DEGREE PROJECT IN COMPUTER SCIENCE AND ENGINEERING, SECOND CYCLE, 30 CREDITS

STOCKHOLM, SWEDEN 2019

Randomly perturbing the bytecode of white box cryptography implementations in an attempt to mitigate side-channel attacks

ALEXANDER KOSKI

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Randomly perturbing the bytecode of white box cryptography implementations in an attempt to mitigate side-channel attacks

ALEXANDER KOSKI

Master in Computer Science
Date: March 29, 2020
Supervisor: Martin Monperrus
Examiner: Roberto Guanciale
School of Electrical Engineering and Computer Science

Abstract

This study takes one step further towards constructing a tool able to automatically amplify the security on your cryptographic implementations. In white box cryptography the encryption key is hidden inside the encryption algorithm out of plain sight. An attacker can try to extract the secret key by conducting a side channel attack, differential computational analysis, which many white boxes are vulnerable to. The technique to increase security explored in this study consists of randomly with different probabilities perturb the white box by adding the value one to a variable inside the running white box. This does break the correctness of the output on all the three tested white box implementations to various extents, but some perturbations can be made which maintains fairly high correctness on the output of the program. Running a white box with perturbations does not cause any significant increase in execution time. Out of more than 100 000 possible perturbation points 25 were chosen to be investigated further. In one case the security of a perturbed white box increased, but in four similar cases the white box was made more insecure, otherwise no change in security was observed. A more sophisticated technique of identifying the best point to insert perturbations are therefore required in order to further investigate how to increase the security of your cryptographic implementations while still maintaining a fairly high correctness despite the program experiencing random perturbations.

Sammanfattning

Denna studie tar ett steg längre i att skapa ett verktyg som automatiskt höjer säkerheten på kryptografiska implementationer. I white box-kryptografi är krypteringsnyckeln som används redan gömd inuti den valda krypteringsalgoritmen. En användare med onda intentioner kan vilja försöka att extrahera denna gömda krypteringsnyckel genom att utföra en typ utav side-channel -attack, differential computational analys, vilket flera white box-implementationer är sårbara för. Tekniken för att öka säkerheten hos dessa white box-implementationer som utforskas i denna studie går ut på att slumpmässigt, med olika sannolikheter störa white box-implementationen genom att addera värdet ett (1) till en godtycklig variabel mitt under programmets körning. Detta sänker korrektheten på outputen från alla de tre testade white box-implementationerna olika mycket, men vissa störningar kan utföras och fortfarande resultera i hög korrekthet. Att sätta in sådana störningar i white box-implementationerna har ingen större inverkan på dess totala exekveringstid. Utav 100 000 möjliga ställen att sätta in en störning på valdes 25 utav dessa att undersökas vidare. I ett utav dessa fall ökade säkerheten på white box-implementationen, men i fyra liknande fall minskade säkerheten, annars på övriga störningspunkter fanns det inga tecken på en förändring i säkerhet. Ett mer sofistikerat sätt att identifiera de bästa störningspunkterna på behöver därför utredas för att bygga vidare på detta arbete med att öka säkerheten hos kryptografiska implementationer genom att utsätta dem för slumpmässiga störningar och samtidigt kunna hålla hög korrekthet i programmets resultat.

Contents

1 Introduction **1**

2 Background **3**
2.1 White Box Cryptography 3
2.2 Side Channel Attacks 4
2.2.1 Differential Power Analysis (DPA) 4
2.2.2 Differential Computation Analysis (DCA) 6
2.3 Attack Tools - Deadpool and Daredevil 6
2.4 Targets 7
2.4.1 Ches2016 7
2.4.2 Kryptologik 7
2.4.3 NSC2013 8
2.5 Correctness Attraction 8

3 State of the Art **10**
3.1 Perturbation as a Defense Technique for White Boxes 10
3.2 Perturbing Arithmetic Calculations 10
3.3 Correctnessbreaking Countermeasures 11

4 Research Questions **12**
4.1 Correctness Attraction of Perturbed Cryptographic White Box Implementations 12
4.2 Execution Time of Perturbed White Box Implementations . . . 12
4.3 Perturbed Cryptographic Implementations Resistance Against Side Channel Attacks 13

5 Experimental Methodology **14**
5.1 Threat Model 14
5.2 Scope of Methodology 15
5.3 Inserting Perturbations Into White Box Implementations . . . 15

5.3.1 Example of Original Code in LLVM IR 16
5.3.2 Example of a Perturbation on the First Argument in a Binary Instruction 17
5.3.3 Example of a Perturbation on the Result Value in a Binary Instruction 18
5.4 Correctness Attraction of Perturbed Cryptographic White Box Implementations 18
5.5 Execution Time of Perturbed White Box Implementations . . . 19
5.6 Perturbed Cryptographic Implementations Resistance Against Side Channel Attacks by Introducing 'Attack Scores' 21

6 Experimental Results **22**
6.1 Correctness Attraction of the Perturbed Cryptographic White Box Implementations 22
6.1.1 Key Takeaways 26
6.2 Execution Time of Perturbed Cryptographic Implementations . 26
6.2.1 Key Takeaways 29
6.3 Perturbed Cryptographic Implementations Resistance Against Side Channel Attacks 30
6.3.1 Reference Attacks 30
6.3.2 Attacks on Ches2016 31
6.3.3 Kryptologik 34
6.3.4 NSC2013 36
6.4 Key Takeaways 40

7 Discussion **41**
7.1 Comments on Correctness Attraction of Perturbed Cryptographic White Box Implementations 41
7.2 Comments on Execution Time of Perturbed White Box Implementations 42
7.3 Perturbed Cryptographic Implementations Resistance Against Side Channel Attacks 43
7.4 Security and Correctness 44
7.5 Limitations 45

8 Conclusions **47**
8.1 Contributions 47
8.2 Future Research 48

Bibliography **50**

Chapter 1

Introduction

It is hard to play a game of hide and seek when your opponent has full control over the game field. The same can be said about computer security within the field of white box cryptography. Cryptographic implementations of different encryption and decryption algorithms were often considered as a black box by any third parties' point of view. Input as plain text gets transformed inside the black box and outputted as a ciphertext [1]. Nowadays more complexity is added to the black box model due to the fact that an attacker can be in possession and have full control over the device running the black box cryptographic implementation. All the black box inner workings get exposed, the black box suddenly gets transparent [2]. This gave rise to a new area in cryptography, white box cryptography. White box cryptography is cryptographic implementations executed on a machine where you are in full control and the algorithms secret encryption key is embedded inside the encryption algorithm. The user only needs to supply plaintext in order to get back the corresponding ciphertext. The white box cryptography designers have to take extra steps in order to keep the embedded encryption key safe inside the white box due to the fact that an attacker has full control over the machine the white box implementation is running on.

Despite it being one of the fundamental functionalities of a white box, keeping the secret encryption key secure inside the implementation is hard to achieve. All publicly known white boxes in academia, to this day, has been broken and its secret key successfully extracted [3, 1]. Hiding a secret key inside program code can be seen as a kind of code obfuscation, which is proven to be a solvable task to unroll [4].

A common method of retrieving the secret key from a white box implementation is by analysing the different side channels that appears when the white

box is executed. Such side channels may be power consumption, electromagnetic waves, timing, and since we have full control over the machine running the white box, we can also look at memory read- and writes [2, 5, 6]. Even though researchers have not been able to find any mathematical loopholes in one of the most commonly used encryption algorithm, AES, it is however with today's binary analysis tools possible to extract a secret key from many different kinds of AES white box implementations that once was called state of the art [7, 3].

