, $g()$ and $h()$ such that*

$$\forall p, k : E_k(p) = f^{-1}\left(E'_{g(k)}(h(p))\right) \tag{4.1}$$

where p is the plaintext, and k is the secret key.

4.1.3 Generic AES duality

Let's assume we have some arbitrary generic AES $\{R(x), \beta\}$.

All elements of the field $\mathrm{GF}(2^8) = \{01, 02, \ldots, FF\}$ can be expressed in terms of the generator β, $\mathrm{GF}(2^8) = \{\beta^i \mid i \in [0, 254]\} = \{\beta^0, \beta^1, \ldots, \beta^{254}\}$.

We can then construct 8×8 matrix $\Delta = \begin{bmatrix}\beta^0 & \beta^{25} & \beta^{50} & \beta^{75} & \beta^{100} & \beta^{125} & \beta^{150} & \beta^{175}\end{bmatrix}$ where $\beta^i \in \mathrm{GF}(2^8) \cong \mathrm{GF}(2)^8$ is a column vector. Then Δ is a base change matrix:

$$\Delta : \{\{11B\}_x, \{03\}_x\} \longrightarrow \{R(x), \beta\} \tag{4.2a}$$

$$\Delta^{-1} : \{R(x), \beta\} \longrightarrow \{\{11B\}_x, \{03\}_x\} \tag{4.2b}$$

For default AES $\{\{11B\}_x, \{03\}_x\}$ holds

$$\begin{aligned}\Delta &= \begin{bmatrix}03^0 & 03^{25} & 03^{50} & 03^{75} & 03^{100} & 03^{125} & 03^{150} & 03^{175}\end{bmatrix}\\ &= \begin{bmatrix}01 & 02 & 04 & 08 & 16 & 32 & 64 & 128\end{bmatrix}\\ &= I_8\end{aligned}$$

as expected.

From this it is clear that following duality holds: $E \sim \{\{11B\}_x, \{03\}_x\},\ E' \sim \{R(x), \beta\}$ then:

$$\forall p, k : E_k(p) = \Delta^{-1}\left(E'_{\Delta(k)}(\Delta(p))\right) \tag{4.3}$$

4.1.4 Constructing Dual AES

We can construct arbitrary dual AES from default AES. Recall there are 4 operations used in single AES round: *ShiftRows*, *AddRoundKey*, *SubByte*, *MixColumn*.

Default AES At first recall two most important functions in AES round that we will then transform to generic form.

SubByte

$$\begin{aligned}S : \mathrm{GF}(2^8) &\longrightarrow \mathrm{GF}(2^8)\\ x &\longmapsto A \times x^{-1} \oplus c\end{aligned} \tag{4.4}$$

where x^{-1} is element inverse in $\mathrm{GF}(2^8)$, A is 8×8 matrix over $\mathrm{GF}(2)$, c is column vector $\mathrm{GF}(2)^8$. A, c are constants defined in NIST standard.

Equation for Sbox:

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \tag{4.5}$$

where $x_i, y_i \in \mathrm{GF}(2)$.

MixColumn

- columns considered as polynomials over $GF(2^8)$
- $p(x) \cdot c(x) \pmod{x^4 + 1}$
  where $c(x)$ is fixed polynomial $c(x) = 03x^3 + 01x^2 + 01x + 02$

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \tag{4.6}$$

where $x_i, y_i \in \mathrm{GF}(2^8)$.

Generic AES In the generic AES operations *ShiftRows*, *AddRoundKey* work same as in default AES, they are not affected by base change operation.

SubByte

$$\begin{aligned} S_{dual} : \mathrm{GF}(2^8) &\longrightarrow \mathrm{GF}(2^8) \\ x &\longmapsto \Delta \times A \times \Delta^{-1}\left(x^{-1}\right) \oplus \Delta\,(c) \end{aligned} \tag{4.7}$$

MixColumn MixColumn matrix coefficients are expressed in terms of generator $\beta = 03$.

$$\begin{bmatrix} \beta^{25} & \beta^1 & \beta^0 & \beta^0 \\ \beta^0 & \beta^{25} & \beta^1 & \beta^0 \\ \beta^0 & \beta^0 & \beta^{25} & \beta^1 \\ \beta^1 & \beta^0 & \beta^0 & \beta^{25} \end{bmatrix}$$

Round function - default AES We consider whole AES round as a single function R of a state array. Let's define

$$\begin{bmatrix} x_{0,0} & x_{0,1} & x_{0,2} & x_{0,3} \\ x_{1,0} & x_{1,1} & x_{1,2} & x_{1,3} \\ x_{2,0} & x_{2,1} & x_{2,2} & x_{2,3} \\ x_{3,0} & x_{3,1} & x_{3,2} & x_{3,3} \end{bmatrix} \xrightarrow{R} \begin{bmatrix} y_{0,0} & y_{0,1} & y_{0,2} & y_{0,3} \\ y_{1,0} & y_{1,1} & y_{1,2} & y_{1,3} \\ y_{2,0} & y_{2,1} & y_{2,2} & y_{2,3} \\ y_{3,0} & y_{3,1} & y_{3,2} & y_{3,3} \end{bmatrix}$$

From this we define $y_{i,j}$ as a function with 4 arguments from GF(2^8):

$$y_{i,j}\left(x_{i,0}, x_{i,1}, x_{i,2}, x_{i,3}\right) = \bigoplus_{l=0}^{3} \alpha_{l,j} \cdot S(x_{i,l} \oplus k_{i,l}) \tag{4.8}$$

where $\alpha_{k,j}$ is MixColumn matrix coefficient in k-th row and j-th column. We are abstracting here *ShiftRows* operation, it won't be needed for our further argumentation. To make it clear here are equations for the first column of state array:

$$y_{0,0}\left(x_{0,0}, x_{1,0}, x_{2,0}, x_{3,0}\right) = 02 \cdot T_{0,0}(x_{0,0}) \oplus 03 \cdot T_{1,0}(x_{1,0}) \oplus 01 \cdot T_{2,0}(x_{2,0}) \oplus 01 \cdot T_{3,0}(x_{3,0}) \tag{4.9a}$$

$$y_{1,0}\left(x_{0,0}, x_{1,0}, x_{2,0}, x_{3,0}\right) = 01 \cdot T_{0,0}(x_{0,0}) \oplus 02 \cdot T_{1,0}(x_{1,0}) \oplus 03 \cdot T_{2,0}(x_{2,0}) \oplus 01 \cdot T_{3,0}(x_{3,0}) \tag{4.9b}$$

$$y_{2,0}\left(x_{0,0}, x_{1,0}, x_{2,0}, x_{3,0}\right) = 01 \cdot T_{0,0}(x_{0,0}) \oplus 01 \cdot T_{1,0}(x_{1,0}) \oplus 02 \cdot T_{2,0}(x_{2,0}) \oplus 03 \cdot T_{3,0}(x_{3,0}) \tag{4.9c}$$

$$y_{3,0}\left(x_{0,0}, x_{1,0}, x_{2,0}, x_{3,0}\right) = 03 \cdot T_{0,0}(x_{0,0}) \oplus 01 \cdot T_{1,0}(x_{1,0}) \oplus 01 \cdot T_{2,0}(x_{2,0}) \oplus 02 \cdot T_{3,0}(x_{3,0}) \tag{4.9d}$$

where $T_{i,j}(x) = S\left(x \oplus k_{i,j}\right)$.

