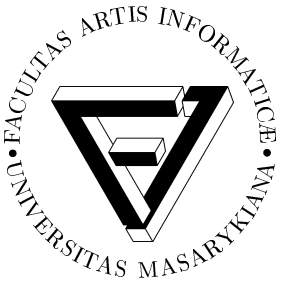
Masaryk University
Faculty of Informatics



White-box attack resistant 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

I would like to express my gratitude to my thesis advisor, RNDr. Petr Švenda, Ph.D., for his guidance, helpfull ideas, patience and non-negligible time invested in consultations what made this thesis possible. I am also grateful to people form Laboratory of Security and Applied Cryptography of the Faculty of Informatics of Masaryk University, for their valuable comments, especially to RNDr. Jiří Kůr.

I would like to thank my parents for their constant support and to my girlfriend, Martina Bošková, for her encouragement and inspiration.

I am also thankfull to all my close friends for their advices and support.

Abstract

This thesis is focused on a study of security issues related to an execution of cryptographic algorithms in an untrusted environment. It mainly studies whitebox cryptography methods of transforming algorithms in such a way they resist attacks like key-extraction and inverting in some extent. Particularly it examines whitebox transformations of AES cipher and attacks on these transformations. Transformations construction and implementation is described. In the thesis was discovered the known attack works also on AES transformation using dual ciphers by Karroumi [1] that was supposed to resist the attack. The new improvements for increasing a resistance of transformations to known attacks were proposed.

Keywords

whitebox attack resistant cryptography, AES, dual ciphers, whitebox cipher implementation

Table of contents

1 Introduction

In the last few decades we have been witnessing a development in a field of outsourced computations and storage. The rising prevalence of this computing model slightly changes the classical attacker model that cryptography used to dealt with, what gave rise to a *mobile cryptography*.

The classical goals the cryptography addresses are confidentiality, data integrity, authentication and non-repundation [2]. From the data confidentiality perspective, the typical scenario is two remote parties, Alice and Bob, want to communicate via untrusted channel, while the computations on both sides are considered as trusted. A potential attacker resides in a communication link. A bunch of cryptography primitives addressing security issues in this scenario was invented, analyzed and widely used, e.g. symmetric and asymmetric cryptosystems, digital signatures, authentication protocols, etc...

But with the expansion of outsourced computations and storage we are getting to a situation that Alice does not trust even to Bob, but wants to use Bob's resources for her own purpose. Such outsourcing rises concerns about the loss of privacy of private data what poses the potential barrier in adopting cloud services widely. To ensure the privacy, data is encrypted. The major problem with this model is that in order to evaluate a function over data, e.g. searching in an encrypted database, data has to be decrypted first. This poses an another additional overhead. The *fully homomorphic encryption* provides a solution for these issues.

Another major part of use cases is the protecting a private function computed in an untrusted environment. A typical example of the function to be protected is the license code verification embedded in a software or it is a software that provides access to some protected material, e.g. copyrighted content. The major goal is to protect these functions from analysis, tampering or extraction of a cryptographic material. Software protection techniques like an *obfuscation* addressing these issues are used in practice. This thesis is devoted to a *whitebox cryptography*, the field of cryptography that studies the level of security of cryptographic algorithms executed in an untrusted environment.

This thesis studies in particular a transformation of the AES [3] implementation in such a way that they provide some level of security when executed in an untrusted environment. This transformed implementation has embedded a symmetric key inside and the main goal of the transformation is to resist practical attacks attempting its extraction. The thesis covers an introduction to whitebox cryptography, describes a first such transformation published together with cryptanalysis. It also analyzes a new proposed transformation that should resist a key extraction. The thesis states also a proof that this scheme can be broken using an already existing attack. In the end, some improvements are suggested.

Chapter 2 describes mobile cryptography concepts and state of the art in this field mainly with focus on the obfuscation and the homomorphic encryption. In chapter 3 the whitebox

cryptography is introduced, followed by an explanation of basic building blocks used for transformations. The first proposed scheme by Chow *et al.* [4] is described in detail together with the following cryptanalysis by Billet *et al.* [5]. Chapter 4 covers a new whitebox scheme for AES by Karroumi [1] in a detail. A description of the schemes and the attack implementation follow. In chapter 5, improvements for whitebox schemes are proposed with their analysis. The last two chapters are devoted to possible further research directions and conclusion.

I declare this thesis is my own work, but I consulted the problems and proposed solutions together with people from the Laboratory of Security and Applied Cryptography of the Faculty of Informatics of Masaryk University, and from this reason is a personal pronoun "we" used instead of "I".

2 Area overview

2.1 Overview

Computing in an untrusted environment is closely related to the notion of *mobile cryptography* [6] which was established two decades ago. It analyzes security problems raised by a concept of mobility of an executable code. The executable code that acts autonomously on behalf a user in collecting and processing information is denoted as a *mobile agent*. Mobile cryptography mainly studies two security threats:

1. protection of the host from malicious mobile code
2. protection of mobile code from the malicious host

The former threat can be mitigated to an acceptable level with countermeasures like sandboxing, virtualization and code signing [2], what is widely adopted by current anti-virus protection software and operating systems.

The latter is much difficult to address. Existing techniques provide protection to some extent, making tampering the code on malicious host difficult for ordinary attacker, but there are no guarantees of protection against very strong and determined attacker with enough resources to invest in the attack.

In order to make tampering of the software very difficult, specialized, tamper-resistant hardware is often used. It is designed with security concerns in mind so that very advanced techniques and a large amount of resources is needed in order to attack such device, making it practically impossible for an ordinary attacker. For an example hardware security modules used by banks or certification authorities protecting their secret cryptographic material. Another example is a cryptographic smart card, widely used in use cases with high security requirements.

However, the tamper-resistant hardware is not suitable for many applications, due to it's cost and physical nature, e.g. need to be distributed somehow, can be lost, forgotten, unintentionally damaged, etc... Then it is preferable to use software-based protection techniques for their low cost and flexibility. The downside of this approach is a limited strength.

2.2 Obfuscation

An *obfuscation* is another technique addressing the same problem, protecting a software implementation. Roughly speaking, the major principle is the transformation of the code to a form, that is very difficult to analyze and eventually to modify. The potential attacker should not be able to gain any extra knowledge[1] from the running program, in the ideal case, while the original functionality of the program is preserved.

1. besides input/output behavior

The program obfuscation received an attention when Barak *et al.* formalized the notion of obfuscation in [7], providing significant theoretical result that it is impossible to create a generic obfuscator. They did so by showing the existence of a (contrived) family of functions that are unobfuscatable, i.e. the family of functions always leaking some information. They used assumption of existence of one-way functions.

On the other hand, later was published a first positive result [8] claiming it is possible to construct some provably secure obfuscators for *point functions*. Point function accepts a single input string and reject all other inputs. It was used to obfuscate complex access control functionalities.

The first positive obfuscation result for a traditional cryptographic functionality (that is significantly more complicated that point functions) was presented by Hohenberger *et al.* [9]. They used slightly modified definition of obfuscation in order to construct a secure obfuscator for re-encryption[2].

The question whether there exist a family of potentially interesting functions for which exist provably secure obfuscators and how to construct them, is a subject of a further research. But the work of Goldwasser *et al.* [10] suggest it is unlikely. Namely they state that there exist many natural classes of functions that cannot be obfuscated w.r.t. auxiliary input.

An *approximate obfuscation* defined by Barak *et al.* [7] is a relaxation of the functionality requirement of the obfuscated program. They presented impossibility result in the case when an obfuscated program is deviates from the original circuit only with negligible probability and allows this event to depend only on the coin tosses of the obfuscator. Recently Bitansky *et al.* [11] improved this impossibility result by hardening the requirements. They showed there exist families of robust unobfuscatable functions for which even approximate obfuscation is impossible. According to their definition, obfuscated program is only required to agree with the original one with probability slightly more than 0.5 on a uniformly sampled input, what was the open problem till then.

In a practice, the obfuscation is widely used as a software protection technique that provides some level of protection from attackers, but it often lacks some proof of security. In major cases it is collection of techniques that makes the static and/or run-time analysis of a program significantly more difficult, but it does not rule out the probability of an successful attack by a strong and determined attacker. A rich collection of state of the art obfuscation techniques, protecting from static and run-time analysis can be found in the dissertation thesis of J. Cappaert [12].

The concept of obfuscation is closely related to *computing with a private function*.

2. This functionality takes a ciphertext for message m encrypted under Alice's public key and transforms it into a ciphertext for the same message m under Bob's public key. [9]

2.3 Computing with encrypted data

As is mentioned in the introduction, the fundamental question whether data can be manipulated without being decrypted has been attracting attention for a long time.

2.3.1 Secure multiparty communication

The first positive results on this question make use of *interaction*. The concept of a *secure multiparty communication* was introduced by Yao [13] in 1982. Roughly speaking, it enables to evaluate a function over private data of remote parties, while keeping the private data still confidential. For this, both parties have to follow some protocol. A typical toy example is a well-known Yao's Millionaire's Problem. In this problem two millionaires, Alice and Bob, want to know which is richer, without disclosing their actual wealth. Note the first protocol solving this problem had exponential time, space and communication complexity. This problem has direct applications in e-commerce, e.g. on-line bidding and auctions and data mining [14].

Many protocols for secure multiparty schemes are based on *arithmetic circuits*. This is also a one of cornerstones used in the following chapter, so it is important to describe it.

The function $\mathcal{F}$ is transformed to a network, that forms a directed acyclic graph, of gates performing addition and multiplication operation, what forms an arithmetic circuit that is able to evaluate the function $\mathcal{F}$. It is known that considering arithmetic circuits is without a loss of generality, i.e. any function that is feasible to compute at all can be specified as a polynomial-size Boolean circuit using *and* and *negation*. Note that any such circuit can be simulated by operations in $\mathbb{F}$: Boolean values *true* or *false* can be encoded as 1 resp. 0. Then *negation* of a bit b is $1 - b$ and *and* of bits b, c is $b \cdot c$ [14]. The resulting circuit is then evaluated by remote parties in order to compute function $\mathcal{F}$ over their private data. A *depth* of a circuit is the longest path in the circuit.

Ishai *et al.*[15] in 2008 demonstrated a two-party computation protocol of a function $\mathcal{F}$ while communication overhead is a fixed constant factor larger than circuit size of $\mathcal{F}$.

2.3.2 Fully homomorphic encryption

A cryptosystem is denoted as a fully homomorphic if it supports evaluation of two operations, *addition* and *multiplication*, on ciphertexts where the result after decryption matches the same operation on corresponding plaintexts.

$$a + b \equiv D\left(E\left(a\right) \oplus E\left(b\right)\right) \tag{2.1a}$$

$$a \cdot b \equiv D\left(E\left(a\right) \odot E\left(b\right)\right) \tag{2.1b}$$

The power of the the fully homomorphic system is in it's ability to evaluate an arbitrary function over encrypted data, without actually decrypting the data. This is exactly the situation where Alice wants to outsource some computations to Bob, but doesn't want Bob to

learn her information. When the function is evaluated homomorphically, the result is again encrypted, so only Alice is able to read it. This is done by using the concept from the previous chapter, *arithmetic circuits*.

The function $\mathcal{F}$ that Alive wants to compute, is converted into the *arithmetic circuits* that computes the same function. This circuit evaluated over plaintext gives the wanted result, but note that it can be evaluated also over ciphertexts using the homomorphic property of the cryptosystem. Intuitively, Alice sends circuit representing $\mathcal{F}$ to Bob, Bob evaluates the circuit on ciphertext and returns a result to Alice. When Alice decrypts the result from Bob, obtains the result of $\mathcal{F}$.

It is important to emphasize that in this use case, the cryptosystem becomes a computational platform, thus the possible space/time overheads slow down entire computation.

History. The concept of *computing with encrypted data* was first proposed by Rivest *et al.* [16] in 1978, a few months ago before introduction of RSA implementation. They suggested that fully homomorphic encryption may be possible, but were unable to find such scheme. The question whether it is possible to construct fully homomorphic scheme was an open problem for 30 years.

Some partially homomorphic schemes were known, for example RSA supports homomorphic evaluation of multiplication. There were also some limited homomorphic schemes published, for example [17] in 2005. Their cryptosystem is based on elliptic curves and supports unlimited number of additions and one multiplication operation. Even this restricted scheme has interesting applications, for example efficient election system, as proposed in their paper.

The breakthrough in this field was done by Gentry in 2009 [18]. He demonstrated the fully homomorphic encryption (FHE) scheme is possible to construct, using an ideal lattices. Since then this field is undergoing a rapid development.

Key ideas. Gentry first constructed "*somewhat homomorphic*" scheme that supports evaluating of low-degree polynomials on ciphertexts (corresponds to evaluating an arithmetic circuit of a small depth). To protect the information (plaintext), it is hidden in a large amount of *noise*. Without going into further details, the main problem is the addition doubles and the multiplication squares the noise level. Once the level exceeds acceptable boundary, decryption is ambiguous even for Alice.

The ingenious idea of a noise reduction is called *refreshing*. It is a process of evaluating decryption circuit homomorphically. Note that such evaluation produces again ciphertext, since the result of homomorphic operation is still encrypted, but the level of noise is normalized.

Using this idea, Gentry built the FHE from the somewhat homomorphic scheme, by periodically applying the refreshing operation when the noise reached the acceptable level.

Both symmetric and asymmetric schemes were proposed.

Recent advances. There are three main FHE schemes known to date:

1. Gentry's original scheme based on ideal lattices. The implementation was introduced by Gentry *et al.* in [19]. The public key has a size 2.3 GB, refreshing operation takes 30 minutes.

2. Dijk's *et al.* [20] scheme DGHV, based on problem from number theory, approximate Greatest Common Divisor (GCD).

   - simpler that previous scheme
   - The latest results by Coron *et al.* [21] from 2012, significantly reduced the public key size to 10.3 MB, refreshing operation takes 11 minutes.
   - The result from 2013 by Coron *et al.* [22] added support for performing *batch* operations with plaintexts
   - The fully homomorphic evaluation of AES encryption was performed, with amortized cost 12 minutes per AES block on a standard desktop computer with 32 GB RAM [22] .