Because of the security demands on white box implementations and the proven vulnerabilities, many attempts to develop techniques to harden white boxes has been made [8, 9, 10, 11, 12, 13] but without luck [14, 13, 15, 16, 17, 18]. Researchers have focused their defensive techniques towards making the white box implementation to behave somewhat random with respect to its earlier executions. The randomness is supposed to mask the real and otherwise predictable behavior of the white box implementation in order to protect the encryption key [13, 9]. This thesis also aims at mitigating side channel attacks with the foundation of randomization in mind. The attempted mitigating technique is implemented by randomly perturbing the values stored inside the variables inside the processes of the running white box. The properties of the perturbed white box is observed to see if the random perturbations are enough to harden the implementation against a common side channel attack and still leave it usable for its intended use.

Earlier attempts at mitigating side channel attacks on white boxes has developed techniques and tools where a large focus is on maintaining the correctness on the output of the original white box. The idea of having a correctness breaking mitigation technique is novel to the white box cryptography community and is within said field contributions to science will be made to from this study.

Chapter 2

Background

2.1 White Box Cryptography

White box cryptographic implementations are a special kind of cryptographic implementation where the encryption key is embedded and hidden inside the encryption algorithm [19]. White box cryptography enables you to deliver a white box to a client where the embedded secret encryption key is chosen for that specific client in mind. The client is unknowing of what secret encryption key has been chosen and baked into the received white box implementation. The client can now encrypt messages using the white box implementation, white boxes are commonly used inside the digital rights management sector [20].

While white box cryptography sounds like a nice idea with some real use cases there exists a catch, not a single white box has been proven to be secure. All public white boxes in academia has been broken, with a demonstration on how the secret key could be extracted in plain text [3, 1]. Hiding the secret key inside the code of the white box implementation can be seen as a sort of code obfuscation and it is known that code obfuscation does not make your code immune towards reverse engineering. No code obfuscation is impossible to reverse. There only exists code obfuscation techniques to make it somewhat inconvenient for the attacker to reverse the obfuscated code. Therefore the aim with obfuscation is to make the obfuscation challenging enough so the attacker gives up on trying to reverse the obfuscated code [21].

Sophisticated efforts are made to hide the secret encryption key inside look up tables (LUT) with pre calculated values [19, 20]. Inside the standard AES and DES encryption algorithms the secret encryption key is being combined with the user input and a predefined initialization vector (IV) during the en-

cryption phases. The combination of the three yields the ciphertext [22, 23]. To create a lookup table which can be used in a white box, the secret encryption key is broken up into smaller parts and equally small parts of all possible user input is generated. These parts are combined with the IV and the result is stored inside the binary implementations LUT. The white box implementation uses the IV and the real supplied user input to access the pre calculated values inside the LUT instead of performing the actual calculations every time the white box gets executed.

The calculations for creating the values stored inside the LUT are irreversible therefore the LUT inside a white box are considered secure against an attacker trying to reverse the calculations to retrieve the secret key [3].

2.2 Side Channel Attacks

A side channel occurring from a running program is the physical side effects that can be measured during and after runtime [24]. Such side channels are execution time, electromagnetic emissions and power consumption [5, 6]. Educated guesses are made towards understanding the executed implementation based on looking at these side channels.

Side channels occur at the very bottom of the system abstraction layers [25] where the hardware and physical circuits of the computer system is located [5]. There do exist techniques to mitigate side channel attacks implemented in software. Making sure your algorithm runs in constant time or in random time [17] is a way to protect against timing attacks. The general rule of protecting an implementation from a side channel is to make it run as uniformly as possible under different inputs and in between executions or to make is run as randomly as possible compared to earlier runs regardless of supplied input. In this way it makes it hard to draw conclusions and extract information from the execution of the implementation with respect to its behavior under different inputs. To develop an exploit based on a side channel attack, reverse engineering to some extent is needed in order to understand how the program works.

2.2.1 Differential Power Analysis (DPA)

Power consumption from the algorithm having various machine instructions executed on the processor is a side channel from which you can learn a lot [2]. One of the oldest side-channel attacks is called simple power analysis, SPA. Simple power analysis can be applied on functional black box implementations of encryption algorithms making the different bits in the encryption key visible

by using an oscilloscope. It is just a matter of probing and zooming in at the right place in the power recording [15, 5, 2] SPA would evolve, get more sophisticated and transform into DPA as cryptographic implementations got more advanced and transformed from black box to white box implementations [1].

What makes true DPA hard is the fact that you need special equipment to measure power consumption on individual power lines coming of individual chips inside the computer. The obtained measurements are also subject to noise which you need to account for in your calculations. A recording of a device power consumption over time is called a power trace. Multiple power traces is needed to perform a successful side channel in order to find the secret encryption key with high probability [3]. A way of finding the secret encryption key is by finding a hypothetical key which has the highest correlation coefficient between it and the targeted white box recorded power traces. A popular correlation coefficient to use is the Pearson's correlation coefficient [26]. This side channel attack is therefore seen as somewhat probabilistic. In the case where the white box cryptography implementation is vulnerable to a DPA attack the accuracy of guessing the right key increases by an increasing number of recorded power traces.

The secret key bits retrieved from the DPA side channel attack is from the actual key used in the calculations together with the plaintext, in AES this means that the round key is the secret key which gets compromised in a successful DPA side channel attack. To retrieve the full secret encryption key an additional step to reverse the AES key schedule needs to be done which is trivial since the AES key schedule is reversible according to the specification of the AES algorithm [22]. Therefore, a white box is considered broken when one of the round keys are retrieved in plaintext to the attacker.

To record and perform calculations on many power traces is a somewhat slow task to the processor. Research has been done making it able to perform the calculations on the GPU which speeds up the attack. To add additional power traces recorded at a later point in time to earlier calculations is also made possible in order to afterwards increase the accuracy of the attack if the result from the previous attack displayed a low probability key guess. The number of traces needed to extract the secret key can be a measure of how secure the white box implementations is [16].

2.2.2 Differential Computation Analysis (DCA)

With the different binary analysis tools available today, it is possible to obtain a trace, equivalent to the earlier mentioned power trace, without any noise. It is no longer necessary to probe individual bus lines for power consumption in order to guess what bit was written to memory at certain times, we can do that with tools like PIN [27] and Valgrind [28] purely in software. This means it is possible to get memory trace files containing the memory read and write address, timestamp and the data being read or written without any external hardware tools. The principle to derive the secret key is the same as in the DPA attack. Therefore DCA is still considered as a side channel attack even though it actually reads the process memory. Researchers have developed a tool based on the principles of the DCA attack and with it manage to break many of the publicly available white box challenges found online. The attack technique works very well on white boxes with implementations relying heavily on memory table lookups [1].

2.3 Attack Tools - Deadpool and Daredevil

The authors of [1] developed tools to be used for conduction a DCA attack. The attack is divided into two parts using their two tools. The first part of the attack uses the tool named Deadpool with the sole purpose of collecting memory trace files. These trace files can later be sent to the second part of the attack, to the tool Daredevil. It is in this second part of the attack the calculations of the Pearson correlation coefficient takes place. The first part of the attack is non deterministic since it records memory traces using random input to the white box, the second part of the attack, calculating the correlation coefficient from a given trace is deterministic. Two equal memory traces produce two equal key guesses from the Daredevil software. Inside the repository there is a collection of 19 publicly available white boxes which have all been cracked with the previously mentioned tools and techniques [29, 30]. The correct key for each challenge is posted along with the challenge inside the Deadpool repository.

2.4 Targets

2.4.1 Ches2016

Language	C
Lines of code	8164
Input format	16 program arguments supplied by std-in, each argument consisting of 1 byte represented in hex
Output format	16 bytes represented in hex separated by a space

Ches2016 [29], an online challenge. This AES white box challenge origins from the Conference on Cryptographic Hardware and Embedded Systems, the year 2016 [31, 10]. The white box has been cracked where the tools Deadpool and Daredevil has been demonstrated to work [1]. There are other techniques to solve this challenge since the leaderboard shows solved submissions with a timestamp that predates the creation date for the Daredevil and Deadpool attack tools [10, 29, 30]. Reverse engineering write-up on the implementation by Ahn et. al. [7] states that the white box consists of 4,048 look-up tables and 41 local variables (8-bit data) in order to take care of intermediate results. The white box conducts 4,080 load and store operations from the tables. It takes around 200 traces to crack the white box.