Round function - generic AES Using aforementioned generic form of *SubByte* and *MixColumn* functions we can define round function also for generic AES in the same way, using base change transformation. From this we define $y_{i,j}$ as a function with 4 arguments from GF(2^8):

$$y_{i,j}\left(x_{i,0}, x_{i,1}, x_{i,2}, x_{i,3}\right) = \bigoplus_{l=0}^{3} \Delta(\alpha_{l,j}) \cdot \left(\Delta \times A \times \Delta^{-1}\left(\left(x_{i,l} \oplus \Delta\left(k_{i,l}\right)\right)^{-1}\right) \oplus \Delta\left(c\right)\right) \tag{4.10}$$

4.1.5 Whitebox AES

Whitebox AES is AES implementation based on table lookups, functions used in AES are stored as look-up tables. Figure 4.1 shows one round of whitebox AES implementation using look-up tables, protected with whitebox techniques like Mixing Bijections and internal encodings.

On the diagram in figure 4.1 are following whitebox functions:

- MB stands for Mixing Bijection. It is 32×32 matrix over GF(2) representing linear transformation over GF(2). $\text{MB}^{-1}_{\{0,1,2,3\}}$ are then column stripes of corresponding MB

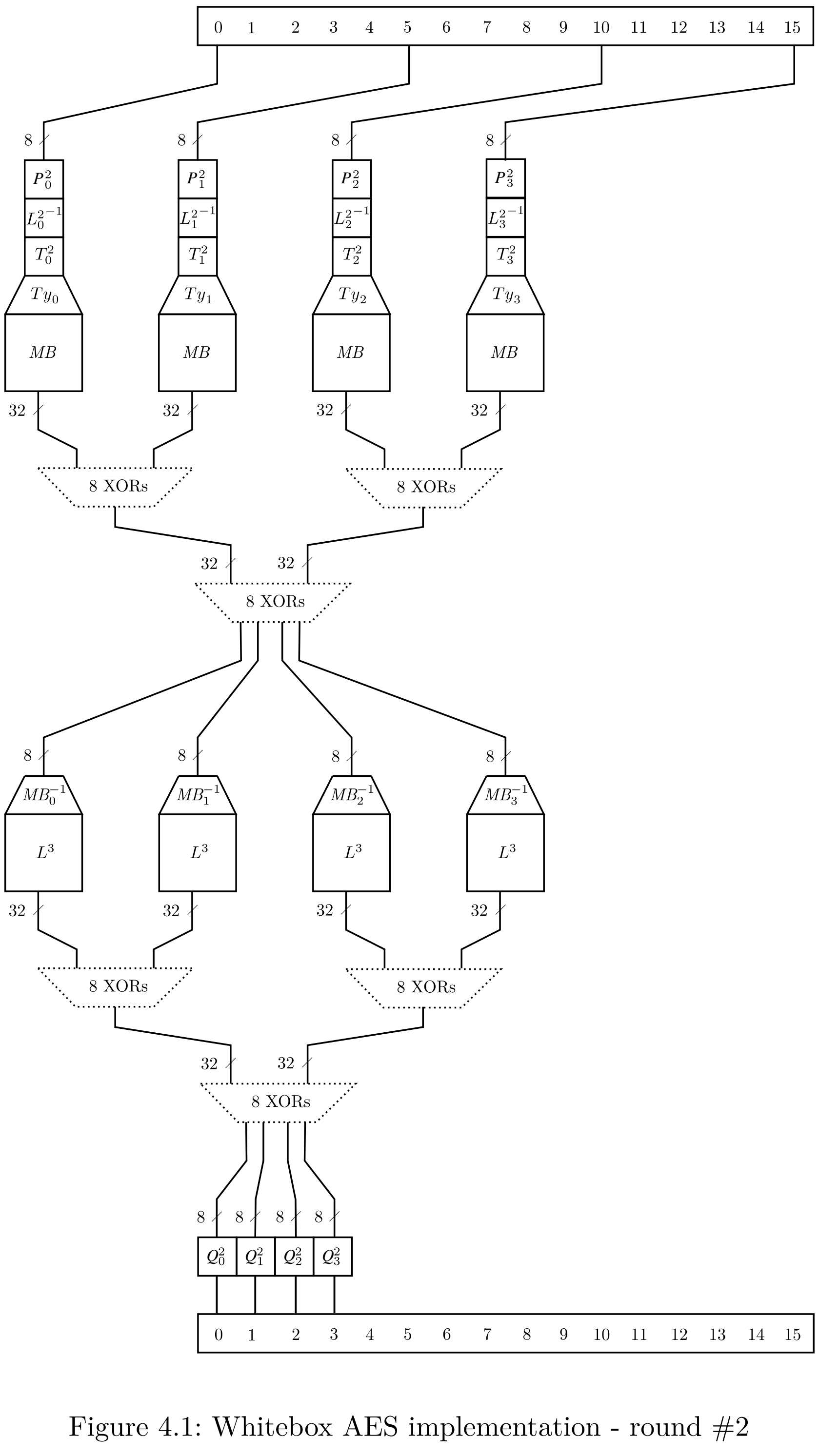

Figure 4.1: Whitebox AES implementation - round #2

inverse matrix - linearity is used here to perform multiplication with MB inverse matrix. This transformation is not interesting since it cancels out within one round. We are interested in round function R, this transformation has no effect on it.

- L stands also for Mixing Bijection but in this case it is 8×8 matrix over GF(2) representing linear transformation over GF(2). Original purpose of it was to protect output of round r connected to the input of round $r + 1$ in table representation.
- Q,P. These are random non-linear bijections in GF(2^8), called internal encodings. It holds that $P^{r+1}_{i,j} \circ Q^r_{i,j} = id$

Since transformation L, L^{-1} is performed byte-wise on state array, we can compose them with corresponding internal encodings bijections

$$Q^{r\,\prime}_{i,j} = Q^r_{i,j} \circ L^{r+1}_j \tag{4.11a}$$

$$P^{r\,\prime}_{i,j} = (L^r_j)^{-1} \circ P^r_{i,j} \tag{4.11b}$$

We again obtain non-linear random bijections with embedded L transformation in it, without loss of generality. This abstraction is done in Billet's attack.