3. Brakerski *et al.* [23] Ring Learning With Errors (RLWE) scheme, adaptation of previous Learning With Errors (LWE) scheme. The LWE hard problem is to recover $s \in \mathbb{Z}_q^n$ given a sequence of approximate random linear equations of s.

   - The improvement by Brakerski *et al.* [24] changed the noise management via *modulus switching*. The refreshing procedure as used by Gentry is not necessary in this case. The noise is reduced gradually after each multiplication, protecting from growing exponentially.
   - Improvement by Gentry *et al.* [25] adds batch operation, using a *cyclotomic number field*[3].
   - The fully homomorphic evaluation of AES encryption was performed [26] with amortized time 37 minutes per AES block on a standard desktop computer with 256 GB RAM.

Batch operation, also called plaintext "packing", is a technique where multiple independent plaintexts slots are embedded into a single ciphertext, using proper algebraic structure. Then when an operation is performed on the ciphertext, it has effect like it is performed on the whole vector of plaintexts embedded in the ciphertext. This strongly resembles Single-Instruction-Multiple-Data (SIMD) architecture of a paralell computer. This adds significant improvement, since multiple blocks (like in AES case) are computed simultaneously, what

3. `http://www.math.harvard.edu/~erickson/pdfs/cyclotomic_fields_part_iii.pdf`

gives better amortized running time of algorithms. Recall that in original Gentry's scheme, the plaintext space size was only 1 bit.

As an illustration[4] consider plaintext space is group $Z_{15} = Z_3 \times Z_5$, from Chinese Remainder Theorem. Using this structure we obtain 2 slots for plaintext of size 3 and 5. Let's have two ciphertexts c, c' with (p_3, p_5) and (p'_3, p'_5) in their plaintext slots respectively.

Then after ADD(c, c') the plaintext slots are $(p_3 + p'_3, p_5 + p'_5)$. The analogy holds also for MULT(c, c') then the plaintext slots are $(p_3 \cdot p'_3, p_5 \cdot p'_5)$.

The batching mechanism proposed in [24] is based on the similar idea but uses ring that optimizes number of plaintext slots in ciphertext, by choosing a more appropriate algebraic structure.

Somewhat homomorphic encryption schemes. FHE schemes is a very active area of research nowadays, with still better and better improvements on a performance of the schemes, but in spite of this the practical use of FHE is stil out of question due to its computational complexity.

Naehrig *et al.* [28] proposed to sacrifice "fully" property and use just *somewaht homomorphic schemes* (SWHM), with limited number of multiplications. In this setup it is not possible to evaluate arbitrary function, but some families of functions can still be useful. They give examples of application in medical, financial sectors and advertising.

Boneh *et al.* [29] designed and implemented a protocol for private database queries using somewhat homomorphic encryption.

4. Example taken from [27]

3 Whitebox cryptography

3.1 Introduction

Whitebox cryptography studies security issues related to an execution of cryptographic algorithms in an untrusted environment, it is than said to be 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.* [4] in 2002. The WBC attacker has a full control over execution of the particular algorithm. Namely attacker has the following abilities:

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

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

The cryptographic algorithms (we are mainly interested in symmetric ciphers in this thesis) were extensively studied for attacks in the BBC in past. They were originaly designed to resist attacks considering only the BBC. But if the context is wrong, it can be a possible entry point for an attacker. Typical example is DRM [1], where software of a vendor (representing the rights owner) is executed in a 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 the 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 the execution or trace the algorithm flow.

One such whitebox attack is the *Key Whitening Attack* [30]. Key whitening is a 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 is used by Twofish [31] and in a modified version (only adding the key material in the last round) also by AES [3]. In Key Whitening Attack cipher's binary is modified (we are in whitebox context) in such a way that the output of the cipher will be the key material itself.

Main two attacks in whitebox context are: (1) Key Recovery, i.e. an extraction of a embedded symmetric key. (2) Plaintext recovery under Chosen Plaintext Attack (PR-CPA), e.g. perform decryption with implementation of cipher with embedded encryption that is supposed to be able only to perform encryption.

Whitebox cryptography is closely related to the *obfuscation* mentioned in the section 2.2. It is also a program transformation, but obfuscation, as defined in the literature, is too restrictive and does not take specific security notios, e.g. cipher invertibility a.k.a. PR-CPA, into account.

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." [32].

3.2 History

Whitebox cryptography is a quite new field of cryptography. The study of the whitebox implementation of the ciphers started by first whitebox implementation of AES [4] and DES [33] by Chow *et al.* in 2002.

At first, the cryptanalysis of DES focused on its simplified variant. The first published in 2002 by Jacob *et al.* uses fault injection [34], another one published in 2005 by Link *et al.* uses statistical analysis [35]. Later cryptanalysis of fully encoded variant of DES was published by Wyseur *et al.* in 2007 using truncated differentials.

The similar case holds for AES. Two years after publishing the whitebox AES scheme the successful cryptanalysis [5] was published by Billet *et al.* that enabled to recover embedded symmetric key in less that 2^{30} steps. Later, in 2008, the generalized version of the previous attack was published [36] by Michiels *et al.* affecting the larger family of ciphers using the same structure as AES.

There were also attempts to fix whitebox AES scheme by adding additional linear mappings and increasing size of the implementation in [37] as a response to the Billet's attack. The attack against improved scheme using a linear equivalence algorithm was published in 2012 [38].

The another attempt, how to fix whitebox AES, was introducing random perturbations [39], complicating algebraic cryptanalysis, but the effective attack was published by Mulder *et al.* [40].

Last, but not least a whitebox AES scheme using dual ciphers [1] was published in 2011. The paper claimed the scheme is robust enough to resist practical attacks on the implementation. We proved this assumption false by finding out the published attack works in the same way on this implementation as on the original one, for the proof see the section 4.2.

This thesis is mainly focused on the scheme using dual ciphers, note it is generalization of the first whitebox AES scheme.

3.3 Whitebox AES scheme

The first whitebox AES implementation, published by Chow *et al.* [4] is based on the look-up tables implementation, that was also mentioned in original AES paper [3]. Note that it is easy to transform AES with an embedded encryption key to a network of look-up tables and to use these computed tables for an encryption (or a decryption). But this implementation is vulnerable in the WBC, since it is possible to extract the encryption key with algebraic analysis of the look-up tables. Recall that all the building blocks (except key schedule) of AES are key-independent and publicly available. Thus these look-up tables have to be further protected to resist algebraic attacks.

We are mainly focused on AES-128, for simplicity, but the same strategy can be applied also to AES-192 and AES-256.

For further explanation we will need the following definitions:

Definition 1. *Linear mapping is a mapping* $L(x)$ *over* $GF(2)^n$ *that satisfies* $\forall\ x, y \in GF(2)^n\ :\ L(x+y) = L(x) + L(y)$.

Definition 2. *Affine mapping is a mapping* $A(x)$ *over* $GF(2)^n$ *such that* $A(x) = L(x) + c, c \in GF(2)^n$ *and* $L(x)$ *is the linear mapping.*

3.3.1 AES-128

A brief introduction of AES is required for further understanding of the whitebox implementation and the implementation based on dual AES in section 4.

AES-128 is a symmetric, iterated, block cipher that maps $128 \rightarrow 128$ bits (block length) using a 128-bit encryption key. It has 10 rounds and operates over 4×4 byte array. AES

works with GF (2^8) and operations used in AES have quite algebraic nature. For each round is generated a key material called round keys by key-schedule routine from encryption key.

The key-schedule is not an important operation from our perspective and can be abstracted in further explanations. Important fact about the key-schedule is that it is reversible i.e. if we have round keys from 2 consecutive rounds it is possible to derive all other round keys, even the encryption key (the encryption key is used as a round key in the first round).

Besides the key-schedule there are 4 operations used in main AES body:

- *AddRoundKey* adds round keys to the state array. It is simple XOR of two 4×4 byte arrays.
- *ShiftRows* performs a simple shift of each row of the state array to the left, by the row index (indexing from 0).
- *SubByte* is $8 \rightarrow 8$ bijection, the only non-linear operation, performing *confusion*[2]. It uses inversion in GF (2^8).
- *MixColum* is the main *diffusion*[3] operation. It corresponds to a matrix multiplication in GF (2^8). State array column is multiplied from left by MixColumn 4×4 matrix (also denoted as MC). Due to this operation, after one round of the cipher, one byte of the state array depends on 4 bytes of the input state array.

3.3.2 AES table implementation

Algorithm 1 is AES in a form suitable for the whitebox implementation, i.e. some operations were regrouped in such a way that *AddRoundKey* is right before *SubByte* operation. This enables to merge these two operations to one 8×8 look-up table as is described in equation 3.1. Resulting tables following this construction are called T-boxes.

$$\begin{aligned} T^r_{i,j}(x) &= S\left(x \oplus k^r_{i,j}\right) \\ T^9_{i,j}(x) &= S\left(x \oplus k^9_{i,j}\right) \oplus k^{10}_{i,j} \end{aligned} \tag{3.1}$$

ShiftRows has no counter-part in table implementation, it is implemented as the way how tables between rounds are connected together (taking *ShiftRows* into account).

The *MixColumn* is problematic to transform into a look-up table since it is $32 \rightarrow 32$ mapping. A naive transformation would take $2^{32} \cdot 4 = 16$ GB. Thus linearity of a matrix multiplication is used, as illustrates equation 3.3.

2. *Confusion* is a property of secure cipher defined by Shannon [41], it should make relation between ciphertext and symmetric encryption key complex.
3. *Diffusion* is a property of secure cipher defined by Shannon [41], it should make relation between ciphertext and plaintext complex, dissipating small change into large range. Diffusion contributes to the avalanche effect.

Algorithm 1 AES algorithm, form suitable for whitebox implementation

```
1: function AES(plaintext, k)                          ▷ k is array of round keys
2:     state ← plaintext                               ▷ state is 4 × 4 byte array
3:     for r ← 0, to 8 do
4:         ShiftRows(state)
5:         AddRoundKey(state, k_r)
6:         SubByte(state)
7:         MixColumn(state)
8:     end for
9:     ShiftRows(state)
10:    AddRoundKey(state, k_9)
11:    SubByte(state)
12:    AddRoundKey(state, k_10)
13:    return state
14: end function
```

Let

$$MC = \begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} \tag{3.2}$$

Then from linearity holds:

$$\begin{aligned} MC \cdot \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} &= MC \cdot \begin{bmatrix} x_0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \oplus MC \cdot \begin{bmatrix} 0 \\ x_1 \\ 0 \\ 0 \end{bmatrix} \oplus MC \cdot \begin{bmatrix} 0 \\ 0 \\ x_2 \\ 0 \end{bmatrix} \oplus MC \cdot \begin{bmatrix} 0 \\ 0 \\ 0 \\ x_3 \end{bmatrix} \\ &= x_0 \cdot \begin{bmatrix} a_{0,0} \\ a_{1,0} \\ a_{2,0} \\ a_{3,0} \end{bmatrix} \oplus x_1 \cdot \begin{bmatrix} a_{0,1} \\ a_{1,1} \\ a_{2,1} \\ a_{3,1} \end{bmatrix} \oplus x_2 \cdot \begin{bmatrix} a_{0,2} \\ a_{1,2} \\ a_{2,2} \\ a_{3,2} \end{bmatrix} \oplus x_3 \cdot \begin{bmatrix} a_{0,3} \\ a_{1,3} \\ a_{2,3} \\ a_{3,3} \end{bmatrix} \\ &= x_0 \cdot MC_0 \oplus x_1 \cdot MC_1 \oplus x_2 \cdot MC_2 \oplus x_3 \cdot MC_3 \end{aligned} \tag{3.3}$$

Thus the $32 \rightarrow 32$ linear mapping represented by a matrix multiplication is decomposed to four $8 \rightarrow 32$ look-up tables connected by three 32-bit XOR tables. The tables performing this operation are denoted as T_y tables in a further text.

Figure 3.1 illustrates an evaluation of the AES round function using look-up tables for one column of state array.

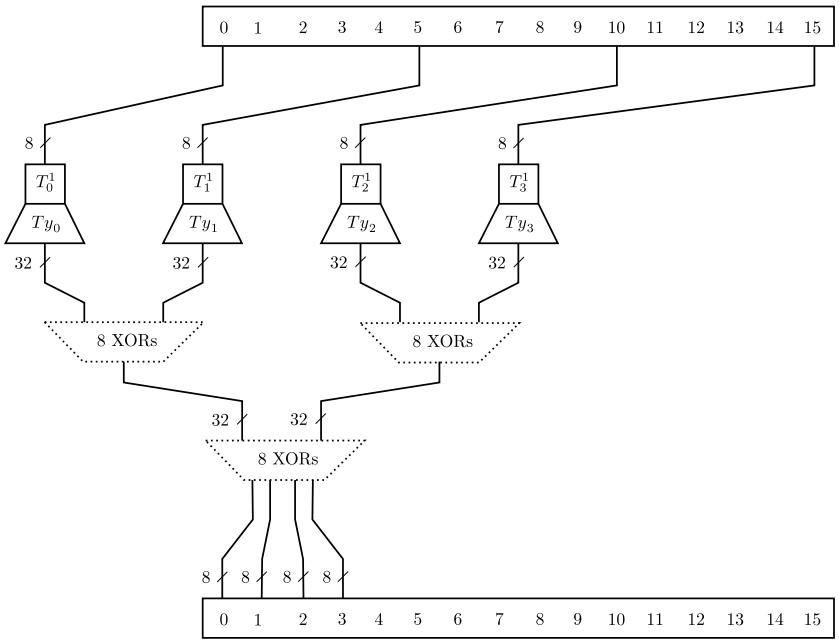


Figure 3.1: Table AES implementation - rounds 2–8, taken from [42]

3.3.3 Whitebox AES

As mentioned before, special techniques are needed to protect look-up tables from algebraic attacks. The most important ones are described in following paragraphs.

Input/output bijections. One of the techniques used in whitebox implementation is a use of *input/output bijections* (also abbreviated as IO bijections). According to Chow *et al.* [4] IO bijection is a random $n \to n$ bijection[4]. Consider $n \to n$ IO bijections $F_1, \ldots, F_k$, they can be *concatenated* to form a $kn \to kn$ bijection $F_1 || \ldots || F_k$. This concatenation enables to build a large bijections with small look-up tables. In a further explanations the size of basic IO bijection is $4 \to 4$.