2.4.2 Kryptologik

Language	C
Lines of code	4197
Input format	16 program arguments supplied by std-in, each argument consisting of 1 byte represented in hex
Output format	16 bytes represented in hex separated by a space

Kryptologik [29], an online challenge [11], is a kind of AES white box although it has been altered to use 15 independent round keys. There exists no single secret encryption key that can be derived from the different round keys with the normal AES key schedule algorithm. This white box uses lookup

tables as its main source of obfuscation [32]. Otherwise it has similarities with the regular AES encryption algorithm.

2.4.3 NSC2013

Language	C
Lines of code	277
Input format	One program argument supplied by std-in consisting of 16 bytes represented in hex
Output format	16 bytes represented in hex separated by a space

NSC2013 [29], an online challenge originated from the NoSuchCon 2013 conference [12], based on the AES encryption algorithm. In this challenge the lookup table is generated separately and done before for the core white box algorithm is run [33].

2.5 Correctness Attraction

Perturbations in programs where a bit is flipped is seldom naturally occurring in regular day to day programming, although Nasa has experienced errors due to bits flipping on their machines deployed in space [34, 35]. Perturbations under more controlled circumstances have been studied by Danglot et. al [36]. How well a program performs under perturbation is measured in a measurement called correctness attraction. The program is said to have high correctness attraction if the program manages to produce correct output from given input while a perturbation on a value inside the running program is applyed. If a perturbation to a value used by the running program leads to the program producing faulty output, the program is said to display poor correctness attraction. A sudden change can for example be an addition or subtraction by one to the value inside a variable. The perturbation to the variable is not part of the programs initially defined algorithm. Most of the programs, 68%, can still manage to deliver correct output under perturbation. Different types of algorithms deliver different correctness attraction under perturbation. It is known that programs related to the area of security show less correctness attraction than the overall average correctness attraction over all the tested programs.

Program	Correctness
RSA	54%
Md5	29%
Rc4	38%

A perturbation can be done in many ways, adding or subtracting the value one to the value inside a variable, bit flipping a boolean, adding random delays to the program, setting a variables value to zero and more. The actual place in the code where there exists a value inside a variable subject for a possible perturbation is called a perturbation point. The correctness is dependent on the perturbation point, some perturbation points produce higher correctness attraction than others. What happens when a variable is perturbed is a change in the value inside the variable used by the actual machine at runtime. The implementation under perturbation might take a different execution path compared to what it would otherwise do without perturbation. It might after perturbation enter an if statement it otherwise would not have entered, or take an extra round inside a loop or it might not affect the execution of a program at all. It all depends on where the perturbation happens. Perturbation points with high correctness attraction, over 75%, are called robust perturbation points. Perturbation points which produce 100% correctness under perturbation are called antifragile points, the rest of the perturbation points are classified as fragile points [36].

Chapter 3

State of the Art

3.1 Perturbation as a Defense Technique for White Boxes

DPA and DCA attacks rely on the white box having a predictable behavior over multiple executions since multiple power- or memory traces from the white box are compared and the correlation between the traces from the different runs are calculated [2, 1]. If the white box behave differently and a bit random at every execution, the idea is that a DCA or a DPA attack will be harder to mount and more recorded traces will be required to calculate a good correlation coefficient and produce a good guess on the embedded secret key.

The underlying thought behind changing the behavior of software to create safer programs can be traced back to research within the field of software diversity in 1994. It is shown that multiple software implementations able to solve the same problem can result in the whole system of software implementations more tolerant to faults and also becomes safer to use [37]. Perturbing code at runtime is similar to diversification of software since it creates alternative behaviors and might force the running implementation into alternative execution paths. Researchers have found that software diversity by some types of perturbation also makes their cryptographic implementations a little bit more secure to side channel attacks as well [9].

3.2 Perturbing Arithmetic Calculations

Using polymorphic functions is one technique that can be implemented in order to generate alternative execution paths for a white box. The critical func-

tion inside the code we wish to protect gets copied a number of times, each copy is a different implementation of the same function and all the functions are therefore different. Whenever the original function is called, a perturbation engine redirects the function call to any of the polymorphic functions instead. The behavior of the white box will then appear different at each execution. The deciding between which polymorphic function to redirect the instruction pointer to is done randomly at runtime. This technique has been proved to harden the white box [9].

Bringer et. al. [14] introduced perturbations by perturbing a system of multivariate polynomials by adding polynomials whose values are zero with probability greater than 0.5. In some cases, these added polynomials would interfere with the intermediate results but the parts of the code where these extra polynomials were added are always run multiple times and the correct intermediate result to pass forward to the next part of the program gets determined by an internal popular vote.

Inserting instructions into the code of the white box to cause a perturbation has been explored by Ambrose et. al. [38]. At certain areas in the code random extra instructions are added. The extra instructions do not have an effect on the correctness of the output. The instruction is for example an 'add' between a randomly chosen register and the zero register. The result is written back to the randomly chosen register. With this method no values change but the execution path is different every execution since more instructions are added. The authors state their technique can be used to prevent DPA power trace attacks on their tested white box implementations.

3.3 Correctnessbreaking Countermeasures

The area of creating countermeasures to cryptographic algorithms where the countermeasure itself breaks the correctness of the program is hard to find research on. Banik et al [39] goes through and implements four common countermeasures for white box implementations and describes the idea behind why the different countermeasures are supposed to help against side channel attacks. It is assumed that a countermeasure is something that preserves the correctness of the program in need of protection.

Chapter 4

Research Questions

4.1 Correctness Attraction of Perturbed Cryptographic White Box Implementations

Programs in cryptography has shown less correctness attraction compared to the overall average of all programs tested by Danglot [36]. In order to use the perturbations as a countermeasure for side channel attacks on a white box, the white box implementation under perturbation still needs to produce usable output for its original intended use case.

1. How likely is a reliable ciphertext returned from a cryptographic white box implementation being randomly perturbed under runtime?

4.2 Execution Time of Perturbed White Box Implementations

The alternative execution paths a white box implementation is forced to take during random perturbation at runtime might have an impact on the execution time of the implementation. A countermeasure is only usable if not too much extra overhead is added in comparison to the original runtime of the implementation.

2. For randomization points with correctness attraction > 0.9 What is the runtime overhead of the considered cryptographic implementations under random perturbation of binaries?

4.3 Perturbed Cryptographic Implementations Resistance Against Side Channel Attacks

A countermeasure, in order to protect your cryptographic implementation against side channel attacks, is to make your implementation behave random out of an attacker's point of view. The practice of correctness attraction introduces perturbations to a program which forces it to follow alternative execution paths. In many cases the correctness attraction of the program under perturbation will still be high enough to make the program usable for its original intended tasks. A combination of randomness and perturbations will result in random perturbations. These random perturbations when inserted in a program will force the program to randomly follow different execution paths.

The random perturbations will be unsound dynamic randomizations. The single perturbation point is dynamically activated at random but each perturbation point is statically inserted at a specific location in the code at compile time according to definitions by Baudry et. al. [40].

3. Does the probability decrease of finding the correct secret encryption key embedded inside a white box, using the 'Differential Computation Analysis' side channel attack, when attacking a white box under random perturbation?