4.1.6 Whitebox Dual AES

There was published a paper describing AES whitebox implementation with use of dual AES. It claimed that this implementation should be harder (in terms of time complexity) to break using Billet's attack on whitebox AES. On the figure 4.2 is scheme for one round, one column of state array, whitebox dual AES implementation for round 2. According to the original paper, in each column is used different generic AES. This implementation is compatible with default AES, so after computing in dual AES we have to transform the result to default AES with base change transformation Δ.

4.2 Implementation of the cipher

Cipher implementation description, generalization of oorschot design. Mixing bijections.

4.3 Results

Practical results for implementation, performance statistics, results.

4.4 Attack

My attack, see proof.tex

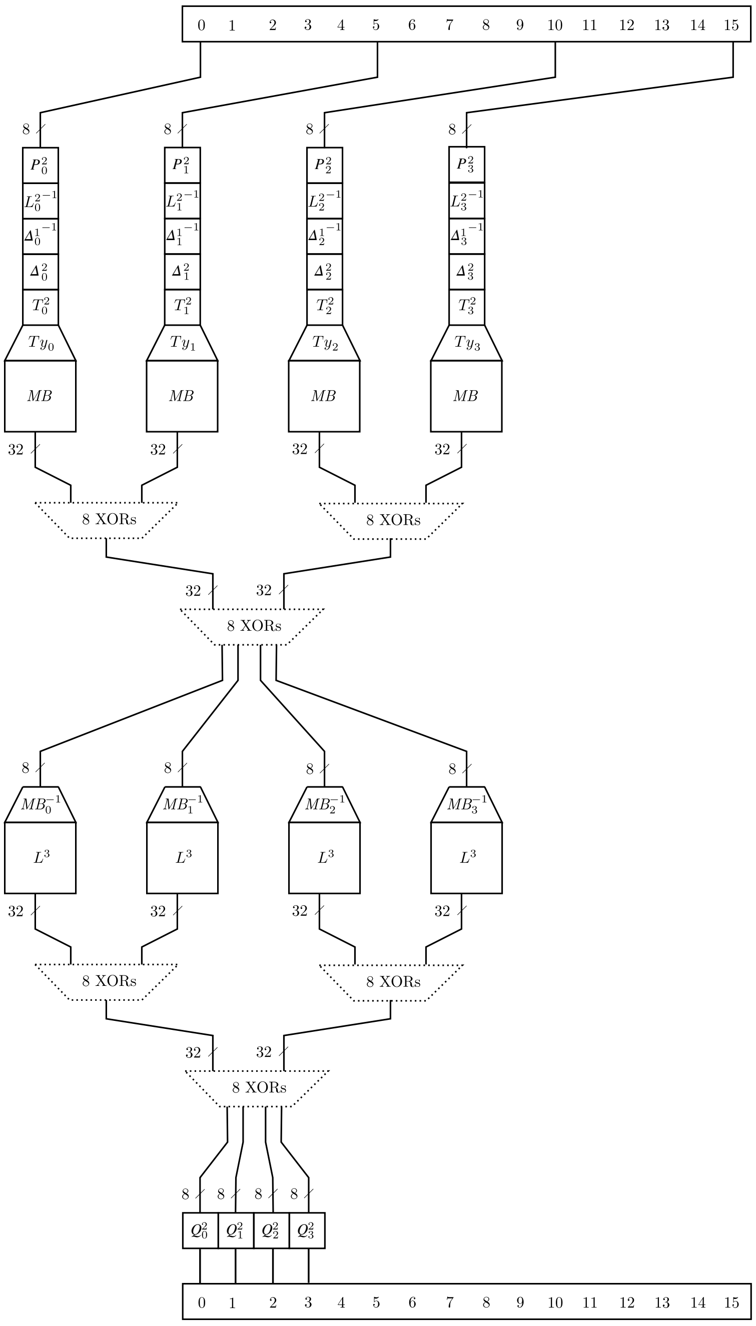

Figure 4.2: Whitebox Dual AES implementation - round #2

4.5 Attacking Dual AES scheme

According to [6] whitebox scheme using Dual AES is considered to be more difficult to crack with BGE attack and thus it is consider safer than original scheme proposed in [1]. But we show that it is not true. This result is new and was not published yet.

Proposition 1. *Whitebox Dual AES scheme can be broken with the Billet's attack with the same time complexity as Whitebox AES scheme.*

Proof. Let's define round function for whitebox AES and for whitebox dual AES and compare it.

Round function - whitebox AES

There are additional Q', P' functions, input and output bijections, for details see [1] [7].

$$y_{i,j}\left(x_{i,0}, x_{i,1}, x_{i,2}, x_{i,3}\right) = Q^{r\,\prime}_{i,j}\left(\bigoplus_{l=0}^{3} \alpha_{l,j} \cdot S\left(P^{r\,\prime}_{i,l}\left(x_{i,l}\right)\right)\right) \tag{4.12a}$$

$$= Q^{r\,\prime}_{i,j}\left(\bigoplus_{l=0}^{3} \alpha_{l,j} \cdot \left(A\left(\left(P^{r\,\prime}_{i,l}\left(x_{i,l}\right) \oplus k_{i,l}\right)^{-1}\right) \oplus c\right)\right) \tag{4.12b}$$

$$= Q^{r\,\prime}_{i,j} \circ R_{i,j}\left(x_{i,0}, x_{i,1}, x_{i,2}, x_{i,3}\right) \tag{4.12c}$$

Round function - whitebox dual AES

$$y_{i,j}\left(x_{i,0}, x_{i,1}, x_{i,2}, x_{i,3}\right) = Q^{r\,\prime}_{i,j}\left(\bigoplus_{l=0}^{3} \Delta\left(\alpha_{l,j}\right) \cdot \left(\Delta \times A \times \Delta^{-1}\left(\left(P^{r\,\prime}_{i,l}\left(x_{i,l}\right) \oplus \Delta\left(k_{i,l}\right)\right)^{-1}\right) \oplus \Delta\left(c\right)\right)\right) \tag{4.13a}$$

$$= Q^{r\,\prime}_{i,j}\left(\bigoplus_{l=0}^{3} \Delta\left(\alpha_{l,j} \cdot \left(A \times \Delta^{-1}\left(\left(P^{r\,\prime}_{i,l}\left(x_{i,l}\right) \oplus \Delta\left(k_{i,l}\right)\right)^{-1}\right) \oplus c\right)\right)\right) \tag{4.13b}$$

Now it is easy to see whitebox dual AES correctness, moreover it is visible that the same attack breaking whitebox AES breaks whitebox dual AES scheme. From figure 4.2 observe how rounds are connected to each other with respect to dual AESes. It is clearly visible in the case of the first round, where the inverse base change matrix Δ^{-1} is identity, because there is no previous dual AES encoding transformation in first round.