Consider the table implementation of AES as mentioned in the section 3.3.2. To protect the tables, each table is wrapped by the concatenated IO bijections in such a way the composition of two connected tables cancels the effect of IO bijections as illustrates equation 3.4.

4. usually $n = 4$

$$T' = g \circ T \circ f^{-1} \tag{3.4a}$$
$$R' = h \circ R \circ g^{-1} \tag{3.4b}$$
$$R' \circ T' = \left(h \circ R \circ g^{-1}\right) \circ \left(g \circ T \circ f^{-1}\right) = h \circ (R \circ T) \circ f^{-1} \tag{3.4c}$$

Where $T,\ R$ are look-up tables, $g,\ h$ are IO bijections, realizing confusion step and making an analysis of a single table harder.

Mixing bijections. Another whitebox building block is a *mixing bijection*. It is a linear transformation (represented as a multiplication by a non-singluar mixing bijection matrix) that realizes the diffusion. It is used together with the IO bijections to increase a security level of the concatenated bijections, since it diffuses a single change in the one sub-bijection to the whole range of the concatenated bijection. In order to fulfill this purpose properly, the mixing bijections have to be constructed in a special way. Zhou *et al.* in [43] describe the algorithm that generates large random non-singular matrices with blocks of a full-rank. The size of the sub-blocks is 4 what matches the size of basic IO bijection what gives good diffusion properties for the concatenated bijections.

Note that full-rank property of matrix blocks provides good level of the diffusion. It lies somewhere between two extreme cases: (1) random non-singular matrices without any requirements on a diffusion power and (2) parity-check matrices of MDS[5] codes that have an optimal diffusion power (for a linear transformation), but it is harder to generate them systematically. Note that MDS matrices are often used as a diffusion element in ciphers.

External input/output encoding. Consider AES protected with aforementioned whitebox techniques. A possible place where to attack is an input and an output of the algorithm, since it is not protected from a whitebox attack. To mitigate this weakness Chow *et al.* also introduces an *external encoding* that wraps whole AES. Usually the cipher is not a standalone element, but a part of the system. This technique helps to tie AES to its context and to prevent from using the cipher separately as an oracle.

Whitebox implementation computes:

$$H \circ AES \circ G^{-1} \tag{3.5}$$

If AES is used as a part of the system, a previous element has to apply transformation G on data before passing them to the WB AES. Similarly, the next element has to apply H^{-1} to cancel the effect of the previous transformation.

Usually G, H are defined as a multiplication by 128×128-bit matrix followed by the $128 \rightarrow 128$ concatenated bijection.

5. (n, k, d)-code is Maximum Distance Separable if $d = n-k+1$, `http://www.mth.msu.edu/~jhall/classes/codenotes/Linear.pdf`

Table types. To fully transform AES to the WB AES the following 4 table types are needed. As mentioned above, the external encoding uses a 128×128 matrix multiplication to protect the input and the output of the cipher. The matrix decomposition technique introduced in section 3.3.2 is used to make table implementation feasible. Figure 3.2 depicts $8 \rightarrow 128$ table used in the first and the last round for the external encoding.

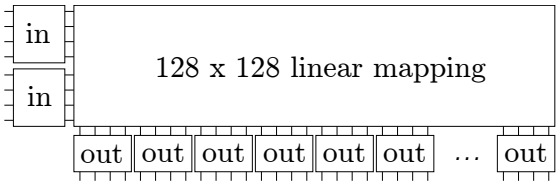


Figure 3.2: Table type I, $8 \rightarrow 128$ mapping, used for external encoding, taken from [44]

Recall already mentioned T-boxes in section 3.3.2 that perform *SubByte* and *AddRoundKey* operations. The *MixColumn* operation is implemented by the matrix multiplication decomposition, these 8×32 tables are called T_y boxes. In order to save space T and T_y boxes are composed to resulting type II table as illustrates figure 3.3.

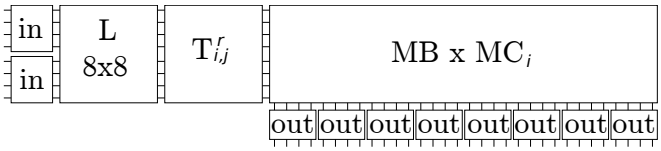


Figure 3.3: Table type II, $8 \rightarrow 32$ mapping, taken from [44]

Note 32×32 mixing bijection added after the *MixColumn* operation to increase a diffusion. To cancel this transformation the table type III is used as illustrates figure 3.4. Also recall the matrix multiplication decomposition requires addition operation. This is done by exclusive-OR (XOR), a cascade of table type IV is used for this purpose. Note that performing XOR by table look-up is rather ineffective, but it is required to protect look-up tables by IO bijections and mixing bijections (to form a protected network of look-up tables without revealing true values on table boundaries).

Overall scheme. Figure 3.5 illustrates how the round function of whitebox AES implementation for one column looks, using aforementioned tables.

On the diagram in figure 3.5 are the following mappings:

- MB stands for Mixing Bijection. It is 32×32 matrix with coefficients from GF(2), representing linear transformation over GF $(2)^8$. $\mathrm{MB}^{-1}_{\{0,1,2,3\}}$ are then column stripes of

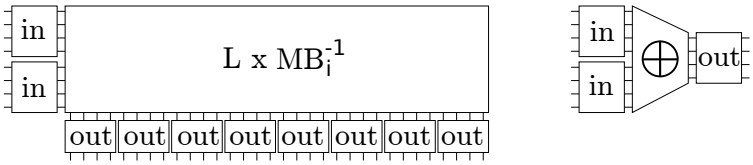


Figure 3.4: Table type III and IV, taken from [44]

the corresponding MB inverse matrix (the matrix multiplication decomposition). This transformation cancels out within one round.

- L stands also for Mixing Bijection but in this case it is 8×8 matrix.
- Q,P. These are random IO bijections. It holds that $P_{i,j}^{r+1} \circ Q_{i,j}^{r} = id$

3.3.4 Cipher invertibility

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

As an example take AES whitebox implementation. Inverting the cipher in the 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 the blackbox, advantageous for an attacker. 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. We propose some improvement addressing this problem in section 5.3.

There are 4 columns of the state array within one round, independent on each other. Thus the cipher can be easily inverted running it backwards and finding an inversion for the each column separately, assuming 128×128 external mixing bijections are I_{128}. Thus the task is to find an inversion of a 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 the wanted result.

A computational complexity to invert the cipher is $10 \cdot 4 \cdot 2^{32}$ operations [6].

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 an algorithm that inverts the WB AES, i.e. performing plaintext recovery attack. In the non-optimized version it takes 13 hours on my hardware[7] to invert the WB AES with negligible memory requirements.

6. 10 (rounds) $\cdot$4 (columns) $\cdot 2^{32}$ (column function input space size)
7. For hardware specifications see A.2

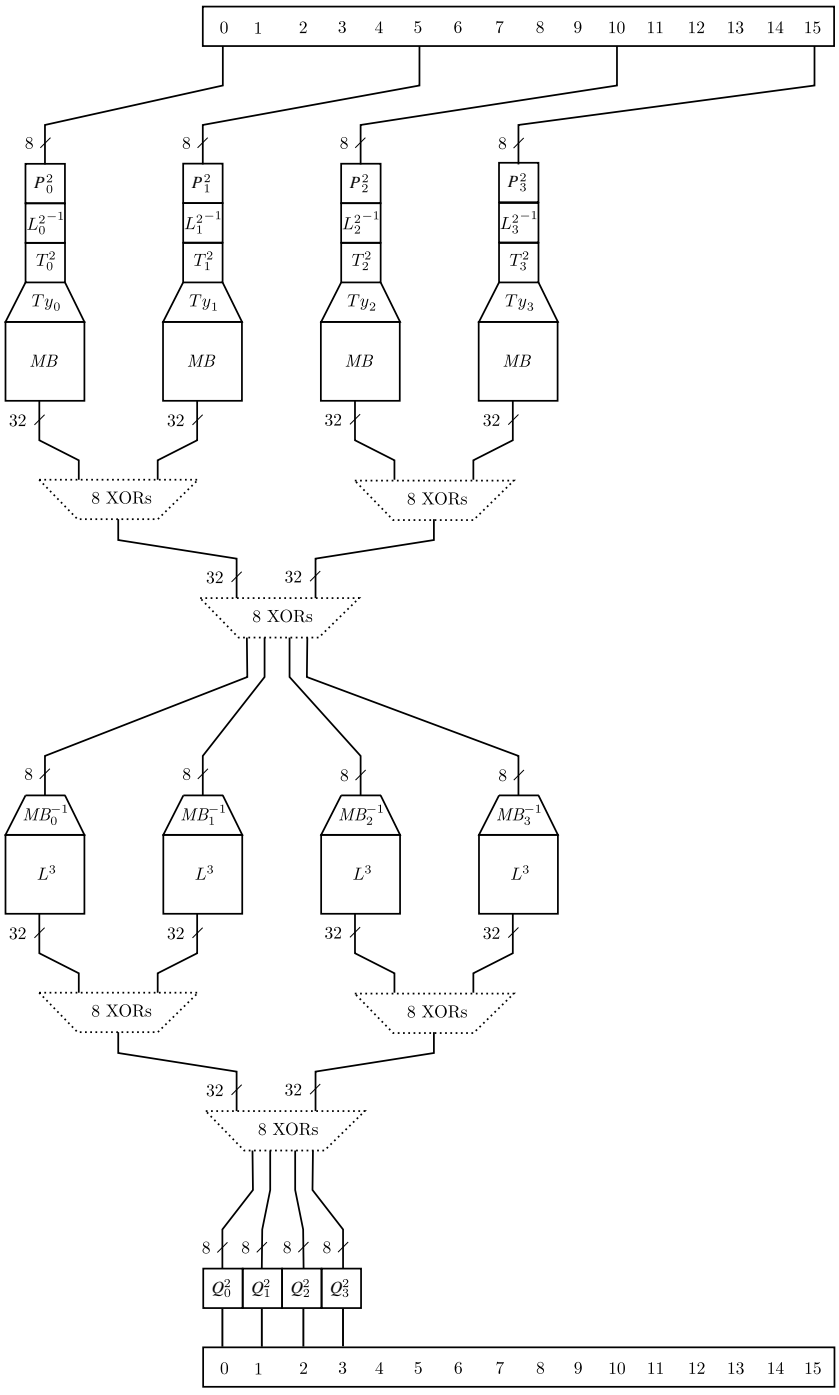


Figure 3.5: The whitebox AES implementation - round #2, based on diagram from [42]

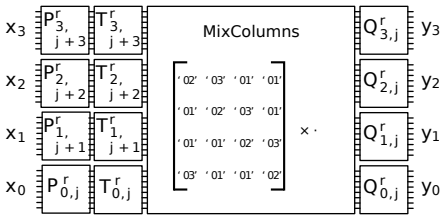


Figure 3.6: AES round function from the BGE attack perspective, taken from [5]

3.4 The BGE attack

The paper [5] by Billet *et al.* demonstrated that the whitebox AES implementation, as described in the previous section, is vulnerable to an algebraic attack. The attack is named after initials of authors, the BGE attack. It recovers the symmetric key from the whitebox AES implementation and negligible memory requirements in 2^{30} computational steps.

The BGE attack does not analyze look-up tables locally, but instead the whole AES round function is analyzed as a single look-up table. This has few benefits:

- the structure of the round function is fixed and well-known, thus it is easy to model it as an algebraic equation.
- 32×32 mixing bijections MB are canceled within one round, so they can be neglected.
- 8×8 mixing bijection L can be easily merged with the IO bijections on round boundary, what simplifies further algebraic analysis.

Figure 3.6 illustrates the round function of AES for one column from the BGE attack perspective. Depicted function can be described as a mapping $(x_0, x_1, x_2, x_3) \xrightarrow{R_j^r} (y_0, y_1, y_2, y_3)$, $j = 0, \ldots, 3$.

Since the transformation L, L^{-1} is performed byte-wise on the state array, it can be composed with corresponding IO bijections, as in equation 3.6.

$$Q_{i,j}^{r\,\prime} = Q_{i,j}^r \circ L_j^{r+1} \tag{3.6a}$$

$$P_{i,j}^{r\,\prime} = (L_j^r)^{-1} \circ P_{i,j}^r \tag{3.6b}$$

The IO bijections are generally non-linear, thus composing then with an another linear transformation (L) results again in a non-linear bijection. Note that in the BGE attack, IO bijections are considered to be general $8 \to 8$ bijections, neglecting the fact they are concatenated from 2 $4 \to 4$ bijections what allows composition with L.

The BGE attack proceeds in three steps:

1. Transform the non-linear IO bijections $Q^r_{i,j}$ to unknown GF (2)-affine transformation (i.e. determine a non-linear part up to unknown affine part).

2. Fully determine $Q^r_{i,j}$ bijection, using the algebraic analysis and the known form of the round function.

3. Obtain round keys from 2 consecutive rounds and recover the symmetric key using reversibility of the AES key-schedule.

3.4.1 Recovering non-linear parts

This step is particularly interesting because of its universality. It could be applied on a different whitebox implementation.

The main goal of this step is to recover non-linear parts of $(Q^r_i)_{i=0,\dots,3}$. Consider y_0 as a function of (x_0, x_1, x_2, x_3). If x_1, x_2, x_3 are fixed as constants c_1, c_2, c_3, it is easy to see that there exist $\alpha, \beta_{c_1,c_2,c_3} \in \mathrm{GF}\left(2^8\right)$ such that the following holds:

$$\begin{aligned} y_0\left(x_0, c_1, c_2, c_3\right) &= Q^r_{0,j}\left(\alpha T^r_{0,j}\left(P^r_{0,j}\left(x\right)\right) \oplus \beta_{c_1,c_2,c_3}\right) \\ &= Q^r_{0,j} \circ \oplus_{\beta_{c_1,c_2,c_3}} \circ\, \alpha \cdot T^r_{0,j} \circ P^r_{0,j}\left(x\right) \end{aligned} \tag{3.7}$$

The function y_0 from the equation 3.7 now takes only 256 input values, thus the functions y_0 and y_0^{-1} can be easily evaluated (as look-up tables). For the rest of the chapter consider constants c_2, c_3 fixed, without loss of generality let $c_2 = c_3 = 0$. For the simplification the r superscript is dropped from the equations if it is clear from the context.