Chapter 5

Experimental Methodology

5.1 Threat Model

We assume the attacker wants to extract one of the round keys from the target white box and that the cryptography implementation is broken when one round key has leaked. This is a valid assumption since the hardest part of an attack is completed when a round key is extracted, as will be explained in the following. One round key can be found in time t. The time it takes to retrieve the original encryption key can be expressed as $O(t \cdot C + B)$ where B and C is constants. Independent of which round key is found, the same number of unknown bits remain after reversing the AES key schedule. These remaining bits needs to be brute forced. The time it takes to brute force the remaining bits are denoted as B. It is however possible to reduce the number of bits needed to be brute forced by once again conducting the attack to find yet another round key. Every newly found round key contributes to revealing some of the previously unknown bits in the original secret encryption key. The constant C denotes the number of times you run the attack to retrieve round keys. After finding one round key and reversing the key schedule, a different number of unknown bits remain depending on the AES implementation, 10 bits remain unknown for AES-128, 12 bits for AES-192, and 14 bits for AES-256. Therefore, the most time-consuming part of the attack is to find the round key. The rest of the attack happens in constant time.

The attacker is also assumed to have full control over the machine the white box implementation is running on. The attacker has access to the compiled binary files of the white box implementation and can supply input to the white box and get the corresponding cipher text produced by the white box.

We assume the attacker is not in control of the random number generator

on the machine running the white box implementation as the perturbations used as a protection technique relies upon the random number generator. It is normal from a cryptographic point of view to make this assumption [9].

5.2 Scope of Methodology

The perturbations will take place in the LLVM IR bytecode level on AES white box implementations found inside the Deadpool repository [29]. By choosing to insert perturbations on a system level close to the real byte code of the executable, we reduce the risk of the compiler undoing our perturbations. Only values, when used in calculations, will get perturbed. The three chosen white boxes examined in this study has available source code written in C for the ease of compiling and inserting the perturbations into the implementation of the white box.

The compile time that arises when inserting a perturbation, one at a time, on many perturbation points and generating a separate executable for all those perturbation points grows linearly with the number of perturbation points. Therefore only a fraction of all perturbation points in each white box implementation will be explored in this study.

A compiled version of the white box implementation will only be randomly perturbed at one of its perturbation points when executed. To investigate a combination of k perturbation points among n possible perturbation points would result in a search space of $\binom{n}{k}$. The search space grows very quick with a growing number of n even for small numbers of k. In order to constrain the search space, either n or k has to be small. Both options are worth studying but in this study the choice has been made to keep k at 1 in order to be able to investigate a larger number of perturbation points.

5.3 Inserting Perturbations Into White Box Implementations

The data and calculations in this thesis are made possible upon the transformation of bytecode. The transformation of bytecode occurs inside the LLVM intermediate representation (LLVM IR) of the various white box implementations. The transformation consists of inserting the perturbation engine alongside the code and then embedding function calls to it at any chosen perturbation point in the bytecode. The fraction of perturbation points tested are evenly spread throughout the binary.

The white box implementation gets the code for generating a perturbation linked at compile time from its original language to LLVM IR. The perturbation code is setup to be run at a specific perturbation point, using a customized opt pass passing over the LLVM IR. Every compilation of the white box source code with a specified perturbation point at a certain perturbation probability results in a new executable with the given perturbation characteristics.

The system abstraction layer, defined by Broman [25], at which the perturbations are inserted is located low in the stack of system abstraction layers at the bytecode level. The LLVM IR has visual similarities with assembler code. But LLVM IR uses so called "values" where assembler would instead use registers. It is these values the perturbation is randomly applied to. A perturbation protocol defined in [36] is the PONE protocol. Every perturbation will consist of adding the value one (1) to the current value. But this does not happen every time a perturbation point is hit during the program execution. The perturbation will activate randomly at a predefined probability. This is illustrated in listing 5.2 and 5.3. Every perturbation point may be hit multiple times in an execution if the perturbation point is located inside a program loop. This means that a perturbation may be activated multiple times during a single program execution.

Information about the type stored inside a value, regarding if it is an integer, char or boolean is lost in the transformation to LLVM IR. All the above mentioned gets transformed to integer values with different bit size. Therefore, the differently defined types in C, chars, integers and booleans are all perturbed the same way in this study. All integer representations in LLVM IR with a length of 8, 16, 32 and 64 bit are subject to perturbation.

Every calculation operation that takes two arguments in LLVM IR is called a binary operation. In the LLVM IR there exists 18 binary operations, see table 5.1 and table 5.2 [41] that offer three perturbation points each. Each operation has three values consisting of two arguments and a result. The instructions containing values subject to perturbation are displayed in the list below. Other instructions such as load, store, initialization of values and more are not subject to perturbation.

5.3.1 Example of Original Code in LLVM IR

The following sample of LLVM IR code consists of a multiplication instruction surrounded by load and store instructions. The following examples will show how this code is transformed to insert perturbations.

Binary operations	
Add	Fadd
Sub	Fsub
Mul	Fmul
Udiv	Sdiv
Fdiv	Urem
Srem	Frem

Table 5.1: Binary Operations in the LLVM IR language

Bitwise Binary Operations	
Shl	Lshr
Ashr	And
Or	Xor

Table 5.2: Bitwise Binary Operations in the LLVM IR language

Listing 5.1: Example of code in LLVM IR

```
...
%8 = load i8, i8* %4, align 1
%9 = sext i8 %8 to i32
%10 = mul nsw i32 %7, %9 // Target multiplication instruction
%11 = trunc i32 %10 to i8
store i8 %11, i8* %5, align 1
...
```

5.3.2 Example of a Perturbation on the First Argument in a Binary Instruction

A perturbation can take place in any of the two arguments to a binary instruction. The implementation for perturbing the second argument to a binary instruction is the same as shown below for the first argument. A perturbation value is generated and stored in the $\%inc$ value. This value is now either one (1) or zero (0) depending on the predefined probability. This perturbation is added to the original value of the argument. The randomly perturbed argument is now used in the original multiplication and the program carries on with the instructions as originally intended.

Listing 5.2: Example of a perturbation on the first argument in a binary instruction

```
...
%8 = load i8, i8* %4, align 1
%9 = sext i8 %8 to i32
%perturbation_i64 = call i64 @pone()
%perturbation = trunc i64 %perturbation_i64 to i32
%inc = add i32 %9, %perturbation
%10 = mul nsw i32 %7, %inc
%11 = trunc i32 %10 to i8
store i8 %11, i8* %5, align 1
...
```

5.3.3 Example of a Perturbation on the Result Value in a Binary Instruction

A perturbation can take place in the resulting value of a calculation. The resulting value of the original multiplication is stored in a $\%tmp$ value. A perturbation is generated which will store one (0) or one (1) with a predefined probability in the $\%perturbation$ value. This perturbation is then added to the temporary result value from the multiplication and stored as the value originally intended to store the multiplication result.

Listing 5.3: Example of a perturbation on the result value in a binary instruction

```
...
%8 = load i8, i8* %4, align 1
%9 = sext i8 %8 to i32
%tmp = mul i32 %7, %9 // Original mul instruction
%perturbation_i64 = call i64 @pone() // Perturbation generated
%perturbation = trunc i64 %perturbation_i64 to i32 // ↩
    Perturbation converted
%tmp2 = add i32 %tmp, %perturbation // Perturbation added to ↩
    the original multiplications  result value
%10 = trunc i32 %tmp2 to i8 // The perturbed value gets ↩
    converted into the right bit length
store i8 %10, i8* %5, align 1
...
```

5.4 Correctness Attraction of Perturbed Cryptographic White Box Implementations

The correctness attraction of the randomly perturbed white box is derived from comparing the output of the randomly perturbed white box to the output from the original white box. The input given to the original white box, and the

perturbed white box is randomly generated. Once the input string is generated, the input string is feed to the original- and the perturbed white box. The output from the two is compared as the figure 5.1 shows. This procedure will repeat 1000 times. Every time an input gets produced by the perturbed white box which is equivalent to the input produced by the original white box the output is considered a "Correct output". If the perturbed implementation experiences a runtime error for a specific input, the output for that run is counted as a faulty output.