Therefore the first round function is:

$$y_{i,j}\left(x_{i,0},x_{i,1},x_{i,2},x_{i,3}\right) = Q^{r\,\prime}_{i,j}\left(\bigoplus_{l=0}^{3}\Delta(\alpha_{l,j})\cdot\left(\Delta\times A\times\Delta^{-1}\left(\left(\Delta\circ P^{r\,\prime}_{i,l}\left(x_{i,l}\right)\oplus\Delta\left(k_{i,l}\right)\right)^{-1}\right)\oplus\Delta\left(c\right)\right)\right) \tag{4.14a}$$

$$= Q^{r\,\prime}_{i,j}\left(\bigoplus_{l=0}^{3}\Delta(\alpha_{l,j})\cdot\left(\Delta\times A\times\Delta^{-1}\left(\Delta\left(P^{r\,\prime}_{i,l}\left(x_{i,l}\right)\oplus k_{i,l}\right)^{-1}\right)\oplus\Delta\left(c\right)\right)\right) \tag{4.14b}$$

$$= Q^{r\,\prime}_{i,j}\left(\bigoplus_{l=0}^{3}\Delta(\alpha_{l,j})\cdot\left(\Delta\times A\times\left(\left(P^{r\,\prime}_{i,l}\left(x_{i,l}\right)\oplus k_{i,l}\right)^{-1}\right)\oplus\Delta\left(c\right)\right)\right) \tag{4.14c}$$

$$= Q^{r\,\prime}_{i,j}\circ\Delta\left(\bigoplus_{l=0}^{3}\alpha_{l,j}\cdot\left(A\times\left(\left(P^{r\,\prime}_{i,l}\left(x_{i,l}\right)\oplus k_{i,l}\right)^{-1}\right)\oplus c\right)\right) \tag{4.14d}$$

$$= Q^{r\,\prime}_{i,j}\circ\Delta\circ R_{i,j}\left(x_{i,0},x_{i,1},x_{i,2},x_{i,3}\right) \tag{4.14e}$$

Transformation from 4.14a to 4.14b is possible since it holds:

$$\forall x,y\in\mathrm{GF}(2^8)\ :\ y=x^{-1}\Rightarrow\Delta\left(y\right)=\Delta\left(x^{-1}\right) \tag{4.15}$$

due to properties of generator and base change matrix.

Now if we compare equations 4.12c and 4.14e, they are very similar, the only difference here is the application of base change matrix Δ.

Here we can do the similar thing we did in equations 4.11a, 4.11b where we composed two transformations, non-linear and linear to non-linear transformation, with equation 4.14e.

We can thus define:

$$Q^{r\,\prime\prime}_{i,j} = Q^{r\,\prime}_{i,j} = Q^{r}_{i,j}\circ\Delta = Q^{r}_{i,j}\circ L^{r+1}_{j}\circ\Delta \tag{4.16a}$$

$$y_{i,j}\left(x_{i,0},x_{i,1},x_{i,2},x_{i,3}\right) = Q^{r\,\prime}_{i,j}\circ\Delta\circ R_{i,j}\left(x_{i,0},x_{i,1},x_{i,2},x_{i,3}\right) \tag{4.16b}$$

$$y_{i,j}\left(x_{i,0},x_{i,1},x_{i,2},x_{i,3}\right) = Q^{r\,\prime\prime}_{i,j}\circ R_{i,j}\left(x_{i,0},x_{i,1},x_{i,2},x_{i,3}\right) \tag{4.16c}$$

Now it is evident that equations for whitebox AES 4.12c and 4.16c are the same, the only difference is only in non-linear transformations Q, but it is important they are both non-linear.

Conclusion is if attack can break whitebox AES scheme with round function 4.12c it can also break whitebox dual AES scheme. During the attack is transformation Q fully determined, we verified that if Dual AES scheme is used, transformation Q is the exact form as described above. □

4.6 Implementation of the attack

Attack implementation description

4.7 Attack results

Practical results for attack implementation, time to break.

5 Scheme improvement

The BGE attack strongly relies on publicly known constants and building blocks used in the AES cipher (MixColumn constants, fixed S-box). This leads to an idea of turning constant part of cipher into key dependent ones, according to Kerckhoffs's principle.

It should increase computational complexity of the attack since attacker would have to try all combinations of key dependent part of the cipher. In the ideal scenario the attack will be unfeasible due to high computational complexity.

As we know AES S-box is constant and has relatively simple algebraic form. In blackbox context, it is quite difficult to construct algebraic equations for whole AES (this was one of design criterias of an AES in order to prevent possible algebraic attacks), but BGE attack aims only on one round of the cipher and from this perspective it is quite easy to construct algebraic equations for 1 round - as we seen in BGE attack, what makes AES vulnerable to algebraic attacks in whitebox context.

In whitebox implementation of cipher we have two contrary goals - to minimize table size and to prevent attack in whitebox context. Table size is what puts quite limitations in implementation and on security boundaries. In one extreme case we would build look-up table for whole AES for every possible input of size $(2^{128} \cdot 16) > 10^{39}$ bytes. This scheme is no weaker than AES in blackbox context, so perfectly secure in whitebox context, but rather unfeasible in practice.

As we seen in BGE attack it is easy to turn random non-linear bijections (input/output encodings), protecting table contents, to affine transformations between rounds of cipher, so more complicated non-linear bijections are probably not the way out of this.

The main idea here is to break backward compatibility with AES (or any other well known cipher)- as it does not have proper structure for whitebox implementation, what is also visible from the fact there no non-broken whitebox scheme of AES exists nowadays [CITE HERE]. In literature was already proposed to design a new cipher with whitebox implementation issues in mind [7].

So we took inspiration from Twofish [3] cipher which has key dependent S-boxes with rather complicated algebraic representations. As I emphasized before, the key idea here is to make expressing one round of cipher as algebraic equations more difficult for an attacker. Our first scheme is to use Twofish key dependent S-boxes in AES algorithm.

5.1 Twofish S-boxes

Here observe Twofish S-boxes (from [3]) and their algebraic representation.

$$s_{0,k_0,k_1}(x) = q_1 [q_0 [q_0 [x] \oplus k_0] \oplus k_1] \tag{5.1a}$$
$$s_{1,k_2,k_3}(x) = q_0 [q_0 [q_1 [x] \oplus k_2] \oplus k_3] \tag{5.1b}$$
$$s_{2,k_4,k_5}(x) = q_1 [q_1 [q_0 [x] \oplus k_4] \oplus k_5] \tag{5.1c}$$
$$s_{3,k_6,k_7}(x) = q_0 [q_1 [q_1 [x] \oplus k_6] \oplus k_7] \tag{5.1d}$$

Where q_0, q_1 are fixed 8-bit permutations, $k_i,\ i \in [0, 7]$ are key bytes, $s_{j,k_a,k_b},\ j \in [0, 4]$ are resulting S-boxes.