Now assume $c_1' \neq c_1$, also denote $\beta_0 = \beta_{c_1,c_2,c_3}, \beta_1 = \beta_{c_1',c_2,c_3}$, it is visible that:

$$\begin{aligned} &y_0\left(x_0, c_1', c_2, c_3\right) \circ y_0^{-1}\left(x_0, c_1, c_2, c_3\right) = \\ &= \left(Q_{0,j} \circ \oplus_{\beta_1} \circ\, \alpha \cdot T_{0,j} \circ P_{0,j}\right) \circ \left(P^{-1}_{0,j} \circ \left(\alpha \cdot T_{0,j}\right)^{-1} \circ \oplus_{\beta_0} \circ\, Q^{-1}_{0,j}\right) \\ &= Q_{0,j} \circ \oplus_{\beta_1} \circ \oplus_{\beta_0} \circ\, Q^{-1}_{0,j} \\ &= Q_{0,j} \circ \oplus_{(\beta_0 \oplus \beta_1)} \circ\, Q^{-1}_{0,j} \end{aligned}$$

Thus by fixing c_1 and iterating c_1' we obtain the set of 256 bijections, represented as look-up tables, of the form $Q_{0,j} \circ \oplus_\beta \circ\, Q^{-1}_{0,j}$, where β takes all values from $\mathrm{GF}\left(2^8\right)$.

Theorem 1. *Given a set of functions $\mathcal{S} = \{Q \circ \oplus_\beta \circ\, Q^{-1}\}_{\beta \in GF(2^8)}$ given by values, where Q is a permutation of $GF\left(2^8\right)$ and the $\oplus_\beta$ is the translation by β in $GF\left(2^8\right)$, one can construct a particular solution $\widetilde{Q}$ such that there exists an affine mapping A so that $\widetilde{Q} = Q \circ A$.*

For the proof see the original paper [5]. In order to recover a non-linear part of the IO bijection it is needed to generate the set $\mathcal{S}$ (by evaluating y_0 functions) and to apply the theorem.

Note that the set $\mathcal{S}$ contains 256 functions, but only as look-up tables so the β is unknown. One easily verifies the set $\mathcal{S}$ together with the operation of a function composition form a group.

Observe there exists isomorphism φ :

$$\varphi : \begin{array}{ccc} \mathcal{S} & \longrightarrow & \mathrm{GF}(2)^2 \\ Q \circ \oplus_\beta \circ Q^{-1} & \longmapsto & [\beta] \end{array}$$

Since we don't know the values β for the functions in $\mathcal{S}$ the φ cannot be constructed directly, however it is possible to recover φ up to unknown linear transformation L i.e. $\psi = L^{-1} \circ \varphi$. Without going into the details, the ψ is determined by finding base functions $f_1, \ldots, f_8$ that span the whole $\mathcal{S}$ under composition. Mapping ψ is then constructed by assigning standard basis vectors $e_1, \ldots, e_8 \in \mathrm{GF}(2)^8$ to these functions.

It is then possible to obtain $\widetilde{Q}$ by using the mapping ψ from the theorem 1 according to the following formula:

$$\widetilde{Q}(\psi(g)) = g(0), \ g \in \mathcal{S} \tag{3.9}$$

Note that the value $g(0)$ is known, the function was evaluated during ψ construction. The term $\psi(g)$ is also easy to evaluate, since we have constructed mapping ψ. Thus in order to recover $\widetilde{Q}$ as a look-up table we just run over the set $\mathcal{S}$ and compute the values of mapping $\widetilde{Q}$ according to the equation 3.9.

3.4.2 Recovering the symmetric key

As the rest of the BGE attack is very AES and implementation specific, it is not covered in detail. In this phase, we are still in situation as is depicted in figure 3.6 with a difference $(Q_i^r)_{i=0,\ldots,3}$ and $\left(P_i^{r+1}\right)_{i=0,\ldots,3}$ are affine, still matching, bijections. This fact makes further algebraic analysis of round function possible.

Consider following useful proposition:

Proposition 1. *For any pair (y_i, y_j) exists a unique linear mapping L and a unique constant c such that:*

$$\forall x_0 \in GF\left(2^8\right) : \ y_i(x_0, 0, 0, 0) = L(y_j(x_0, 0, 0, 0)) \oplus c \tag{3.10}$$

By using Theorem 1 one can obtain the linear parts of Q_1, Q_2, Q_3 from the knowledge of the Q_0 linear part in 2^{16} steps by simple running over c and testing the resulting mapping for linearity in 2^8 steps. Thus the rest of the attack focuses on Q_0 determination.

First of all, the linear part of Q_0 up to $[\gamma], \gamma \in \mathrm{GF}\left(2^8\right) \setminus \{0\}$ is determined, where $[\gamma]$ denotes the matrix in $\mathrm{GF}(2)$ corresponding to multiplication by γ in $\mathrm{GF}\left(2^8\right)$. For more details see appendix A.4.

Then γ and a constant part q_0 of Q_0 are determined. During these steps, the public knowledge of S-boxes and MixColumn matrix coefficients is used.

With completely determined Q_0 the round keys are extracted. The attack ends using the fact the key-schedule of AES is reversible, so from two consecutive round keys one can derive original symmetric key.

4 WBCAR AES using dual ciphers

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

In this chapter we describe a whitebox scheme proposed in [1] that make use of AES dual ciphers. It is supposed that using dual AES, different in each round, will increase security of the whitebox implementation of the cipher. The paper says that this modification results in raising BGE attack [5] complexity to 2^{91} computational steps, making it unfeasible to perform it in practice. It is shown in section 4.2 that this assumption is false due to existence of an attack we performed.

4.1 Scheme

In the original paper [1] the explanation how to obtain dual AES ciphers and how to construct mapping from one to another is not given. This is important part since it plays a crucial role in a proof that this scheme is vulnerable. At first is described the generalization of the AES and how to construct mappings between them.

4.1.1 Ciphers duality

This section introduces a notion of dual ciphers used in this chapter. Dual ciphers have interesting properties and are used in a construction of the scheme that is described by this chapter. Consider the following definition.

Definition 3. *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.

4.1.2 Generic AES

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

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

Default AES (as in NIST standard [45]) in this notation is represented as $\{\{11B\}_x, \{03\}_x\}$. Polynomial is expressed in the hexadecimal notation. Each bit corresponds to a polynomial

coefficient, LSB[1] 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 the duality mentioned in the next section.

4.1.3 Generic AES duality

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

All elements of the field GF $(2^8) = \{01, 02, \ldots, FF\}$ can be expressed in terms of the generator β, 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 $\beta^i \in$ GF $(2^8) \cong$ 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 *ShiftRows*, *AddRoundKey* functions are quite simple and remain same after AES generalization. *SubByte*, *MixColumn* are affected by generalization. For proper understanding of construction generic AES, description follows.

1. least significant bit

SubByte

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

where x^{-1} is element inverse in GF (2^8), A is 8×8 matrix over GF(2), c is column vector $\mathrm{GF}(2)^8$. A, c are constants defined in NIST standard [45].

Equation for affine transformation used in S-box:

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

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

MixColumn

- columns of the state array 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 $x_i, y_i \in \mathrm{GF}\left(2^8\right)$.

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

SubByte

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

MixColumn MixColumn matrix coefficients are expressed in terms of the 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 the 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 the equations for the first column of the 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 the 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 Dual AES

The paper describing the AES whitebox implementation with use of the dual AES [1], claimed that this implementation should be more difficult (in terms of time complexity) to break using the BGE attack on the whitebox AES. On the figure 4.1 is scheme for one round, one column of state array, the 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.

By changing irreducible polynomial and generator we obtain $30 \cdot 8 = 240$ different generic AES ciphers. The bigger is set of possible ciphers to use, the more difficult is for attacker to break the dual scheme, since he has to try all possible combinations, according to [1]. But in [1] is assumed there are 61200 different generic AES ciphers but it is not said how they are constructed and how such construction influences the whitebox implementation.

In order to generate 61200 different AES representations it is needed to study AES S-Box affine self-equivalences [46].

Definition 4. *$n \times n$-bit mapping S is affine self-equivalent if there exist $n \times n$ affine relations A_1, A_2 such that:*

$$A_2 \circ S \circ A_1 = S \tag{4.11}$$

In [46] are published effective algorithms for finding linear and affine equivalences for permutations (S-boxes). We have implemented them [2] to verify number of self-equivalences for AES S-Boxes for an encryption and a decryption algorithm. This algorithm can be further used to study modified S-boxed or basic building blocks of the cipher (see chapter 5.1) for equivalences, what can lead to revealing potential weaknesses.

The algorithm found 2040 affine self-equivalences. Together with 30 possible irreducible polynomials there are 61200 dual ciphers. Biryukov *et al.* derived general expressions for A_1, A_2:

$$A_1(x) = [a] \cdot Q^i \cdot x \tag{4.12a}$$

$$A_2(x) = A\left(Q^{-i} \cdot [a] \cdot A^{-1}(x)\right) \tag{4.12b}$$

Where

- A is a fixed affine mapping from AES S-box definition (see section 4.1.4), A^{-1} it's inverse.
- $[a]$ denotes 8×8 matrix with coefficients from $\mathrm{GF}(2)$ representing a *multiplication* by $a \in \mathrm{GF}(2^8) \setminus \{0\}$ in $\mathrm{GF}(2^8)$. From the fact $\mathrm{GF}(2)^8 \simeq \mathrm{GF}(2^8)$ (all finite

2. path: implementation/LinearAffineEq.{h,cpp}

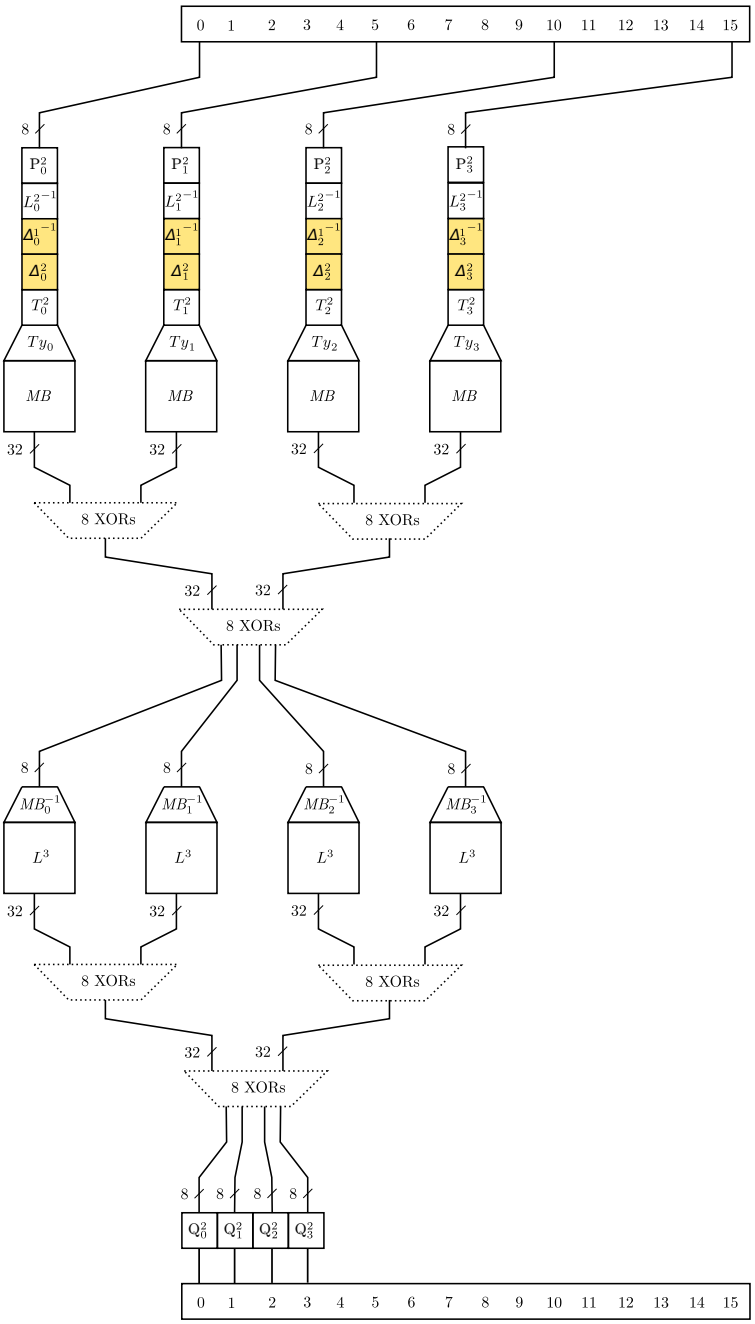


Figure 4.1: Whitebox Dual AES implementation - round #2, based on diagram from [42]

fields with same number of elements are isomorphic), the multiplication is linear transformation in GF (2^8), it can be expressed in a matrix form. Refer to A.4 to see how to construct such matrix.

Q denotes 8×8 matrix with coefficients from GF (2) representing *squaring* in GF (2^8). Squaring is a *linear* operation in GF (2^8) so it is possible to represent it as a matrix. Refer to appendix A.3 for the construction and the proof. Note that $Q^8 = I \Rightarrow Q^{-i} = Q^{8-i}$. Thus there are 8 different powers of Q, $Q^i,\ i \in [0, 7]$.

It is visible that with the general expressions we can obtain $8 \cdot 255 = 2040$ different $A_1,\ A_2$ relations[3] confirming the output of the algorithm.

Observe that by inserting $A_1,\ A_2$ before and after S-Box the cipher is not affected. Note that A_1 is linear, this fact can be used to push the input mapping A_1 through the mixing layer and combine it with A_2 from the previous round, then we obtain 2040 different AES tables evaluating the same function. Note that they are dual, but with identity isomorphism, it is just a different table implementation of AES, thus $\Delta = I$. This is a potential flaw of the whitebox scheme using dual ciphers, since it does not increase security against known attacks at all. This construction is neglected in the original paper [1].