Correctness attraction is derived for perturbation probabilities of 0.5, 0.10, 0.50, 0.90 and 0.99. The perturbation probabilities are chosen to make it easier to extrapolate the correctness attraction for a specific perturbation point over different perturbation probabilities between the examined probabilities values.

$$Correctness\ attraction = \frac{Number\ of\ correct\ outputs}{Number\ of\ outputs}$$

The 10 points showing the highest correction attraction at the perturbation probability of 90% will be examined in further experiments regarding execution time and attack resistance. The intuition is that if a point shows a high correctness attraction at 90% it will still show high correctness attraction at lower perturbation probabilities.

5.5 Execution Time of Perturbed White Box Implementations

Execution time is measured as the sum of the system time and the user time of a process. A python runner spawns a new child process for executing an instance of the white box. The execution time of 1000 individual executions of the white box is recorded. Input to the white box is randomly generated. Only the execution time of a successful termination is recorded.

The top ten points showing the highest correction attraction at perturbation probability of 90% is chosen and execution time is examined for 90%, 50% and 10% perturbation probability of these chosen points. Since correctness is a requirement in order to have a functioning program, only the points with the highest correctness attraction are chosen to proceed with.

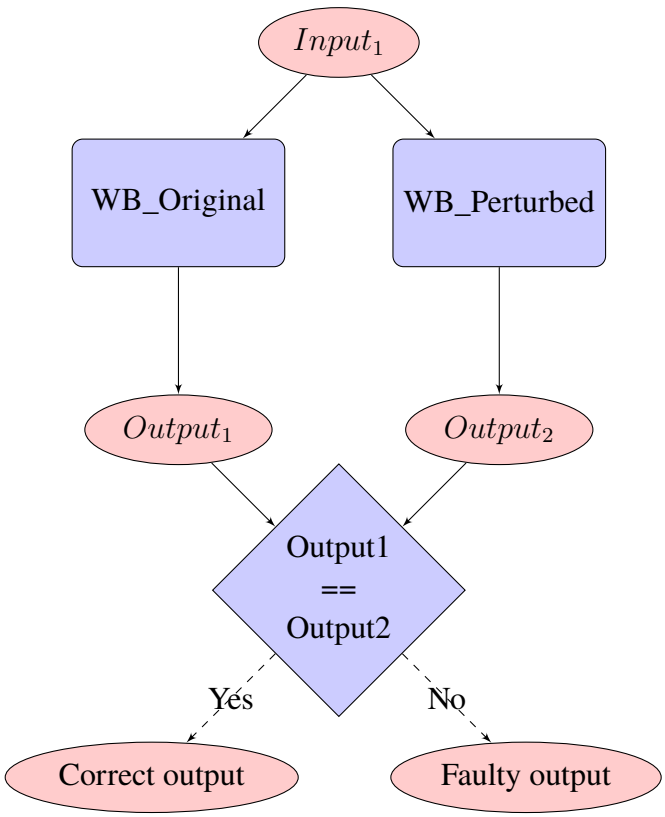

Figure 5.1: Procedure for deriving correctness attraction for a specific perturbed white box

5.6 Perturbed Cryptographic Implementations Resistance Against Side Channel Attacks by Introducing 'Attack Scores'

As described in 2.3, the complete DCA attack using Deadpool and Daredevil is to some extent non deterministic. Therefore, a scoring system is set in place to assess the success of an attack. The score goes from 0.00, which indicates a completely failed attack, up to 1.00 which indicated a completely successful attack. The score is derived with respect to the number of correctly guessed bytes divided by the total number of bytes in the round key. On the scale of 0.00 to 1.00 there are 16 different possible scores an attacked can be scored with.

The score of an attack on a regular white box differs with respect to the white box implementation and the number of memory traces collected. Some white boxes are harder to perform a successful attack on than others due to the creator's design decisions. Therefore, a reference score is set for the three chosen white box implementations. To get the reference score for the three white boxes 200 memory traces were used in the attack conducted on the Ches2016 challenge, 80 memory traces were used in the Kryptologik challenge and 20 traces were used in the NSC2013 challenge.

The perturbed white box is considered harder to crack if it gets a lower attack score compared to the reference attack. A lower score indicates there is a need to record more memory traces in order to get a reliable key guess. To record more memory traces requires more time and processor power, therefore it is considered harder to break an implementation that needs more recorded memory traces.

Chapter 6

Experimental Results

6.1 Correctness Attraction of the Perturbed Cryptographic White Box Implementations

Recall research question number one: **How likely is a reliable ciphertext returned from a cryptographic white box implementation being randomly perturbed under runtime?** The question is answered by analysing correctness attraction for the three white boxes and its perturbation points set to perturb the white box at different perturbation probabilities.

Ches2016

Ches2016 implementation yield 60492 possible perturbation points. 604 of these evenly distributed throughout the binary were chosen and investigated. In total 3020 binaries with different probabilities of 5, 10, 50, 90 and 95% were generated. The experiment took about three hours to run.

The Ches2016 white box has no antifragile or robust perturbation points, as defined in section 2.5, with the perturbation probability set to 90%, recall section 5.4. The correctness attraction for most points are decreasing linearly with a slower rate than the perturbation probability increases. Correctness attraction for all the tested perturbation point inside the Ches2016 white box are shown as a scatter plot in figure 6.1. The mean number of activations for all perturbation points were one (1) activation. The median number of activations per perturbation point were one (1) activation.

The perturbation points showing the highest correction attraction at 90%

Figure 6.1: Correctness attraction for white box Ches2016

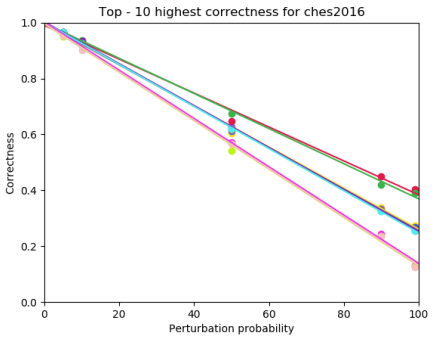

Figure 6.2: Correctness attraction of the 10 points showing the highest correctness attraction at perturbation probability 90%.

are singled out and piloted with a linear approximation, in one color and one line per perturbation point in figure 6.2. The highest correction attraction at 90% is 0.449. All of the chosen top points in figure 6.2 gets activated once per execution in general. The lowest correctness attraction among the chosen top points at perturbation probability of 90% is 0.235.

Out of the top ten chosen points at perturbation probability 90% five of them are not reoccurring among the points showing the highest correctness attraction at perturbation probability of 50% or 10%.

The probability of getting a correct ciphertext from the perturbed Ches2016 white box decreases linearly with increasing perturbation probability.

Kryptologik

Kryptlologik implementation yield 23721 possible perturbation points. 230 of these evenly distributed throughout the binary were chosen and investigated. In total 1150 binaries with different probabilities of 5, 10, 50, 90 and 95% were generated. The experiment took about two hours to run.

The Kryptologik white box has eight robust perturbation points, as defined in section 2.5, and no antifragile perturbation points at the perturbation probability set to 90%, recall section 5.4. The correctness attraction for most points are decreasing linearly with a slower rate than the perturbation probability increases. Correctness attraction for all the tested perturbation point inside the Kryptologik white box are shown as a scatter plot in figure 6.3. The mean number of activations for all perturbation points were one (1) activation. The

median number of activations per perturbation point were one (1) activation.