Thus instead of fixed AES S-box we use Twofish key dependent S-boxes. In particular we use $s_{j,k_a,k_b},\ j \in [0, 4]$ instead of 4 the same constant S-boxes in computation of one column of state matrix - consistent approach with Twofish algorithm, in Twofish we have MDS as a diffusion element, here we have MixColumn operation [FIGURE HERE].

In blackbox context there is disadvantage for key dependent S-boxes since it takes some time to generate them, for each encryption key, but in whitebox context the whole cipher is generated before use, including S-boxes, so during encryption/decryption there is no such disadvantage anymore.

5.2 Key schedule

BGE attack also make use of reversible AES key schedule to obtain encryption key. It is only needed to obtain round keys for two consecutive rounds of cipher in order to obtain full encryption key.

In order to avoid this reversing we also modify key schedule. In particular we suggest to use hash-chains as round keys, so attacker would not be able to combine knowledge of two consecutive rounds as in BGE attack.

We suggest to use *bcrypt* [8] or *scrypt* [9] as a hash function for generating hash chains. The main reason is high time complexity needed to evaluate such hash functions. This makes eventual brute-forcing even harder, because of low hashes per second ratio. We could use for example also *SHA-256* hash function to generate hash chain, but nowadays there exists even special hardware for computing SHA digests (ASICS chips, for Bitcoin mining), with performance 1500 G hashes per second for one device [10], brute-forcing with such device would be much faster.

In [11] M. Gosney used cluster made of GPUs (general purpose hardware) and benchmarked hash functions from performance perspective, for details see table 5.1. *bcrypt* is by orders of magnitude slower than SHA1, almost by factor 10^6. This makes brute-force unfeasible on general purpose hardware.

function	hashes per second
SHA1	63 G/s
MD5	180 G/s
BCrypt	71 K/s

Table 5.1: Hash functions performance comparison

In AES-128 we have 128 bit cipher key, $k_0, \dots, k_{15}$. We define k_i^r to be round key byte $i \in [0, 15]$ used in round $r \in [0, 10]$.

We define hash function used further in our modified key schedule

$$hash\left(inp, salt\right)_{N_{bc}, N_{sha}} = bcrypt\left(N_{bc}, salt, \text{SHA-256}^{N_{sha}}\left(inp\right)\right) \tag{5.2}$$

Where we have 2 security parameters in this scheme. N_{bc} is work load for bcrypt, determines computation complexity of bcrypt hash function. N_{sha} is number of nested iterations of SHA-256 function.

With this we define key schedule for our new cipher:

$$k_i^r = \begin{cases} hash_{N_{bc}, N_{sha}}(key, salt)_i & \text{if } r = 0 \\ hash_{N_{bc}, N_{sha}}(k^{r-1} \mid\mid key, salt)_i & \text{otherwise} \end{cases} \tag{5.3}$$

Where

i	subscript on right side stands for i-th byte of resulting hash
key	is encryption key, 128 bits
k^{r-1}	is whole round key for round $r - 1$
$\mid\mid$	symbol is concatenation of two binary arguments
$salt$	is arbitrary 128 bit salt used in bcrypt algorithm. This can be publicly known - published together with ciphertext or in particular whitebox cipher instance.

Equation for k_i^r is chosen with two primary goals in mind, attacker is not able to:

1. derive encryption key from two (or more) consecutive round keys. This results from infeasability of reversing hash chain. We are also using computational intensive hash function so even brute-forcing is unfeasible.
2. derive round key for $r_1 - 1$ or $r_2 + 1$ if he already have round keys for rounds $[r_1, r_2]$. Unavailability of deriving round key for $r_1 - 1$ results from the previous argument, but here is also important that from already derived round keys we are not able to derive next ones (compared AES schedule case) since it also depends on encryption key directly.

5.3 Key bytes for S boxes

In order to increase strength of proposed scheme we don't use round key bytes for S-box computation directly. If someone succeeds in determining this round key bytes by computing proposition 3 from BGE attack for all key bytes possibilities it could help to derive the round keys.

From this reason we use completely different keys for key-dependent S-boxes that in rest of the cipher.

$$k_i^{r\,\prime} = \begin{cases} hash_{N_{bc},N_{sha}}(key \mid\mid \text{"magicConstant"}, salt)_i & \text{if } r = 0 \\ hash_{N_{bc},N_{sha}}(k^{r-1\,\prime} \mid\mid key \mid\mid \text{"magicConstant"}, salt)_i & \text{otherwise} \end{cases} \tag{5.4}$$

The equation 5.4 is the same as 5.3 with only difference of concatenation of "magicConstant". This makes two hash chains (1 for round keys, 1 for S-boxes) completely different and non-transformable one to another.

5.4 Diffusion layer modification

In section 3.3.1 was mentioned cipher invertibility. I suggest to extend input/output space of the round function from 32-bits to 128-bits, raising complexity of mentioned inverting attack to $10 \cdot 4 \cdot 2^{128}$ operations. In AES one byte of state array depends only on 4 bytes = one column of state array. Round function of AES acts independently on 4 columns, making it easy to invert it.

Proposed improvement is in changing MDS (Maximum Distance Separable) matrix from 4×4 to 16×16. Then would one byte of state array depend on 16 bytes, making round function 128-bit wide.

MDS matrix acts as diffusion element in the cipher, since our cipher is of type substitution-permutation cipher, our MDS matrix represents invertible linear mapping. The important metric for its security is *branch number* [12], it gives measure on worst case diffusion. If the diffusion matrix has a maximal possible branch number, it is *optimal.* AES [4], Twofish [3] and SHARK [13] ciphers are using MDS matrices optimizing branch number as main security measure of diffusion layer.

For generating such MDS matrices is particularly interesting following proposition from SHARK cipher paper [13] (for proof see original paper).

Proposition 2. *Let C be a $(2n,\ n,\ n+1)$-code over the Galois field $GF(2^m)$. Let G_e be the generator matrix of C in echelon form:*

$$Ge = \begin{bmatrix} I_{n\times n} & B_{n\times n} \end{bmatrix} \tag{5.5}$$

Then C defines an optimal invertible linear mapping γ:

$$\gamma \ : \ GF(2^m)^n \rightarrow GF(2^m)^n = X \mapsto Y = B \cdot X \tag{5.6}$$

Recall that $(2n,\ n,\ n+1)$-code is MDS [1]. Reed-Solomon codes are subset of MDS codes, so their parity check matrix can be used as MDS matrix, in the cipher acting as a strong diffusion element. MDS matrices derived from Reed-Solomon codes are used by many ciphers, for example Twofish, Shark.