The layers of mixing bijections (L, MB) cancel between rounds so they can be neglected in pushing A_1 to the previous round. Recall $A_2\left(S\left(A_1\left(x\right)\right)\right) = S(x)$. An input to S-box is output of previous round, so apply A_1 on previous round. It is shown only for one column (equation 4.13), others are analogical. It is easy to verify that $[a] \cdot Q^i \cdot [c] = \left[c^{2^i}\right] \cdot [c] \cdot Q^i$. Redefine $T_r(x) = A_2^r(S(x \oplus k))$ to take the affine relations into account, where k is a particular round key byte. Then we can write:

$$A_1^{r+1} \cdot \begin{bmatrix} 02 \cdot T_r(x) & 01 \cdot T_r(x) & 01 \cdot T_r(x) & 03 \cdot T_r(x) \end{bmatrix}^T \tag{4.13a}$$

$$= \begin{bmatrix} A_1^{r+1}(02 \cdot T_r(x)) & A_1^{r+1}(01 \cdot T_r(x)) & A_1^{r+1}(01 \cdot T_r(x)) & A_1^{r+1}(03 \cdot T_r(x)) \end{bmatrix}^T$$

$$= \begin{bmatrix} 02^{2^i} \cdot A_1^{r+1}(T_r(x)) & 01^{2^i} \cdot A_1^{r+1}(T_r(x)) & 01^{2^i} \cdot A_1^{r+1}(T_r(x)) & 03^{2^i} \cdot A_1^{r+1}(T_r(x)) \end{bmatrix}^T$$

$$= \begin{bmatrix} 02^{2^i} \cdot A_1^{r+1}(T_r(x)) & 01 \cdot A_1^{r+1}(T_r(x)) & 01 \cdot A_1^{r+1}(T_r(x)) & 03^{2^i} \cdot A_1^{r+1}(T_r(x)) \end{bmatrix}^T \tag{4.13b}$$

This gives us different tables for the AES round function when using different $A_1,\ A_2$ relations. Note that table of type II with A_1 pushed from next round is still the same, no matter which form of equation 4.13 is used, due to linearity of A_1. For simplicity in further proofs and description we can assume form 4.13a.

3. by running through possible values for a, i

4.2 Attacking Dual AES scheme

According to [1] the whitebox scheme using dual AES is considered to be more difficult to crack with the BGE attack and thus it is consider safer than an original scheme proposed in [4]. But we show that it is not true. This result has not been published yet.

Proposition 2. *Whitebox Dual AES scheme can be broken with the BGE attack with the same time complexity as the whitebox AES scheme.*

Proof. Let's define a round function for the whitebox AES and for the whitebox dual AES and compare them. Note that MB mixing bijections are left out since their effect is canceled within one round. If we also assume use of $A_1,\ A_2$ affine relations, they can be merged together with L mixing bijection to input/output encodings. Thus for simplicity $A_1,\ A_2$ is omitted from the proof (it is clearly visible it does not increase resistance against the BGE attack - input/output encodings are fully determined in the attack).

Round function - whitebox AES. Q', P' functions represent IO bijections, for details see [4, 5].

$$y_{i,j}\left(x_{i,0}, x_{i,1}, x_{i,2}, x_{i,3}\right) = Q_{i,j}^{r\,\prime}\left(\bigoplus_{l=0}^{3} \alpha_{l,j} \cdot S\left(P_{i,l}^{r\,\prime}\left(x_{i,l}\right)\right)\right) \tag{4.14a}$$

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

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

Round function - whitebox dual AES. For simplicity define:

$$P_{i,j}^{r\,\prime\prime} = \left(\Delta^{r-1}\right)^{-1} \circ P_{i,j}^{r\,\prime} = \left(\Delta^{r-1}\right)^{-1} \circ \left(L_j^r\right)^{-1} \circ P_{i,j}^{r} \tag{4.15}$$

Also we have to distinguish in which dual AES is element encoded, so define x^{Δ} as an element of the dual AES which base change matrix is Δ from standard AES field. The same holds for inversion operation $^{-1}$. Denote $^{-1^{\Delta}}$ inversion in field of dual AES which has base change matrix Δ.

For simplicity assume that $\Delta = \Delta^r$ for round r if it is obvious from context and is not

defined otherwise. According to figure 4.1 the equation for one round is:

$$
\begin{aligned}
& y_{i,j}\left(x_{i,0}, x_{i,1}, x_{i,2}, x_{i,3}\right) = \\
& = Q_{i,j}^{r\,\prime}\left(\bigoplus_{l=0}^{3}\Delta(\alpha_{l,j})\cdot\left(\Delta\times A\times\Delta^{-1}\left(\left(\Delta\circ P_{i,l}^{r\,\prime\prime}\left(x_{i,l}\right)\oplus\Delta\left(k_{i,l}\right)\right)^{-1^{\Delta}\,\mathrm{GF}\left(2^8\right)}\right)\oplus\Delta\left(c\right)\right)\right) \\
& = Q_{i,j}^{r\,\prime}\circ\Delta\left(\bigoplus_{l=0}^{3}\alpha_{l,j}\cdot\left(A\times\Delta^{-1}\left(\left(\Delta\circ P_{i,l}^{r\,\prime\prime}\left(x_{i,l}\right)\oplus\Delta\left(k_{i,l}\right)\right)^{-1^{\Delta}\,\mathrm{GF}\left(2^8\right)}\right)\oplus c\right)\right) \\
& = Q_{i,j}^{r\,\prime}\circ\Delta\left(\bigoplus_{l=0}^{3}\alpha_{l,j}\cdot\left(A\times\Delta^{-1}\left(\left(\Delta\left(P_{i,l}^{r\,\prime\prime}\left(x_{i,l}\right)\oplus k_{i,l}\right)\right)^{-1^{\Delta}\,\mathrm{GF}\left(2^8\right)}\right)\oplus c\right)\right) \qquad (4.16a) \\
& = Q_{i,j}^{r\,\prime}\circ\Delta\left(\bigoplus_{l=0}^{3}\alpha_{l,j}\cdot\left(A\times\Delta^{-1}\left(\Delta\left(P_{i,l}^{r\,\prime\prime}\left(x_{i,l}\right)\oplus k_{i,l}\right)^{-1\,\mathrm{GF}\left(2^8\right)}\right)\oplus c\right)\right) \qquad (4.16b) \\
& = Q_{i,j}^{r\,\prime}\circ\Delta\left(\bigoplus_{l=0}^{3}\alpha_{l,j}\cdot\left(A\times\left(\left(P_{i,l}^{r\,\prime\prime}\left(x_{i,l}\right)\oplus k_{i,l}\right)^{-1\,\mathrm{GF}\left(2^8\right)}\right)\oplus c\right)\right) \\
& = Q_{i,j}^{r\,\prime}\circ\Delta\left(\bigoplus_{l=0}^{3}\alpha_{l,j}\cdot\left(A\times\left(\left(P_{i,l}^{r\,\prime\prime}\left(x_{i,l}\right)\oplus k_{i,l}\right)^{-1\,\mathrm{GF}\left(2^8\right)}\right)\oplus c\right)\right) \\
& = Q_{i,j}^{r\,\prime}\circ\Delta\circ R_{i,j}^{\prime}\left(x_{i,0}, x_{i,1}, x_{i,2}, x_{i,3}\right) \qquad (4.16c)
\end{aligned}
$$

Now it is easy to see whitebox dual AES correctness, moreover it is visible that the same attack breaking the whitebox AES breaks the whitebox dual AES scheme as well

A transformation from 4.16a to 4.16b is possible due to base change matrix properties and fields we are computing in.

$$\forall x, y\in\mathrm{GF}\left(2^8\right)\ :\ y=x^{-1}\Rightarrow\Delta y=(\Delta x)^{-1^{\Delta}} \qquad (4.17)$$

Note that element inversion GF (2^8) has changed from one field to another.

Now if we compare equations 4.14c and 4.16c, they are very similar, the only difference here is the application of the base change matrix Δ.

Here we can do the similar thing we did in equations 3.6, 4.15 where we composed two transformations, non-linear and linear to non-linear transformation, with equation 4.16c.

We can thus define:

$$
\begin{aligned}
Q_{i,j}^{r\,\prime\prime} = Q_{i,j}^{r\,\prime} = Q_{i,j}^{r}\circ\Delta = Q_{i,j}^{r}\circ L_{j}^{r+1}\circ\Delta \qquad (4.18a) \\
y_{i,j}\left(x_{i,0}, x_{i,1}, x_{i,2}, x_{i,3}\right) = Q_{i,j}^{r\,\prime}\circ\Delta\circ R_{i,j}^{\prime}\left(x_{i,0}, x_{i,1}, x_{i,2}, x_{i,3}\right) \qquad (4.18b) \\
y_{i,j}\left(x_{i,0}, x_{i,1}, x_{i,2}, x_{i,3}\right) = Q_{i,j}^{r\,\prime\prime}\circ R_{i,j}^{\prime}\left(x_{i,0}, x_{i,1}, x_{i,2}, x_{i,3}\right) \qquad (4.18c)
\end{aligned}
$$

Now it is evident that equations for whitebox AES 4.14c and 4.18c are the same, the only differences are only in non-linear transformations Q, P (inside R'), but it is important they are both non-linear.

Conclusion is if the attack can break the whitebox AES scheme with the round function 4.14c, it can also break the whitebox dual AES scheme. During the attack is transformation Q fully determined, we verified that if the dual AES scheme is used, transformation Q is of the exact form as described above. □

4.3 Implementation of the cipher

As a implementation part of this thesis, 2 whitebox schemes were implemented. The scheme using dual AES was chosen to be implemented, since in time of making this decision there was no published cryptanalysis of this scheme and it had interesting construction. The fact it is a generalization of the original whitebox AES scheme published by Chow *et al.* is an additional benefit what simplifies the implementation.

4.3.1 NTL library

Recall AES cipher uses finite fields algebra extensively, namely in GF (2) and GF (2^8). For this reason the library NTL[4] was selected as a cornerstone of this implementation. It enables both an effective computation in the finite fields and to write a simple, understandable and easy-to-read source code.

Besides NTL there was no need to use any other math library.

4.3.2 Generic AES

In order to be able to implement the whitebox AES, it was necessary to implement the basic building blocks of the AES as described in previous chapters (such as S-boxes, MixColumn, etc...), using the NTL library. It was preferred to implement it from the scratch, according to the specification, rather than taking existing implementations. The emphasis was put on a simple and extendable source code, rather than optimized and difficult to understand. The implementation is publicly available so that other researchers could use it, e.g. for experimenting with their own transformations. This was also the reason why to write it from the scratch.

The another reason was the fact that AES implementations work with fixed finite field (as generated by an irreducible polynomial and a generator - according to the AES specifications) so the algebraic nature of the operations is not clear from the code anymore, because of various optimizations. Due to the fact dual AES scheme uses generalized AES, it was needed to generate such AES instances with changed irreducible polynomial and generator.

The output from the key-schedule (generated by generic AES) was also required by the

4. NTL is a high-performance, portable C++ library providing data structures and algorithms for manipulating signed, arbitrary length integers, and for vectors, matrices, and polynomials over the integers and over finite fields. `http://www.shoup.net/ntl/`

whitebox implementation, because the S-boxes with an embedded symmetric key are generated during the construction.

4.3.3 Mixing Bijections

As described in section 3.3.3 the mixing bijection matrices have to have special form in order to provide a sufficient level of protection. The algorithm described by Zhou *et al.* in [43] to generate matrices with desired properties was implemented.

This part of the implementation is independent on whitebox context and thus can be used separately, to generate matrices with good diffusion properties.

In order to implement mentioned algorithm it was necessary to extend the NTL library. Namely it was needed to generate such matrices P, Q for a matrix A so that the equation 4.19 holds. The Gauss-Jordan elimination was modified to generate these matrices[5].

$$P \cdot A \cdot Q = \begin{bmatrix} 1 & \cdots & 0 & 0 & \cdots & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & & & \vdots \\ 0 & \cdots & 1 & 0 & \cdots & \cdots & 0 \\ 0 & \cdots & 0 & 0 & \cdots & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & & & \vdots \\ 0 & \cdots & 0 & 0 & \cdots & \cdots & 0 \end{bmatrix} \tag{4.19}$$

It is also needed to generate a random linear transformation in the algorithm. The number of all possible $p \times p$ matrices in $\mathbb{F}_q$ is $q^{p \cdot p}$, while the number of all possible non-singular (invertible) matrices is $\prod_{i=0}^{p-1}(q^p - q^i)$. In our case $q = 2$, $\mathbb{F}_q \equiv \mathrm{GF}(2)$. The table 4.1 shows the probability of randomly generated matrix of particular dimension being non-singluar. For our parameters the value converges around 0.2888 so it is enough to generate a random matrix and test whether it is invertible. Within 4 iterations the non-singular matrix is found with a sufficient probability.

The rest of the algorithm for construction mixing bijections follows the paper by Zhou *et al.* [43].

4.3.4 Linear and affine equivalences

In order to generate 61200 different dual AES ciphers, it is needed to study S-box affine self-equivalences, as mentioned in the [46]. The paper also presents effective algorithms for finding linear and affine equivalences on arbitrary $n \to n$ permutations.

In order to verify the general form of affine relations for AES S-box (see section 4.1.5) the mentioned algorithm was implemented. It is independent on the rest of the implementation so it can be used separately.

5. Matrix P corresponds to multiplication of elementary row matrices that transform matrix A to row-echelon form. The matrix Q corresponds to multiplication by column elementary matrices that reorder the columns in a way the equation 4.19 holds.

dimension	probability
2×2	0.375
4×4	0.30762
8×8	0.28992
16×16	0.2888

Table 4.1: Probability of randomly generated matrix being non-singular

Note the attack on the perturbated whitebox AES scheme [40] and the attack on the Xiao *et al.* implementation [38] use this algorithm so it is particularly interesting and useful tool. It can be also used to analyze properties of new S-boxes proposed in sections 5.1 and 5.4 during a further research.

4.3.5 Whitebox AES

The implementation exactly follows description given in the previous chapters. Detailed description of the implementation is out of scope of this thesis, for further details, please, refer to the source code.