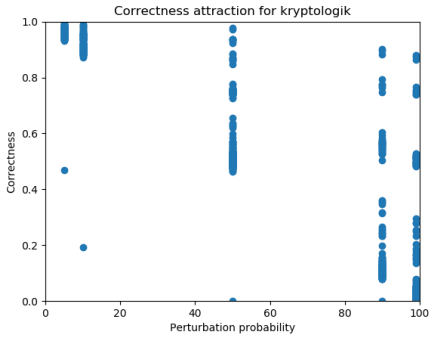

Figure 6.3: Correctness attraction for white box Kryptologik

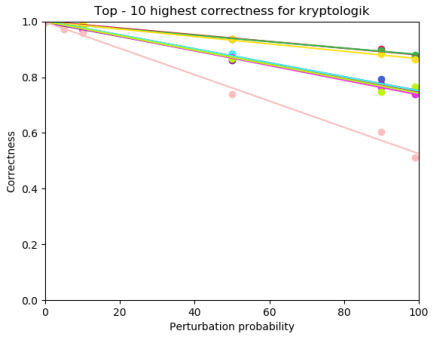

Figure 6.4: Correctness attraction of the 10 points showing the highest correctness attraction at perturbation probability 90%.

The perturbation points showing the highest correction attraction at 90% are singled out and piloted with linear approximation, one color and one line per perturbation point in figure 6.4. The highest correction attraction at 90% is 0.901. All of the chosen top points in figure 6.4 gets activated once per execution in general. The lowest correctness attraction among the chosen top points at perturbation probability of 90% is 0.604.

Out of the top ten chosen points at perturbation probability 90% two are not reoccurring among the points showing the highest correctness attraction at perturbation probability of 50% or 10%.

The probability of getting a correct ciphertext from the perturbed Kryptologik white box decreases linearly with increasing perturbation probability in a similar fashion compared to the Ches2016 white box.

NSC2013

NSC2013 generator implementation yield 408 possible perturbation points. 400 of these evenly distributed throughout the binary were chosen and investigated. In total 2000 binaries with different probabilities of 5, 10, 50, 90 and 95% were generated.The experiment took about 40 minutes to run.

The NSC2013 white box has no robust or antifragile perturbation points, as defined in section 2.5, at the perturbation probability set to 90%, recall section 5.4. The correctness attraction for most points are decreasing linearly with a

faster rate than the perturbation probability increases. Correctness attraction for all the tested perturbation point inside the NSC2013 white box are shown as a scatter plot in figure 6.5. The mean number of activations for all perturbation points were 6294 activations. The median number of activations per perturbation point were 40 activations.

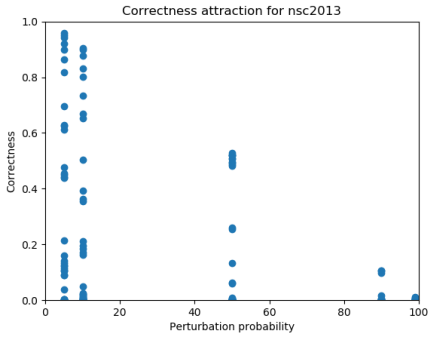

Figure 6.5: Correctness attraction for white box Kryptologik

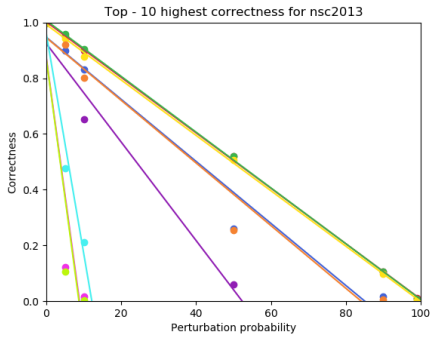

Figure 6.6: Correctness attraction of the 10 points showing the highest correctness attraction at perturbation probability 90%.

The perturbation points showing the highest correction attraction at 90% are singled out and piloted with linear approximation, one color and one line per perturbation point in figure 6.6 The highest correction attraction at 90% is 0.106. Three perturbation points among the points in figure 6.4 gets activated once per execution. Rest of the points gets activated at a mean between 2 and 512 times per execution. The five lowest correctness attraction among the chosen top points at perturbation probability of 90% is 0.00.

Out of the top ten chosen points at perturbation probability 90% five of them are not reoccurring among the points showing the highest correctness attraction at perturbation probability of 50% or 10%. These five points are the points showing a correctness attraction of 0.00

The probability of getting a correct ciphertext from the perturbed NSC2013 white box decreases linearly with increasing perturbation probability. The decrease in correctness is much larger when compared to Ches2016 and Kryptologik white boxes.

6.1.1 Key Takeaways

The correctness attraction for the three white boxes behave different from each other even though they implement the same algorithm. The correctness of a returned ciphertext is a linear function dependent on the implementation, the perturbation point and the perturbation probability. Therefore no general in depth assumption can be drawn as to how likely a reliable ciphertext is returned from a cryptographic white box implementation being randomly perturbed under runtime

6.2 Execution Time of Perturbed Cryptographic Implementations

Recall the research question number two, **For randomization points with correctness attraction >.9: What is the runtime overhead of the considered cryptographic implementations under random perturbation of binaries?** The question is answered by running each implementation and its transformations 1000 times with randomly generated input and comparing the runtimes of the transformed implementation with its original implementation in the graphs bellow.

Ches2016

The execution time for the Ches2016 white box implementation takes on a Poisson distribution with an expected value close to 3 milliseconds. In the graphs below the distribution of execution times are plotted with the reference execution times under the blue curve which has been shaded light blue for easier comparison.

The other execution times with a perturbation inserted is plotted on top of the reference curve in the corresponding figure for the corresponding perturbation probability in figure 6.7, figure 6.8 and figure 6.9. The points are labeled $a - j$ and are the chosen perturbation points showing highest correctness attraction at the perturbation probability of 90% in figure 6.2.

There are no mean values lower than the reference values and the variance is either the same or smaller than the reference variance.

With one perturbation inserted and activated with different perturbation probabilities in the Ches2016 white box yields no change in execution time.

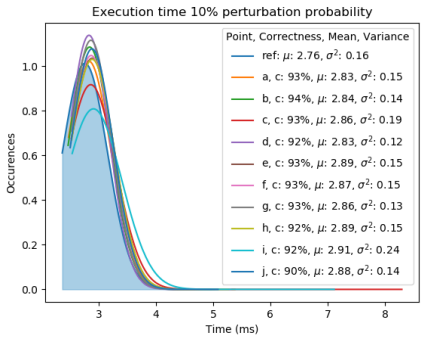

Figure 6.7: Execution time for white box with perturbations set to 10%

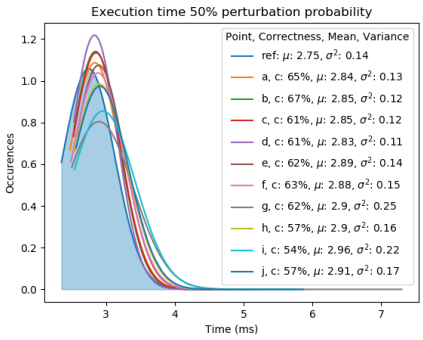

Figure 6.8: Execution time for white box with perturbations set to 50%

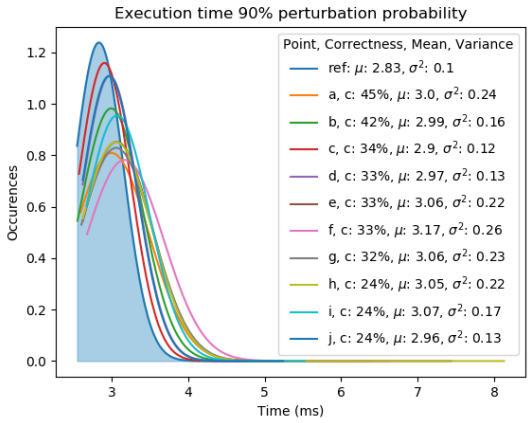

Figure 6.9: Execution time for white box with perturbations set to 90%

Kryptologik

The execution time for the Kryptologik white box implementation takes on a Poisson distribution with an expected value close to 3.9 milliseconds (ms). In the graphs below the distribution of execution times are plotted with the reference execution times under the blue curve which has been shaded light blue for easier comparison.