In our case we would be interested in $(32,\ 16,\ 17)$-code, to obtain 16×16 MDS matrix with wanted properties.

Another way how to generate MDS matrices with is described in [14] using Cauchy matrices.

It is important to mention that ciphers using MDS matrix as diffusion element usually puts additional requirements on the MDS matrix, also optimizing performance and simplicity in hardware implementation. In blackbox context it is usually security/performance trade off. In whitebox context we need to have diffusion element very strong, so we can neglect performance point of view to increase security.

Also article [15] mentions AES diffusion layer modification from 4×4 to 16×16 MDS matrix, argumenting with stronger security within one round, what is particularly interesting in whitebox context. They are constructing MDS matrix using Cauchy matrices. Cauchy matrices depends on the first row only, this increases possible diversity of 16×16 MDS matrices helping the following idea - key dependent diffusion.

Consider also idea to have key-dependent MDS matrices. If we can generate set S_{MDS} of MDS matrices representing optimal linear mapping, their selection can be based on key-dependent criteria. Set S_{MDS} can be also extended using following proposition from [16].

Proposition 3. *Let* $B = [b_{i,j}]_{n \times n}$, $b_{i,j} \in \mathbb{F}_q$ *an MDS matrix, for an element* $e \in \mathbb{F}_q,\ e \neq 0$, $e \cdot B$ *is an MDS matrix.*

Having key-dependent diffusion layer also complicates whitebox attacks, namely Billet's [7] and Generic attack by Michiels [17] requires known MDS matrix coefficients (thus key-independent).

5.5 Analysis

In this chapter we try to analyze suggested scheme improvement from whitebox point of view, particularly we try to mount BGE attack to this modified variant.

S-box definitions are needed in proposition 3 in BGE attack where we obtain 4 affine

1. $(n,\ k,\ d)$-code is MDS iff $d = n - k + 1$

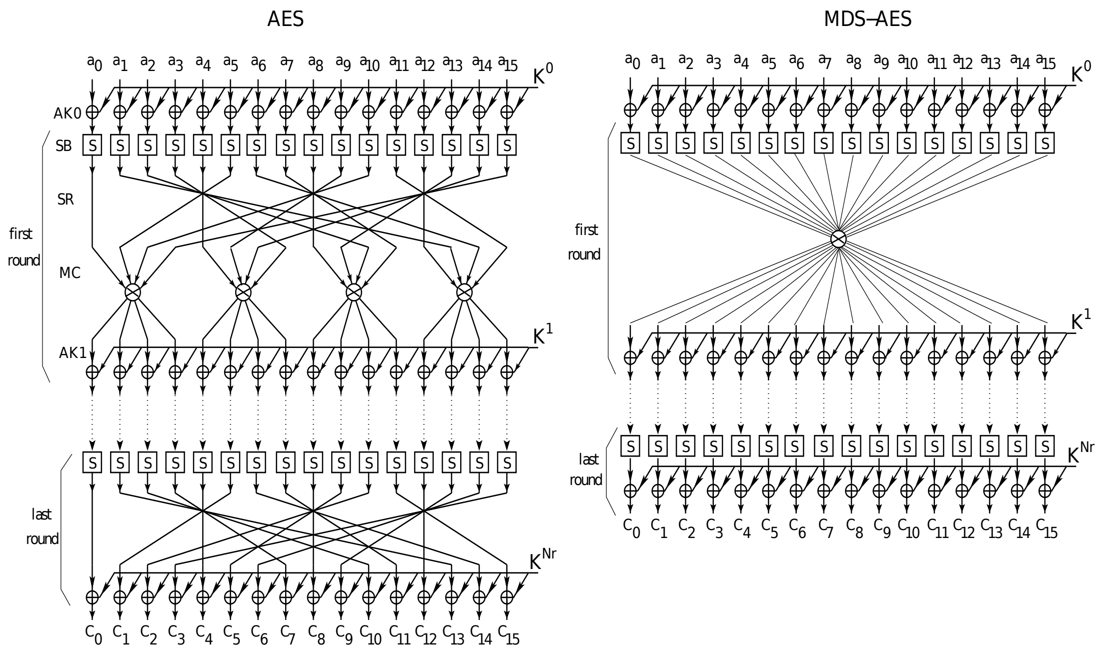
AES
MDS–AES
a_0 a_1 a_2 a_3 a_4 a_5 a_6 a_7 a_8 a_9 a_{10} a_{11} a_{12} a_{13} a_{14} a_{15}
K^0
AK0
SB
SR
first round
MC
K^1
AK1
last round
K^{Nr}
c_0 c_1 c_2 c_3 c_4 c_5 c_6 c_7 c_8 c_9 c_{10} c_{11} c_{12} c_{13} c_{14} c_{15}

Figure 5.1: Computational diagrams of AES and MDS-AES (taken from [15])

mappings.

$$\widetilde{P}_0 \; : \; x \mapsto \left(S^{-1} \circ \Lambda_{\delta_0} \circ \widetilde{A}_0^{-1}\right) \left(y_0\left(x, 00, 00, 00\right)\right) \tag{5.7a}$$

$$\widetilde{P}_1 \; : \; x \mapsto \left(S^{-1} \circ \Lambda_{\delta_1} \circ \widetilde{A}_0^{-1}\right) \left(y_0\left(00, x, 00, 00\right)\right) \tag{5.7b}$$

$$\widetilde{P}_2 \; : \; x \mapsto \left(S^{-1} \circ \Lambda_{\delta_2} \circ \widetilde{A}_0^{-1}\right) \left(y_0\left(00, 00, x, 00\right)\right) \tag{5.7c}$$

$$\widetilde{P}_3 \; : \; x \mapsto \left(S^{-1} \circ \Lambda_{\delta_3} \circ \widetilde{A}_0^{-1}\right) \left(y_0\left(00, 00, 00, x\right)\right) \tag{5.7d}$$

In our implementation of the BGE attack we iterate over $(\delta_i, c_i)_{i=0,\ldots,3} \in \mathrm{GF}(2^8) \times \mathrm{GF}(2^8)$ what gives complexity 2^{16} for one mapping. In each step is mapping checked for affinity in 2^8 steps (for affinity check algorithm see A.0.1), altogether one relation takes 2^{24} steps, for all relations 2^{26} steps.

Here is the place where we use public knowledge of AES S-Box definitions. One way how to mount BGE attack to this modified variant is to guess also particular S-box mapping for each $\widetilde{P}$ and to test for its affinity.