It is important to emphasize the implementation is not optimized for speed. The main focus was on the code readability and simplicity.

One can choose several options how to generate whitebox AES. It is possible to enable/disable[6] the following protections (as described in sections 3.3.3 and 4):

1. 4×4 IO bijections
2. external input/output encoding
3. 8×8 L mixing bijection
4. 32×32 MB mixing bijection
5. irreducible polynomial change in each column (to generate a dual cipher)
6. usage of affine $A_1,\ A_2$ relations (to generate a dual cipher)

The algorithm for generating IO bijections in a networked manner, so the whole cipher works in the same way as without them, takes the substantial part of the source code. It was non-trivial to design this approach of generating and connecting bijections together. The main idea of the approach is that 4×4 bijections are generated at once, each having its own unique identifier. There is a map of the IO bijections describing particular relations between

6. If the feature is disabled it means the particular transformation is identity

them for each look-up table in the cipher. This map is constructed in a similar way as the encryption algorithm works when using look-up tables for encryption. It is generated at runtime, but is constant for the scheme and thus can be cached to speed-up the generation algorithm. The generating of tables for the whitebox AES itself is then very simple, the map is used to determine the IO bijection identifier that should be used for the particular table that is being generated at the moment.

The emphasis was also put on the comments in the source code. The details of every important step/block are described in the comments in a detail. The source code comments also contains diagrams, algebraic equations and important details about the implementation.

The implementation is partially influenced by ideas used in similar whitebox implementation by Petr Švenda [7]. For more details, please, refer to the implementation.

4.3.6 Evaluation

The benchmark was implemented as a part of the implementation[8] to measure the real time needed to generate AES whitebox scheme and how it performs in encryption. The results are shown in table 4.2.

Recall implementation is not optimized. The external encoding was disabled so the input can be directly passed to the cipher. Note that even if some protection is disabled, the whole cipher works in the same way, there are still the same tables (but just containing identity transformation) and the same amount of work needs to be done during the encryption/decryption. The only difference is during generating the tables, but the time difference is negligible.

The throughput was tested by measuring the time needed to encrypt files of various sizes. The files were either null (full of zeros) or random (/dev/urandom) to compare results and influences induced by a potential caching.

Also the performance of AES-128-ECB implemented in OpenSSL[9] was measured by encrypting the same files [10].

4.4 Implementation of the attack

The BGE attack as described in the section 3.4 was implemented. The implementation exactly follows the description. Since the attack proceeds in steps, the implementation is quite straightforward. It has the same structure as described in the original paper [5].

The detailed description of the attack implementation is out of scope of this thesis. For the implementation details of the attack please refer to the attack source codes.

---

7. `http://www.fi.muni.cz/~xsvenda/securefw.html`
8. The hardware and software described in appendix A.2 was used to do the tests.
9. http://www.openssl.org/
10. Command used: time openssl aes-128-ecb -k fi.muni.cz -in /tmp/input_file -out /dev/null

Test	Result	Additional info.	OpenSSL result
generate WB AES	8.48 s avg.	100 samples	
throughput, 1 MB random	867.8 KB/s	1.18 s	57283 KB/s
throughput, 10 MB random	1022.977 KB/s	10.01 s	54179 KB/s
throughput, 100 MB random	1028.319 KB/s	99.58 s	74744 KB/s
throughput, 1024 MB random	1124.792 KB/s	932.24 s	63723 KB/s
throughput, 1 MB null	975 KB/s	1.05 s	93091 KB/s
throughput, 10 MB null	969.970 KB/s	10.56 s	68821 KB/s
throughput, 100 MB null	1058.507 KB/s	96.74 s	56356 KB/s
throughput, 1024 MB null	1050.593 KB/s	998.08 s	57283 KB/s

Table 4.2: Results of the benchmark for whitebox AES generator

The demonstration of the attack is included in the implementation. At first the new whitebox AES is generated. Then the set $\mathcal{S}$ is computed by composing functions y_0. The ψ recovering follows. This first step of the attack is performed for rounds $0 \dots 7$. The same would be possible for the round 8 but the implementation would be complicated due to type I tables. When this step finishes, the IO bijections on particular round boundaries are affine and matching. This modified cipher is tested with AES test vectors to verify the attack managed to transform IO bijections properly. The first step takes the 1/3 of the time needed for the attack approximately.

Note it is not required to transform all possible non-linear IO bijection to affine bijections. In order to finish the attack it needs to be done only for 3 consecutive rounds.

Parts of the affine transformations are gradually recovered in the rest of the attack. The attack ends by printing out the symmetric key that was embedded to the whitebox AES.

The benchmark of the attack is part of the implementation. The average time needed to break whitebox AES was 46.8 s (sample size = 10).

5 Scheme improvement

The BGE attack [5] strongly relies on publicly known constants and building blocks used in the AES cipher (MixColumn constants, fixed S-box). This leads us 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 an 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 criterion of AES in order to prevent possible algebraic attacks), but the 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 the BGE attack, this is what makes AES vulnerable to algebraic attacks in the whitebox context.

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

As we seen in the BGE attack it is easy to turn random non-linear IO bijections, protecting table contents from the local analysis, to the affine transformations between rounds of the cipher, what helps further algebraic analysis. From this reason, constructing more complicated non-linear IO bijections is probably not the way how to solve this problem. Whitebox construction should not solely rely on the non-linear IO bijections [5, 36].

Widely used ciphers nowadays were designed also with the goal to be effectively implemented in a hardware (one of the criterion in AES contest). It influenced a design of the cipher building blocks such as size of a diffusion layer, matrix coefficients (lower are better), recycling of cipher primitives in a key-schedule algorithm, etc... Such additional restrictions do not lower security level of the cipher usually, because of the iterating round function, but this can be problem in WBC.

To summarize the weaknesses of the ciphers used to mount the whitebox attacks:

- simple key-schedule (reversible, forward/backward)
- key whitening technique
- publicly known, constant (key-independent) primitives

- weak round function, simple algebraic description
- use of diffusion elements that are easy to remove in the whitebox context (e.g. ShiftRows in the AES)
- diffusion element with low diffusion power with respect to the one round of the cipher, i.e. low dependency of output byte from round function on input bytes (1 output byte of the AES round function depends on 4 input bytes)

As the AES in a standard form is not suitable for the WBC implementation, we are proposing to break the 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 are no non-broken whitebox scheme of AES nowadays, as far as we know. The whitebox schemes using new techniques to protect cipher implementation (but still preserving backward compatibility with the cipher) were successfully cryptanalyzed effectively [5, 36, 40, 38]. In literature was already proposed to design a new cipher with whitebox implementation issues in mind [5, 44].

In our proposal of a new cipher suitable for WBC implementation we are addressing the aforementioned weaknesses. The base of our cipher is AES since it uses widely accepted cipher building blocks and structure, it was extensively analyzed and is in general considered as a secure cipher.

Our modifications of the AES consist of:

- key-dependent S-Boxes
- non-reversible, strong, key schedule
- stronger (output byte dependence on input bytes), key-dependent diffusion element

In our proposal, we took inspiration from the Twofish [31] cipher which has key dependent S-boxes with rather complicated algebraic representations. As emphasized before, the key idea here is to make expressing one round of the cipher as algebraic equations more difficult for an attacker. Our scheme uses Twofish-like key dependent S-boxes in AES algorithm.

5.1 Twofish S-boxes

Here observe the Twofish S-boxes (from [31]) and their algebraic representation.

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

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

Table 5.1: Hash functions performance comparison

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 the Twofish key dependent S-boxes. In particular we use $s_{j,k_a,k_b},\ j \in [0,4]$ instead of four the same constant S-boxes in computation of one column of the state matrix - approach consistent with use of S-boxes Twofish algorithm (diffusion element is connected to the output of S-box).

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

5.2 Key schedule

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

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

We suggest to use *bcrypt* [47] or *scrypt* [48] as a hash function for generating hash chains. We could also use iterated SHA hash function for this purpose, but the main reason we are proposing bcrypt or scrypt is the fact SHA has been very successfully implemented in a hardware (ASICS chips, for Bitcoin[1] mining), with performance 1500 G hashes per second for one device [49]. Also the Bitcoin economy is based on SHA hash function, so there is motivation to build machines specialized on computing SHA hashes - making brute force attacks on SHA easier.

As an illustration consider [50], where M. Gosney used a 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 attack practically unfeasible on the general purpose hardware.

1. `http://bitcoin.org/en/`

From this reason we propose to use hash functions that were specifically designed to take a long time to evaluate (bcrypt, scrypt) and/or to be more difficult to implement in a hardware (contrary to standard hash functions like SHA, MD5).

Note that as key-schedule is done during preparation of whitebox instance (generating tables) of a cipher, there is no time penalty during usage of the implementation.

In AES-128 we have 128 bit cipher key, $k_0, \ldots, k_{15}$. 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 a work load for bcrypt, determines computation complexity of bcrypt hash function. N_{sha} is a 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 the right side stands for i-th byte of resulting hash

key is an encryption key, 128 bits

k^{r-1} is a whole round key for round $r - 1$

$||$ 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 the ciphertext or in particular whitebox cipher instance.

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

1. derive the encryption key from two (or more) consecutive round keys. This results from the infeasability of reversing hash chain. We are also using computational intensive hash function so even brute-forcing is unfeasible.

2. derive the round key for $r_1 - 1$ or $r_2 + 1$ if he already has 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 (when compared to standard AES) since it also depends on the encryption key directly.

5.2.1 Key bytes for S boxes

In order to increase strength of the 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 the Proposition 3 from the BGE attack for all key bytes possibilities, it could help to derive the round keys.

From this reason we use completely different keys for the key-dependent S-boxes that in the 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 a "magicConstant". This makes two hash chains (one for round keys, one for S-boxes) completely different and non-transformable one to another.

5.3 Diffusion layer modification

In section 3.3.4 was mentioned the cipher invertibility. We 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 the state array depends only on 4 bytes = one column of the state array. Round function of AES acts independently on 4 columns, making it easy to invert it.

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

MDS matrix acts as a diffusion element in the cipher, since our cipher belongs to substitution-permutation cipher category. Our MDS matrix represents invertible linear mapping. The important metric for its security is *branch number* [51], it gives measure on the worst case diffusion. If the diffusion matrix has a maximal possible branch number, it is *optimal*. AES [3], Twofish [31] and SHARK [52] ciphers are using MDS matrices optimizing the branch number as a main security measure of diffusion layer.

For generating such MDS matrices is particularly interesting following proposition from SHARK cipher paper [52] (for the proof see the original paper).

Proposition 3. *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 [2]. Reed-Solomon codes are subset of the MDS codes, so their parity-check matrix can be used as MDS matrix in the cipher acting as a strong diffusion element. The MDS matrices derived from the 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 the MDS matrices is described in [53] using *Cauchy matrices*[3].

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

Also an article [54] mentions AES diffusion layer modification from 4×4 to 16×16 MDS matrix, argumenting with a stronger security within one round, what is particularly interesting in the whitebox context. They constructs the MDS matrix using Cauchy matrices. Cauchy matrices depend on the first row only, this increases a possible diversity of 16×16 MDS matrices helping the following idea - a key dependent diffusion element.

Consider also idea to have key-dependent MDS matrices. If we can generate the set S_{MDS} of MDS matrices representing an optimal linear mapping, their selection can be based on a key-dependent criteria. Set S_{MDS} can be also extended using following proposition from [55].

Proposition 4. *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 the BGE attack [5] and the Generic attack by Michiels [36] requires known MDS matrix coefficients (thus key-independent).

5.4 Analysis

In this chapter we try to analyze the suggested scheme improvements from the whitebox point of view, particularly we try to mount the BGE attack to this modified variant. Note the diffusion layer modification is not assumed here, but in the next section.

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

2. $(n,\ k,\ d)$-code is MDS iff $d = n - k + 1$
3. `http://www.proofwiki.org/wiki/Definition:Cauchy_Matrix`

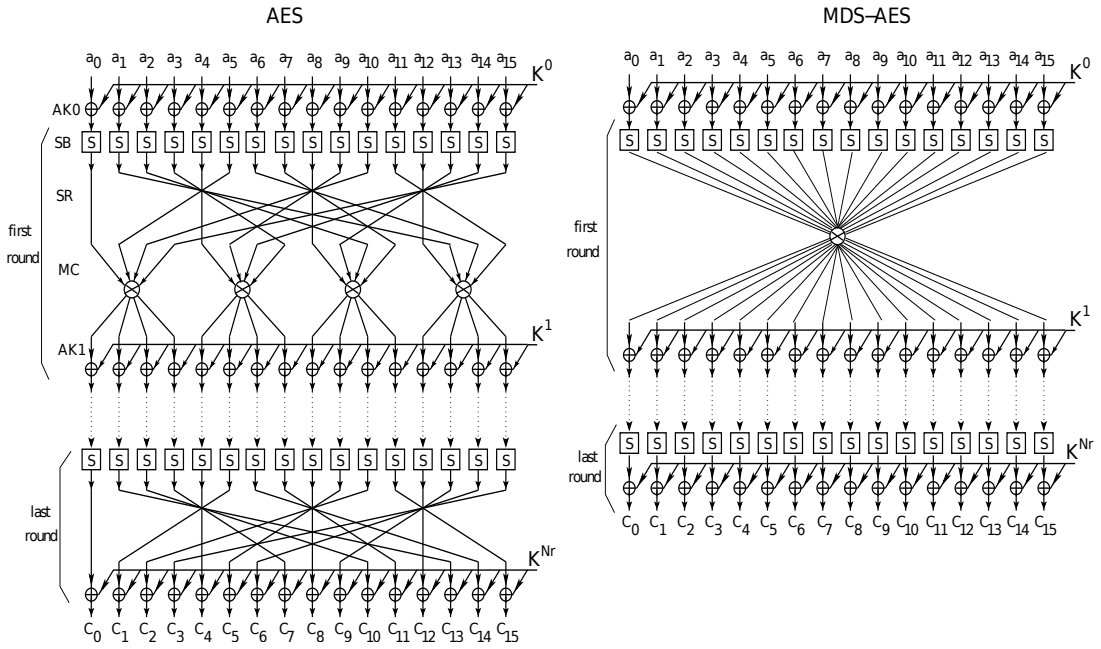


Figure 5.1: Computational diagrams of AES and MDS-AES (taken from [54])

mappings.