The other execution times with a perturbation inserted are plotted on top of the reference curve in the corresponding figure for the corresponding perturbation probability figure 6.10, figure 6.11 and figure 6.12. The points are labeled $a - j$ and are the chosen perturbation points showing highest correctness attraction at the perturbation probability of 90% in figure 6.4.

There are mean values both lower and higher than the reference values for

the different perturbation probabilities and the variance is either the same or smaller than the reference variance.

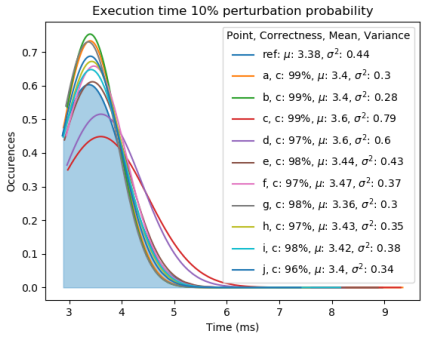

Figure 6.10: Execution time for Kryptologik white box with perturbations set to 10%

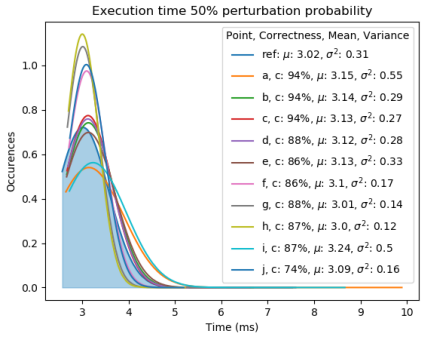

Figure 6.11: Execution time for Kryptologik white box with perturbations set to 50%

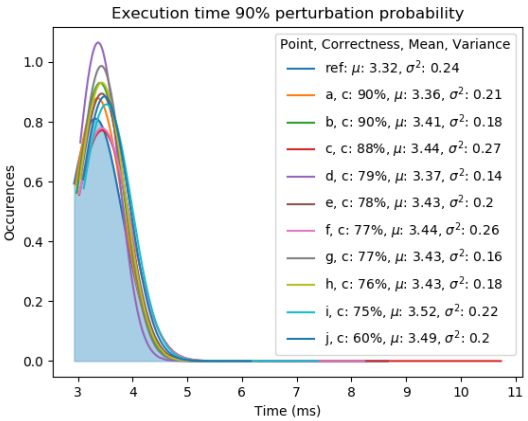

Figure 6.12: Execution time for Kryptologik white box with perturbations set to 90%

With one perturbation inserted and activated with different perturbation probabilities in the Kryptologik white box yields no change in execution time. The same result as for the previously tested white box Ches2016.

NSC2013

The execution time for the NSC2013 white box implementation takes on a Poisson distribution with an expected value close to 6.2 ms. In the graphs be-

low the distribution of execution times are plotted with the reference execution times under the blue curve which has been shaded light blue for easier comparison.

The other execution times with a perturbation inserted is plotted on top of the reference curve in the corresponding figure for the corresponding perturbation probability 6.13, 6.14 and 6.15. The points are labeled $a - j$ and are the chosen perturbation points showing highest correctness attraction at the perturbation probability of 90% in figure 6.6.

There are no mean values lower than the reference values for the different perturbation probabilities. The difference in means is clearly visible on all three figures 6.13, 6.14 and 6.15. The variance is either the same or smaller than the reference variance.

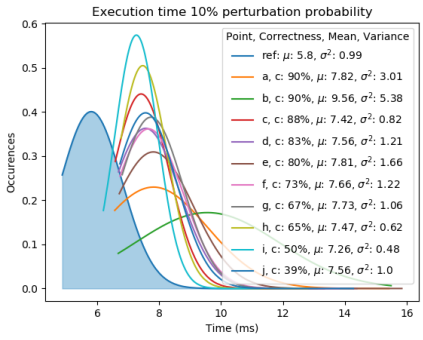

Figure 6.13: Execution time for NSC2013 white box with perturbations set to 10%

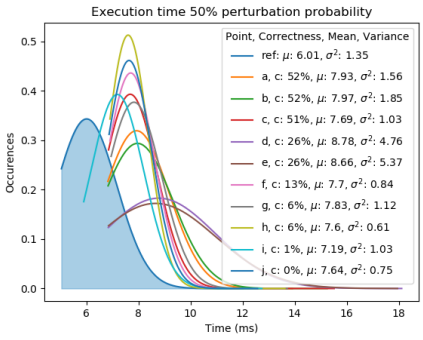

Figure 6.14: Execution time for NSC2013 white box with perturbations set to 50%

With one perturbation inserted and activated with different perturbation probabilities in the NSC2013 white box yields a small change in execution time. The change in execution time is the same regardless of perturbation. The result differs from the two previously tested white box which did not display any change in execution time.

6.2.1 Key Takeaways

One of the three implementations, NSC2013 showed an increase in execution time with inserted perturbation points compared to its original implementa-

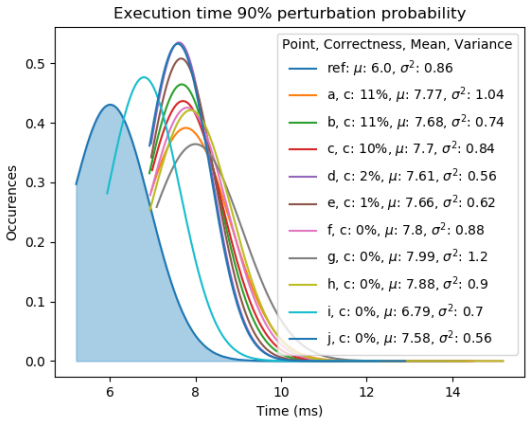

Figure 6.15: Execution time for NSC2013 white box with perturbations set to 90%

tion. The increase in execution time seems to be small and nondependent of the chosen perturbation point or perturbation probability. The other perturbed whitboxes did not display any difference in execution time compared to the original implementation for any chosen perturbation point or perturbation probability. For randomization points with correctness attraction > 0.9: None of the white boxes showed a significant runtime overhead to raise concerns for the usability of the perturbed implementation.

6.3 Perturbed Cryptographic Implementations Resistance Against Side Channel Attacks

Recall the research question number three: **Does the probability decrease of finding the correct secret encryption key embedded inside a white box, using the 'Differential Computation Analysis' side channel attack when attacking the white box under random perturbation?** The question is answered by running 30 attacks on each implementation and its transformations and comparing the attack scores, recall section 5.6 explaining the attack scores used to compare the difficulty of conducting an attack.

6.3.1 Reference Attacks

30 individual attacks on each white box implementation gave the reference attack scores in figure 6.16 as three box plots. The first and second quartile for

the attack score regarding the reference attack on the Ches2016 white box are the same value and therefore lies on top of each other.

The reference boxplot for each implementation is incorporated in the graphs showing attack score for the corresponding implementations transformations. The three reference boxplots are to be examined separate from each other and only compared to its respective implementation's transformations. The highest and lowest of the horizontal blue lines represent the highest and lowest attack score given from the reference attacks. The three lines in between are the three quartiles. The area inside the box of the boxplot in figure 6.16 are shaded blue.

Recall the first and second quartile for Ches2016 being the same, this reflects the fact that only in total of four horizontal lines are visible when viewing the Ches2016 reference attack score boxplots.