Equations 5.1 describe Twofish S-boxes. There are 2^{16} possible s_0 S-boxes. One S-box mappings stored as look-up table takes 2^8 bytes. Thus pre-computed s_0 s-box for all possible key bytes would take $2^8 \cdot 2^{16} = 2^{24} > 10^7$ bytes.

Even if attacker determines round keys for S-boxes it will be completely useless for further extraction of cipher key since hash chains are different.

Twofish S-boxes thus increase complexity of proposition 3 from BGE from 2^{24} to 2^{40}. This is still not strong enough, it is highly parallelizable problem. In order to increase work needed to mount proposition 3 attack one could redefine key-dependent S-boxes to increase attack complexity to level 2^{128} what is larger than best known attack on AES. We then would S-box need to depend on 13 bytes derived from encryption key. Upper bound on number of different non-linear S-boxes is $256! \approx 8,5 \cdot 10^{506}$ so there are still options to expand S-box space.

$$s'_j\left(x\right) = sboxgen(j, 12, x) \tag{5.8}$$

Where S-box is generated recursively by *sboxgen*, following the idea of nesting fixed permutations with addition of round key:

$$sboxgen(j,l,x) = \begin{cases} q'_{j,0}\left[x\right] \oplus k_{j+1} & \text{if } l = 0 \\ q'_{j,l}\left[sboxgen\left(j, l-1, x\right)\right] \oplus k_{(j+1)\cdot(l+1)} & \text{otherwise} \end{cases} \tag{5.9}$$

Where $q'_{j,l}$ is one of two fixed Twofish 8-bit permutations. Particular sequence of nested permutations would require deeper analysis, to avoid possible weaknesess induced by composing inappropriate permutations together, but we can choose for example:

$$q'_0 = \begin{bmatrix} q_1 & q_0 & q_1 & q_0 & q_1 & q_0 & q_1 & q_0 & q_1 & q_0 & q_1 & q_0 & q_0 \end{bmatrix} \tag{5.10a}$$

$$q'_1 = \begin{bmatrix} q_1 & q_1 & q_0 & q_0 & q_1 & q_1 & q_0 & q_0 & q_1 & q_1 & q_0 & q_0 & q_1 \end{bmatrix} \tag{5.10b}$$

$$q'_2 = \begin{bmatrix} q_0 & q_0 & q_1 & q_1 & q_0 & q_0 & q_1 & q_1 & q_0 & q_0 & q_1 & q_1 & q_0 \end{bmatrix} \tag{5.10c}$$

$$q'_3 = \begin{bmatrix} q_0 & q_0 & q_0 & q_1 & q_1 & q_1 & q_0 & q_0 & q_0 & q_1 & q_1 & q_1 & q_1 \end{bmatrix} \tag{5.10d}$$

This gives us good security margin for BGE attack, raising computational complexity to 2^{128}. Pre-computed s_0 S-box for all possible key bytes would take $2^8 \cdot 2^{8\cdot 13} = 2^{112} > 10^{33}$ bytes.

Also as long as are SHA-256 and bcrypt uncracked, key extraction should be unfeasible, since it is not possible to invert hash function easily.

Cipher modification that affects only S-boxes and round keys, so other parts of the BGE attack are not affected, namely transforming non-linear parts to affine, propositions 1 and 2 work as before.

Cipher modification don't increase table sizes, because modifications made are only in S-box definitions and key schedule - both parts of the cipher are already evaluated and stored in the look-up tables. Our modifications also don't affect encryption/decryption performance since the only difference is made during particular whitebox instance (look-up tables) generation. Encryption/decryption algorithm itself is not affected.

5.6 Analysis of diffusion layer

Taking also section 5.4 into account will result also in bigger look-up tables. All table types are affected, in particular type 2, previously it was mapping $2^8 \rightarrow 2^{32}$, with cascade of XOR tables to sum 4×32-bit values to obtain one column of state array. Now type 2 tables are $2^8 \rightarrow 2^{128}$, with cascade of XORs to obtain whole state column vector. XOR tables takes 128-bit arguments, plus 3 additional 128-bit XOR tables are needed (to sum 4×128-bit results from columns).

From security point of view this modifications also prevents inverting attack mentioned in section 3.3.1. Function is now too wide to be inverted by running over $GF(2^8)^{16}$.

Key-dependent MDS matrix also prevents mounting Billet's attack. In particular we don't know MixColumn matrix coefficients so we are not able to construct set β from section 3.3 in [7]. This leads to further ambiguities so proposition 2 and 3 won't work anymore. This also makes recovering affine parts of I/O encodings unfeasible, using this attack. But proposition 1 still works, random I/O bijections can be converted of affine ones quite easily.

As noted above, also Generic attack by Michiels [17] requires known MDS matrix coefficients.

Drawbacks

By modification of AES design we are coming up with new cipher, what brings also some possible problems. AES and Twofish have advantage of being well analyzed from blackbox context and being relatively secure. Designing a new cipher may help with increasing security in whitebox context but there also may be weaknesses in blackbox context. It would be needed to analyze the new cipher from this point of view, for example for resistance to linear or differential cryptanalysis. There is no guarantee that proposed cipher modifications are strong enough to resist classical blackbox context cryptanalysis.

But when designing scheme improvements we tried to follow standard principles in designing secure block cipher, inspired by AES, Twofish, Shark and others.

5.7 Discussion

Few words about field, discussion about WB cryptography.

6 Future work

As a future work I would like to study blackbox properties of suggested improved cipher. In particular it is worth to study S-box construction, to test it for possible weaknesses. It will be needed to model S-boxes as algebraic equations and to test it for fixed points, for example. One can also study this S-boxes and their differential / linear probability coefficients (measuring resistance of an S-box to differential / linear cryptanalysis). One can test S-boxes for Square and linear approximation attacks.

Resistance to Billet's attack results from dependence on 13 round key bytes. If one shows that S-box space is smaller than assumed, it can be vulnerability that can be exploited by whitebox attack.

I would like to examine attacks on whitebox implementations, when each round of the cipher is considered as a single mini-cipher with its own key. Whitebox cipher implementation then would be network of serially connected mini-ciphers. From this point of view I would be interested in resistance of the mini-ciphers to linear and differential cryptanalysis.

A Appendix A

A.0.1 Affinity check

In this section we describe affininity check needed in proposition 3 of BGE attack. We are given relation $\widetilde{P}$ as a look-up table and the task is to test it for affinity. If $\widetilde{P}$ is affine it must hold:

$$\widetilde{P}\left(x\right) = M \times x \oplus c \tag{A.1}$$

For some square matrix M with coefficients from GF(2) and constant $c \in \mathrm{GF}(2^8)$ (or equivalently 8×1 vector with coefficients from GF(2)).