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

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

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

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

In our implementation of the BGE attack we iterate over $(\delta_i, c_i)_{i=0,\ldots,3} \in \mathrm{GF}\left(2^8\right) \times \mathrm{GF}\left(2^8\right)$ what gives complexity 2^{16} for one mapping. In each step is mapping checked for an affinity in 2^8 steps (for the affinity check algorithm see A.5), 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 the BGE attack to this modified variant is to guess also a particular S-box mapping for each $\widetilde{P}$ and to test it for affinity.

Equations 5.1 describe Twofish S-boxes. There are 2^{16} possible s_0 S-boxes. One S-box mapping 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 an attacker determines round keys for S-boxes it will be completely useless for further extraction of the cipher key since the hash chains are different.

Twofish S-boxes thus increase a complexity of the Proposition 3 from the BGE from 2^{24} to 2^{40} steps. This is still not strong enough, it is a highly parallelizable problem. In order to increase the work needed to mount the 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 [56]. We then would need S-box to depend on 13 bytes derived from the 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 the S-box space.

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

The S-box is generated recursively by a *sboxgen* (defined in 5.9), following the idea of nesting fixed permutations with addition of the round key:

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

Where $q'_{j,l}$ is one of two fixed Twofish 8-bit permutations. In order to select a good sequence of nested permutations, a deeper analysis woudl be needed, to avoid possible weaknesses induced by composing inappropriate permutations together. We can choose for example:

$$\begin{aligned} q'_0 &= \begin{bmatrix} q_1 & q_0 & q_1 & q_0 & q_1 & q_0 & q_1 & q_0 & q_1 & q_0 & q_1 & q_0 & q_0 \end{bmatrix} \\ q'_1 &= \begin{bmatrix} q_1 & q_1 & q_0 & q_0 & q_1 & q_1 & q_0 & q_0 & q_1 & q_1 & q_0 & q_0 & q_1 \end{bmatrix} \\ q'_2 &= \begin{bmatrix} q_0 & q_0 & q_1 & q_1 & q_0 & q_0 & q_1 & q_1 & q_0 & q_0 & q_1 & q_1 & q_0 \end{bmatrix} \\ q'_3 &= \begin{bmatrix} q_0 & q_0 & q_0 & q_1 & q_1 & q_1 & q_0 & q_0 & q_0 & q_1 & q_1 & q_1 & q_1 \end{bmatrix} \end{aligned} \tag{5.10}$$

We suggest to study linear/affine self-equivalences [46] as a part of an further analysis of the S-boxes generated by this algorithm.

Proposed S-Boxes give us a good security margin for the BGE attack, raising computational complexity to 2^{128} steps. 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 the hash easily.

The proposed cipher modifications affects only S-boxes and round keys, so other parts of the BGE attack are not affected, namely transforming non-linear parts to affine, the Propositions 1 and 2 work as before modifications.

The cipher modifications don't increase table sizes, because modifications are made 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 the encryption/decryption

performance since the only difference is made during particular whitebox instance (look-up tables) generation. An encryption/decryption algorithm itself is not affected.

5.5 Analysis of diffusion layer modification

Taking also the section 5.3 into account will result in bigger look-up tables. Mainly type II tables are affected, previously it was mapping $2^8 \rightarrow 2^{32}$, with cascade of XOR tables to sum four 32-bit values to obtain one column of state array. Now type II tables are $2^8 \rightarrow 2^{128}$, with cascade of XOR tables to a obtain whole state column vector. The cipher with modified structure uses type I tables instead of type II tables. Cascade of XOR tables is working in the same manner as in the external encodings in the first and the last round.

Table 5.2 summarizes changes in table sizes of WB AES. From this is visible that they come at a substantial price (215%) compared to the original whitebox AES implementation.

Type	original			modified		
	# of tables	bit-width	total size	# of tables	bit-width	total size
I	$4 \cdot 4 \cdot 2 = 32$	$8 \rightarrow 128$	131072 B	$4 \cdot 4 \cdot 2 = 32$	$8 \rightarrow 128$	131072 B
II	$4 \cdot 4 \cdot 9 = 144$	$8 \rightarrow 32$	147456 B	$4 \cdot 4 \cdot 9 = 144$	$8 \rightarrow 128$	589824 B
III	$4 \cdot 4 \cdot 9 = 144$	$8 \rightarrow 32$	147456 B	$4 \cdot 4 \cdot 9 = 144$	$8 \rightarrow 32$	147456 B
IV	$8 \cdot 6 \cdot 4 \cdot 9 = 1728$	$8 \rightarrow 4$	221184 B	$8 \cdot 3 \cdot 4 \cdot 9 = 864$	$8 \rightarrow 4$	110592 B
	$8 \cdot 4 \cdot 15 \cdot 2 = 960$	$8 \rightarrow 4$	122880 B	$8 \cdot 4 \cdot 15 \cdot 2 = 960$	$8 \rightarrow 4$	122880 B
IV				$8 \cdot 4 \cdot 15 \cdot 9 = 4320$	$8 \rightarrow 4$	552960 B
Total			770048 B 752 kB			1654784 B 1616 k B

Table 5.2: Whitebox implementation size with/without our modifications

From the security point of view this modifications also prevent the inverting attack mentioned in section 3.3.4. The function is now too wide to be inverted by running over GF $\left(2^8\right)^{16}$.

The key-dependent MDS matrix also prevents mounting the BGE attack. In particular we don't know MixColumn matrix coefficients so we are not able to construct the set β from the section 3.3 in [5]. This leads to further ambiguities so the Proposition 2 and 3 from the the BGE attack won't work anymore. This also makes complete recovering of I/O encodings unfeasible, using this attack. The transformation of random IO bijections to affine is not affected by these modification, but the parameters of the affine IO bijections are unkown.

As noted above, also the Generic attack by Michiels [36] requires known MDS matrix coefficients, that are not known in our proposal due to their key-dependency.

5.6 Drawbacks

By modification of an AES design we are coming up with a new cipher, what could also bring some possible problems. AES and Twofish have advantage of being well analyzed from the blackbox context and being relatively secure. Designing a new cipher may help to increase security level in whitebox context but there also may be weaknesses in the blackbox context. It would be needed to analyze the new cipher from this point of view, for example for a resistance to the linear or differential cryptanalysis.

We tried to design the scheme improvements following the standard established principles for a designing a secure block cipher, inspired by AES, Twofish, Shark and others. However, there is no guarantee that the proposed cipher is strong enough to resist classical blackbox context cryptanalysis.

6 Future work

As a future work we would like to study the blackbox properties of suggested improved cipher. In particular, it is worth to study the S-box construction and to test it for possible weaknesses. It will be needed to model S-boxes as algebraic equations and to perform various test, e.g. number of fixed points. One can also study these S-boxes and their differential / linear probability coefficients (measuring a resistance of the S-box to a differential / linear cryptanalysis). Testing S-boxes for Square and linear approximation attacks are also possible research directions.

The resistance to the BGE attack results from dependence on 13 round key bytes. If one shows that S-box space is smaller than assumed, it can be a vulnerability that can be exploited by a whitebox attack. Possible, a deeper study of the key-dependent S-boxes construction will be needed. One could also analyzethe key-dependent S-boxes of another ciphers, e.g. Blowfish [57].

We are proposing the systematic generation of the MDS matrices in the previous chapter. It would be needed to analyze the size of the MDS matrix space and the contribution of the key-dependence to the increasing resistance to attacks.

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

From the fact the cryptanalysis uses algebraic approach to break the AES based scheme, it would be needed to study and adapt ideas from schemes with a complex algebraic representation, in order to make the algebraic analysis more difficult. For example an IDEA [58] is a cipher with a complex algebraic representation. The design of IDEA is supported by a careful analysis of the interaction and algebraic incompatibilities of operations across the groups $(\mathbb{F}_2^n, \oplus)$, $(\mathbb{Z}_{2^n}, \boxplus)$ and $(\mathbb{Z}_{2^n+1}^*, \odot)$ [2].

Since the basic building block, look-up tables with XOR function, is vulnerable to recovering non-linear bijections, what enables to mount a potential algebraic attack, it is needed to come up with new building blocks, resistant to this kind of attacks. Possibly adding some noise could help in constructing new, probabilistic building blocks.

The important part of the further research should also be devoted to an analysis of known hard problems and their possible use in whitebox cryptography. Incorporating such a hard problem to a cipher should not limit its proper use, and what is more, it should be difficult for an attacker to break such scheme.

7 Conclusion

This thesis gives an introduction to a computing in an untrusted environment and summarizes current state of the art in this field, focusing mainly on the obfuscation, fully homomorphic encryption and whitebox cryptography.

Foundations and the basic building blocks used in whitebox implementations were explained. The thesis further goes through the construction of classical and new whitebox schemes using AES. The proof that scheme using dual ciphers is not more secure than the classical one was given. This result is new and was not published before. The thesis also describes implementation of those schemes and the attack. The implementation is publicly available so that researches interested in whitebox cryptography could use it, test their new improvements and attacks, since it aims at providing flexible framework for whitebox applications.

The thesis also suggests some new improvements that should increase a resistance to the algebraic attack, with following analysis of the improvements. But there is still a work to be done toward making whitebox implementation secure. The current state of the art suggests new primitives must be discovered, since no strong whitebox non-broken implementation is known. As the literature [5, 44] suggest, in order to design a strong scheme, new and innovative approaches has to be used. It is suggested to design a new cipher, with whitebox implementation pitfalls in mind. This thesis attempted to make such a small step toward this target by modifying the design of AES, introducing key-dependent building blocks.

A Appendix A

A.1 Attachments

The electornic archive of this thesis contains an archive *impl.tgz* that contains implemented algorithms. Note that in order to compile it NTL library is needed to be installed on the system.

The following list puts files with given file name prefix and implemented algorithm into a relation:

- `NTLUtils` contains some helper methods for NTL library
- `MixingBijections` contains algorithm for generating mixing bijections as described in the sections 3.3.3 and 4.3.3
- `LinearAffineEq` contains algorithm for finding linear and affine equivalences as described in the sections 4.1.5 and 4.3.4
- `GenericAES` implements generalized AES, with adjustable irreducible polynomial and generator, as described in the section 4.1.2
- `WBAES` implements both whitebox AES scheme described by Chow *et al.* [4] and Karroumi [1]
- `WBAESGenerator` contains algorithm for generating schemes mentioned in the previous point
- `BGEAttack` implements the BGE attack as described in the sections 3.4 and 4.4
- `main` contains benchmarking tests, compiles to binary executable code, use `--help` for more details

A.2 Hardware and software specifications

Several performance tests were performed in this thesis, here is a detailed description of used hardware and software.

A.3 Squaring matrix

Here is shown how to compute matrix Q from the section 4.1.5 that represents squaring operation in GF (2^8). Consider the following proposition:

Component	Specifications	Additional information
CPU	Intel® Core™ i5 M 560 @ 2.67GHz `http://ark.intel.com/products/49653`	64-bit, 4 thread cores 3 MB cache
RAM	2 x 4 GiB SODIMM DDR3 Synchronous 1067 MHz Kinston 9905428-005.A02LF	width: 64 bits
OS	Fedora 15	kernel 2.6.42.9-2.fc15.x86_64

Proposition 5. $\forall a, b \in GF\left(2^8\right) \ : \ (a+b)^2 = a^2 + b^2$.

Proof. From binomial theorem, assume general case, $a, b \in \mathrm{GF}(p^m)$, where p is prime.

$$(a+b)^p = a^p + \sum_{k=1}^{p-1} \binom{p}{k} a^{p-k} b^k + b^p = a^p + b^p$$

since $\binom{p}{k}$ is multiple of p for $0 < k < p$ (from binomial coefficient definition) and multiples of p are 0 in $\mathrm{GF}(p^m)$. ($\binom{p}{k} \notin \mathrm{GF}(p^m)$ is coefficient, so it is reduced mod p). To finish proof let $p = 2$. □

Proposition 6. $\forall a, b \in GF\left(2^8\right) \ : \ (a+b)^2 = a^2 + b^2 \Rightarrow$ *squaring in* $GF\left(2^8\right)$ *is linear operation which is equivalent to matrix multiplication with coefficients from* $GF\left(2\right)$.

Proof. Let's have $a \in \mathrm{GF}\left(2^8\right)$. Elements from this field are represented as polynomials, in polynomial basis $\begin{bmatrix} x^0 & x^1 & x^2 & x^3 & x^4 & x^5 & x^6 & x^7 \end{bmatrix}$. Thus we can write: $a = \sum_{i=0}^{7} a_i \cdot x^i$ where $a_i \in \mathrm{GF}\left(2\right)$. Also note that $a_i = a_i^2$, because $\mathrm{GF}\left(2\right) = (\{0,1\}, +, \cdot)$, so $0 = 0^2 \wedge 1 = 1^2$.

Thus we can write

$$a^2 = \left(\sum_{i=0}^{7} a_i \cdot x^i\right)^2 = \sum_{i=0}^{7} a_i^2 \cdot x^{i^2} = \sum_{i=0}^{7} a_i \cdot x^{i^2} = \sum_{i=0}^{7} x^{i^2} \cdot a_i = \left(\sum_{i=0}^{7} x^{i^2}\right) \cdot a = Q \cdot a$$

□

Thus the squaring matrix is Q from the proof. It holds that $\mathrm{GF}\left(2^8\right) \simeq \mathrm{GF}\left(2\right)^8$ since finite fields with the same number of elements are isomorphic. We use this isomorphism to obtain matrix Q. It is enough to transform polynomial base in $\mathrm{GF}\left(2^8\right)$ to vector base in $\mathrm{GF}\left(2\right)^8$. It is easy to see that:

$$\begin{bmatrix} x^0 & x^1 & x^2 & x^3 & x^4 & x^5 & x^6 & x^7 \end{bmatrix}_{\mathrm{GF}(2^8)} \mapsto \begin{bmatrix} e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & e_7 & e_8 \end{bmatrix}_{\mathrm{GF}(2)^8} = I_{8,\,\mathrm{GF}(2)^8}$$

Where e_i is i-th base vector from standard base.