By evaluating $\widetilde{P}\left(0\right) = c$ we obtain affine constant c so we derive new mapping $\widetilde{P'}$, reducing the problem to test $\widetilde{P'}$ for being linear.

$$\widetilde{P'}\left(x\right) = \widetilde{P}\left(x\right) \oplus c \tag{A.2}$$

And from linearity the following formula must hold:

$$\forall x \in \mathrm{GF}(2^8),\ \exists!\ k_j \in \{0,1\},\ j \in [0,7]\ :\ x = \sum_{j=0}^{7} k_j \cdot \widetilde{P'}\left(e_j\right) \tag{A.3}$$

It says that each element from the field has to be unique sum of its basis vectors. Assuming that $\widetilde{P'}$ is linear, we can obtain mapped base vectors for this transformation easily as $g_j = \widetilde{P'}\left(e_j\right)$. Now it is visible that time complexity is 2^8.

Algorithm 1 Algorithm for testing given mapping for being affine

```
1: function isAffine($\widetilde{P}$ : GF($2^8$) $\mapsto$ GF($2^8$))        ▷ Determine if P is affine mapping
2:     $c \leftarrow \widetilde{P}[0]$                                       ▷ $c$ is affine constant
3:     $\widetilde{P'}[x] \leftarrow \widetilde{P}[x] + c$                   ▷ $2^8$ time complexity
4:     $isAffine \leftarrow true$
5:     for $x \leftarrow 0, (2^8 - 1)$ do
6:         $px \leftarrow \widetilde{P'}[x]$
7:         $cx \leftarrow 0$
8:         for $i \leftarrow 0, 7$ do
9:             if $x_i = 1$ then                                             ▷ $x_i$ is $i$-th bit of x in binary
10:                $cx \leftarrow cx \oplus \widetilde{P'}[e_i]$             ▷ $\widetilde{P'}[e_i]$ is mapped base vector
11:            end if
12:        end for
13:        if $px \neq cx$ then                                              ▷ $cx$ is expressed via mapped base vectors
14:            $isAffine \leftarrow false$
15:            return
16:        end if
17:    end for                                                               ▷ All elements from field checked for linearity
18:    return $isAffine$
19: end function
```

Bibliography

[1] S. Chow, P. Eisen, H. Johnson, and P. C. Van Oorschot. White-box cryptography and an aes implementation. In *Proceedings of the Ninth Workshop on Selected Areas in Cryptography (SAC 2002*, pages 250–270. Springer-Verlag, 2002.

[2] Tim Kerins and Klaus Kursawe. A cautionary note on weak implementations of block ciphers. In *In 1st Benelux Workshop on Information and System Security (WISSec 2006*, page 12, 2006.

[3] Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, and Niels Ferguson. Twofish: A 128-bit block cipher. In *in First Advanced Encryption Standard (AES) Conference*, 1998.

[4] Joan Daemen and Vincent Rijmen. *The design of Rijndael: AES—the advanced encryption standard*. Springer-Verlag, 2002. ISBN 3–540–42580–2.

[5] Brecht Wyseur. White-box cryptography: Hiding keys in software. [online], 2012. URL `http://whiteboxcrypto.com/files/2012_misc.pdf`.

[6] Mohamed Karroumi. Protecting white-box aes with dual ciphers. In *Proceedings of the 13th international conference on Information security and cryptology*, ICISC'10, pages 278–291, Berlin, Heidelberg, 2011. Springer-Verlag. ISBN 978-3-642-24208-3. URL `http://dl.acm.org/citation.cfm?id=2041036.2041060`.

[7] Olivier Billet, Henri Gilbert, and Charaf Ech-Chatbi. Cryptanalysis of a white box aes implementation. In *Proceedings of the 11th international conference on Selected Areas in Cryptography*, SAC'04, pages 227–240, Berlin, Heidelberg, 2005. Springer-Verlag. ISBN 3-540-24327-5, 978-3-540-24327-4. doi: 10.1007/978-3-540-30564-4_16. URL `http://dx.doi.org/10.1007/978-3-540-30564-4_16`.

[8] Niels Provos and David Mazieres. A future-adaptable password scheme. In *In Proceedings of the 1999 USENIX, Freenix track (the on-line version)*, page 99, 1999.

[9] Colin Percival. Stronger key derivation via sequential memory-hard functions, 2009. URL `http://www.daemonology.net/papers/scrypt.pdf`.

[10] Butterfly labs. 1,500 GH/s Bitcoin Miner. `https://products.butterflylabs.com/homepage/1500gh-bitcoin-miner.html`, 2013. [Online; accessed 15-May-2013].

[11] Jeremi M. Gosney. Password cracking hpc. Passwords Î2 Security Conference, [online], December 2012. URL `http://passwords12.at.ifi.uio.no/Jeremi_Gosney_Password_Cracking_HPC_Passwords12.pdf`.

[12] Joan Daemen. *Cipher and hash function design. Strategies based on linear and differential cryptanalysis.* Phd. thesis, Katholieke Universiteit Leuven, 1995. URL `http://docserver.bis.uni-oldenburg.de/publikationen/dissertation/2001/aneacr01/aneacr01.html`.

[13] Vincent Rijmen, Joan Daemen, Bart Preneel, Antoon Bosselaers, and Erik De Win. The cipher shark. In Dieter Gollmann, editor, *FSE*, volume 1039 of *Lecture Notes in Computer Science*, pages 99–111. Springer, 1996. ISBN 3-540-60865-6. URL `http://dblp.uni-trier.de/db/conf/fse/fse96.html#RijmenDPBW96`.

[14] Ron M Roth and Gadiel Seroussi. On generator matrices of mds codes. *IEEE Trans. Inf. Theor.*, 31(6):826–830, nov 1985. ISSN 0018-9448. doi: 10.1109/TIT.1985.1057113. URL `http://dx.doi.org/10.1109/TIT.1985.1057113`.

[15] Jorge Nakahara Jr. and Elcio Abrahao. A new involutory mds matrix for the aes. *I. J. Network Security*, 9(2):109–116, 2009. URL `http://dblp.uni-trier.de/db/journals/ijnsec/ijnsec9.html#JrA09`.

[16] Muhammad Yasir Malik and Jong-Seon No. Dynamic mds matrices for substantial cryptographic strength. *IACR Cryptology ePrint Archive*, 2011:177, 2011. URL `http://dblp.uni-trier.de/db/journals/iacr/iacr2011.html#MalikN11`.

[17] W. Michiels and P. Gorissen. Mechanism for software tamper resistance: an application of white-box cryptography. In *Proceedings of the 2007 ACM workshop on Digital Rights Management*, DRM '07, pages 82–89, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-884-8. doi: 10.1145/1314276.1314291. URL `http://doi.acm.org/10.1145/1314276.1314291`.