Now it is obvious how to construct matrix Q:

$$Q = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}_{\mathrm{GF}(2)^8}$$

A.4 Multiplication matrix

In this section is described how to construct a 8×8 matrix $[a]$ with coefficients from $\mathrm{GF}(2)$ that represents multiplication by constant $a \in \mathrm{GF}\left(2^8\right)$ in $\mathrm{GF}\left(2^8\right)$. It uses the same technique as in section A.3, using isomorphism.

Recall $u_{\mathrm{GF}(2^8)} = \sum_{i=0}^{7} u_i \cdot x^i \mapsto \begin{bmatrix} u_0 & \cdots & u_7 \end{bmatrix}^T_{\mathrm{GF}(2)^8} = \sum_{i=1}^{8} u_{i-1} \cdot e_i$ where $u_i \in \mathrm{GF}(2)$. Multiplication is linear transformation, so let's denote multiplication by a as $L_a : \mathrm{GF}(2)^8 \to \mathrm{GF}(2)^8$. Now from linearity:

$$\begin{aligned} L_a(u) &= L\left(\sum_{i=1}^{8} u_{i-1} \cdot e_i\right) = \sum_{i=1}^{8} u_{i-1} \cdot L_a(e_i) \\ &= \sum_{i=1}^{8} L_a(e_i) \cdot u_{i-1} = \begin{bmatrix} L_a(e_1) & \ldots & L_a(e_8) \end{bmatrix} \cdot \begin{bmatrix} u_0 \\ \vdots \\ u_7 \end{bmatrix} \end{aligned}$$

Thus matrix $[a]$ has form:

$$[a] = \begin{bmatrix} L_a(e_1) & \ldots & L_a(e_8) \end{bmatrix}_{\mathrm{GF}(2)^8}$$

A.5 Affinity check

In this section we describe affinity check needed in proposition 3 of the 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:

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

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

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

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

And from linearity the following formula must hold:

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

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 $g_j = \widetilde{P'}(e_j)$. Now it is visible that time complexity is 2^8.

Algorithm 2 Algorithm for testing given mapping for being affine

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

Bibliography

[1] 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`.

[2] Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot. *Handbook of Applied Cryptography*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 1996. ISBN 0849385237.

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

[4] Stanley Chow, Phil Eisen, Harold Johnson, and Paul 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.

[5] 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`.

[6] *Towards mobile cryptography*, 1998. URL `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=674837`.

[7] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '01, pages 1–18, London, UK, UK, 2001. Springer-Verlag. ISBN 3-540-42456-3. URL `http://dl.acm.org/citation.cfm?id=646766.704152`.

[8] Benjamin Lynn, Manoj Prabhakaran, and Amit Sahai. Positive results and techniques for obfuscation. In *In EUROCRYPT '04*, 2004.

[9] Susan Hohenberger, Guy N. Rothblum, Abhi Shelat, and Vinod Vaikuntanathan. Securely obfuscating re-encryption. In *Proceedings of the 4th conference on Theory of cryptography*, TCC'07, pages 233–252, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 978-3-540-70935-0. URL `http://dl.acm.org/citation.cfm?id=1760749.1760767`.

[10] Shafi Goldwasser and Yael Tauman Kalai. On the impossibility of obfuscation with auxiliary input. In *Proceedings of the 46th Annual IEEE Symposium on Foundations*

of Computer Science, FOCS '05, pages 553–562, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2468-0. doi: 10.1109/SFCS.2005.60. URL http://dx.doi.org/10.1109/SFCS.2005.60.

[11] Nir Bitansky and Omer Paneth. On the impossibility of approximate obfuscation and applications to resettable cryptography. *IACR Cryptology ePrint Archive*, 2012:729, 2012. URL http://dblp.uni-trier.de/db/journals/iacr/iacr2012.html#BitanskyP12.

[12] Jan Cappaert. *Code Obfuscation Techniques for Software Protection*. Phd. thesis, Katholieke Universiteit Leuven, 2012. URL http://www.cosic.esat.kuleuven.be/publications/thesis-199.pdf.

[13] Andrew C. Yao. Protocols for secure computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, SFCS '82, pages 160–164, Washington, DC, USA, 1982. IEEE Computer Society. doi: 10.1109/SFCS.1982.88. URL http://dx.doi.org/10.1109/SFCS.1982.88.

[14] Ronald Cramer, Ivan Damgard, and Jesper Buus Nielsen. Secure multiparty computation and secret sharing - an information theoretic appoach. [online] http://www.daimi.au.dk/~ivan/MPCbook.pdf, cit. 26.5.2013, 2012.

[15] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer — efficiently. In *Proceedings of the 28th Annual conference on Cryptology: Advances in Cryptology*, CRYPTO 2008, pages 572–591, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 978-3-540-85173-8. doi: 10.1007/978-3-540-85174-5_32. URL http://dx.doi.org/10.1007/978-3-540-85174-5_32.

[16] Ronald L. Rivest, Len Adleman, and Michael L. Dertouzos. On data banks and privacy homomorphisms. *Foundations of Secure Computation, Academia Press*, pages 169–179, 1978.

[17] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-dnf formulas on ciphertexts. In *Proceedings of the Second international conference on Theory of Cryptography*, TCC'05, pages 325–341, Berlin, Heidelberg, 2005. Springer-Verlag. ISBN 3-540-24573-1, 978-3-540-24573-5. doi: 10.1007/978-3-540-30576-7_18. URL http://dx.doi.org/10.1007/978-3-540-30576-7_18.

[18] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st annual ACM symposium on Theory of computing*, STOC '09, pages 169–178, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-506-2. doi: 10.1145/1536414.1536440. URL http://doi.acm.org/10.1145/1536414.1536440.

[19] Craig Gentry and Shai Halevi. Implementing gentry's fully-homomorphic encryption scheme. In *Proceedings of the 30th Annual international conference on Theory and applications of cryptographic techniques: advances in cryptology*, EUROCRYPT'11, pages 129–148, Berlin, Heidelberg, 2011. Springer-Verlag. ISBN 978-3-642-20464-7. URL http://dl.acm.org/citation.cfm?id=2008684.2008697.

[20] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *Proceedings of the 29th Annual international conference on Theory and Applications of Cryptographic Techniques*, EUROCRYPT'10, pages 24–43, Berlin, Heidelberg, 2010. Springer-Verlag. ISBN 3-642-13189-1, 978-3-642-13189-9. doi: 10.1007/978-3-642-13190-5_2. URL http://dx.doi.org/10.1007/978-3-642-13190-5_2.

[21] Jean-Sébastien Coron, David Naccache, and Mehdi Tibouchi. Public key compression and modulus switching for fully homomorphic encryption over the integers. In *Proceedings of the 31st Annual international conference on Theory and Applications of Cryptographic Techniques*, EUROCRYPT'12, pages 446–464, Berlin, Heidelberg, 2012. Springer-Verlag. ISBN 978-3-642-29010-7. doi: 10.1007/978-3-642-29011-4_27. URL http://dx.doi.org/10.1007/978-3-642-29011-4_27.

[22] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. Batch fully homomorphic encryption over the integers. *IACR Cryptology ePrint Archive*, 2013:36, 2013.

[23] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-lwe and security for key dependent messages. In *Proceedings of the 31st annual conference on Advances in cryptology*, CRYPTO'11, pages 505–524, Berlin, Heidelberg, 2011. Springer-Verlag. ISBN 978-3-642-22791-2. URL http://dl.acm.org/citation.cfm?id=2033036.2033075.

[24] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, pages 309–325, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1115-1. doi: 10.1145/2090236.2090262. URL http://doi.acm.org/10.1145/2090236.2090262.

[25] Craig Gentry, Shai Halevi, Chris Peikert, and Nigel P. Smart. Ring switching in bgv-style homomorphic encryption. In *Proceedings of the 8th international conference on Security and Cryptography for Networks*, SCN'12, pages 19–37, Berlin, Heidelberg, 2012. Springer-Verlag. ISBN 978-3-642-32927-2. doi: 10.1007/978-3-642-32928-9_2. URL http://dx.doi.org/10.1007/978-3-642-32928-9_2.

[26] Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic evaluation of the aes circuit. *IACR Cryptology ePrint Archive*, 2012:99, 2012. URL `http://dblp.uni-trier.de/db/journals/iacr/iacr2012.html#GentryHS12`. informal publication.

[27] Craig Gentry. Fully Homomorphic Encryption: current state of the art. Africacrypt 2012, [online], 2012. URL `http://www.aui.ma/africacrypt2012/images/africacrypt2012/fully%20homomorphic%20encryption.pdf`.

[28] Michael Naehrig, Kristin Lauter, and Vinod Vaikuntanathan. Can homomorphic encryption be practical? In *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*, CCSW '11, pages 113–124, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-1004-8. doi: 10.1145/2046660.2046682. URL `http://doi.acm.org/10.1145/2046660.2046682`.

[29] Dan Boneh, Craig Gentry, Shai Halevi, and Frang Wang. Private database queries using somewhat homomorphic encryption. Cryptology ePrint Archive, Report 2011/449, 2012.

[30] 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.

[31] 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.

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

[33] Stanley Chow, Phil Eisen, Harold Johnson, and Paul C. Van Oorschot. A white-box DES implementation for DRM applications. In *In Proceedings of ACM CCS-9 Workshop DRM*, pages 1–15. Springer, 2002.

[34] Matthias Jacob, Dan Boneh, and Edward W. Felten. Attacking an obfuscated cipher by injecting faults. In Joan Feigenbaum, editor, *Digital Rights Management Workshop*, volume 2696 of *Lecture Notes in Computer Science*, pages 16–31. Springer, 2002. ISBN 3-540-40410-4. URL `http://dblp.uni-trier.de/db/conf/ccs/ccsdrm2002.html#JacobBF02`.

[35] Hamilton E. Link and William D. Neumann. Clarifying obfuscation: Improving the security of white-box des. In *Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'05) - Volume I - Volume 01*, ITCC '05, pages 679–684, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2315-3. doi: 10.1109/ITCC.2005.100. URL `http://dx.doi.org/10.1109/ITCC.2005.100`.

[36] Wil Michiels and Paul 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.

[37] Yaying Xiao and Xuejia Lai. A secure implementation of white-box AES. In *Computer Science and its Applications, 2009. CSA '09. 2nd International Conference on*, pages 1–6, 2009. doi: 10.1109/CSA.2009.5404239. URL http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=05404239.

[38] Yoni De Mulder, Peter Roelse, and Bart Preneel. Cryptanalysis of the Xiao - Lai white-box AES implementation. In Lars R. Knudsen and Huapeng Wu, editors, *Selected Areas in Cryptography*, volume 7707 of *Lecture Notes in Computer Science*, pages 34–49. Springer, 2012. ISBN 978-3-642-35999-6. URL http://dblp.uni-trier.de/db/conf/sacrypt/sacrypt2012.html#MulderRP12.

[39] Julien Bringer, Hervé Chabanne, and Emmanuelle Dottax. White box cryptography: Another attempt. *IACR Cryptology ePrint Archive*, 2006:468, 2006. URL http://dblp.uni-trier.de/db/journals/iacr/iacr2006.html#BringerCD06a.

[40] Yoni De Mulder, Brecht Wyseur, and Bart Preneel. Cryptanalysis of a perturbated white-box AES implementation. In Guang Gong and Kishan Chand Gupta, editors, *INDOCRYPT*, volume 6498 of *Lecture Notes in Computer Science*, pages 292–310. Springer, 2010. ISBN 978-3-642-17400-1. URL http://dblp.uni-trier.de/db/conf/indocrypt/indocrypt2010.html#MulderWP10.

[41] Claude Shannon. Communication theory of secrecy systems. *Bell System Technical Journal, Vol 28, pp. 656–715*, 1949.

[42] James A. Muir. A tutorial on white-box AES, 2011. URL http://eprint.iacr.org/2013/104.pdf.

[43] James Xiao and Yongxin Zhou. Generating large non-singular matrices over an arbitrary field with blocks of full rank. *IACR Cryptology ePrint Archive*, 2002:96, 2002. URL http://dblp.uni-trier.de/db/journals/iacr/iacr2002.html#XiaoZ02.

[44] Brecht Wyseur. *White-Box Cryptography*. Phd. thesis, Katholieke Universiteit Leuven, 2009. URL http://docserver.bis.uni-oldenburg.de/publikationen/dissertation/2001/aneacr01/aneacr01.html.

[45] Specification for the Advanced Encryption Standard (AES). Federal Information Processing Standards Publication 197, 2001. URL http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

[46] Alex Biryukov, Christophe De Cannièere, An Braeken, and Bart Preneel. A toolbox for cryptanalysis: linear and affine equivalence algorithms. In *Proceedings of the 22nd international conference on Theory and applications of cryptographic techniques*, EUROCRYPT'03, pages 33–50, Berlin, Heidelberg, 2003. Springer-Verlag. ISBN 3-540-14039-5. URL http://dl.acm.org/citation.cfm?id=1766171.1766175.

[47] 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.

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

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

[50] Jeremi M. Gosney. Password cracking hpc. Passwords ^12 Security Conference, [online], December 2012. URL http://passwords12.at.ifi.uio.no/Jeremi_Gosney_Password_Cracking_HPC_Passwords12.pdf.

[51] 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.

[52] 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.

[53] 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.

[54] 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.

[55] 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.

[56] Andrey Bogdanov, Dmitry Khovratovich, and Christian Rechberger. Biclique cryptanalysis of the full AES. Cryptology ePrint Archive, Report 2011/449, 2011. http://eprint.iacr.org/.

[57] Bruce Schneier. Description of a new variable-length key, 64-bit block cipher (blowfish). In *Fast Software Encryption, Cambridge Security Workshop*, pages 191–204, London, UK, UK, 1994. Springer-Verlag. ISBN 3-540-58108-1. URL `http://dl.acm.org/citation.cfm?id=647930.740558`.

[58] Xuejia Lai and James L. Massey. A proposal for a new block encryption standard. In *Proceedings of the workshop on the theory and application of cryptographic techniques on Advances in cryptology*, EUROCRYPT '90, pages 389–404, New York, NY, USA, 1991. Springer-Verlag New York, Inc. ISBN 0-387-53587-X. URL `http://dl.acm.org/citation.cfm?id=112331.112375`.