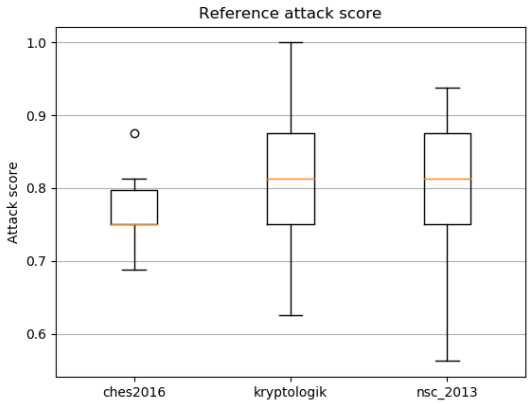

Figure 6.16: Reference attack score over 30 attacks, for the 3 white boxes without perturbations

6.3.2 Attacks on Ches2016

Attacks at 10% Perturbation Probability

T-test on Ches2016 at perturbation probability of 10% shows there is no significant difference in populations between the reference attacks and the perturbation attacks for any of the ten perturbation points. A complete table over p-values is displayed in section 6.3.4.

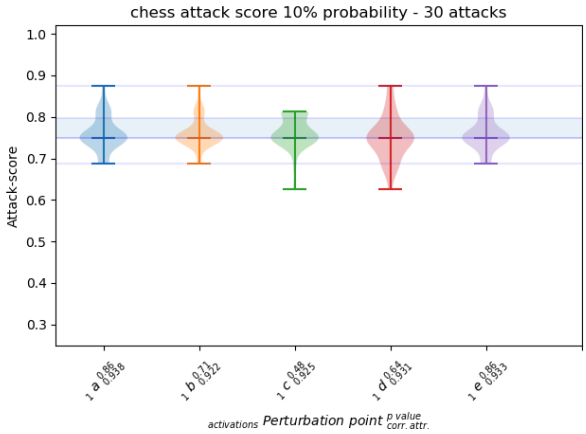

Figure 6.17: Attack score over 30 attacks for the 10 points showing highest correctness attraction at 90%, here shown with a correctness attraction of 10%

Attacks at 50% Perturbation Probability

T-test on Ches2016 at perturbation probability of 50% shows there is no significant difference in populations between the reference attacks and the perturbation attacks for any of the ten perturbation points. A complete table over p-values is displayed in section 6.3.4.

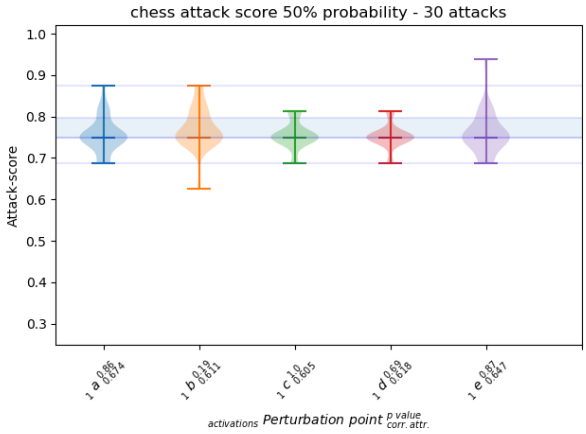

Figure 6.18: Attack score over 30 attacks for the 10 points showing highest correctness attraction at 90%, here shown with a correctness attraction of 50%

Point	10%	50%	90%
a	0.86	0.86	0.49
b	0.71	0.19	1.0
c	0.48	1.0	0.86
d	0.64	0.69	0.85
e	0.86	0.87	0.35

Table 6.1: P-values for t-test on different perturbation probabilities on the Ches2016 white box

Attacks at 90% Perturbation Probability

T-test on Ches2016 at perturbation probability of 90% shows there is no significant difference in populations between the reference attacks and the perturbation attacks for any of the ten perturbation points. A complete table over p-values is displayed in section 6.3.4.

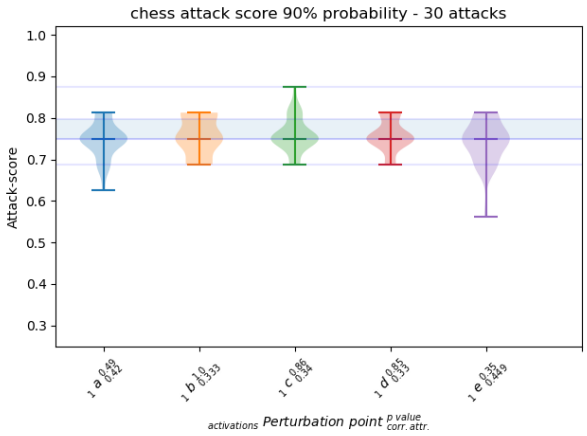

Figure 6.19: Attack score over 30 attacks for the 10 points showing highest correctness attraction at 90%, here shown with a correctness attraction of 90%

T-test Between Reference- and Perturbed- Attack Scores P-values

See p-values derived from the attack score between the original Ches2016 white box and the Ches2016 white boxes with inserted perturbations at different probabilities in table 6.1

6.3.3 Kryptologik

Attacks at 10% Perturbation Probability

T-test on Kryptologik at perturbation probability of 10% shows there is no significant difference in populations between the reference attacks and the perturbation attacks for any of the ten perturbation points. Perturbation point d with a p-value of 0.11 is the lowest p value with a positive increase in mean attack score between the populations. A complete table over p-values is displayed in section 6.3.4.

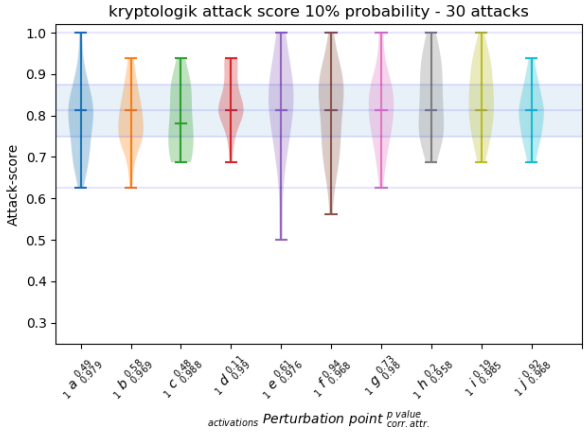

Figure 6.20: Attack score over 30 attacks for the 10 points showing highest correctness attraction at 90%, here shown with a correctness attraction of 10%

Attacks at 50% Perturbation Probability

T-test on Kryptologik at perturbation probability of 50% shows there is no significant difference in populations between the reference attacks and the perturbation attacks for any of the ten perturbation points. Perturbation point d with a p-value of 0.11 is the lowest p value with a positive increase in mean attack score between the populations. A complete table over p-values is displayed in section 6.3.4.

Attacks at 90% Perturbation Probability

T-test on Kryptologik at perturbation probability of 90% shows there is no significant difference in populations between the reference attacks and the perturbation attacks for any of the ten perturbation points. Perturbation point d with

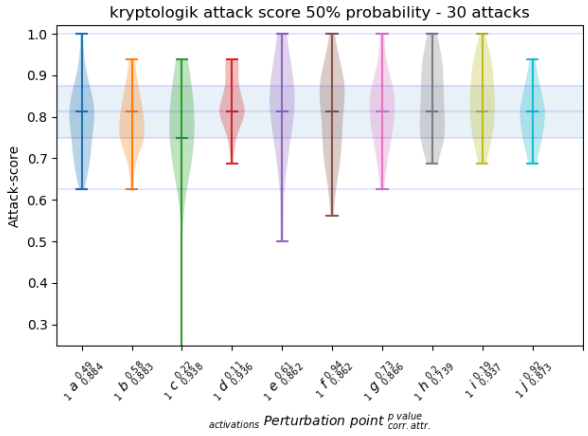

Figure 6.21: Attack score over 30 attacks for the 10 points showing highest correctness attraction at 90%, here shown with a correctness attraction of 50%

a p-value of 0.11 is the lowest p value with a positive increase in mean attack score between the populations. A complete table over p-values is displayed in section 6.3.4.

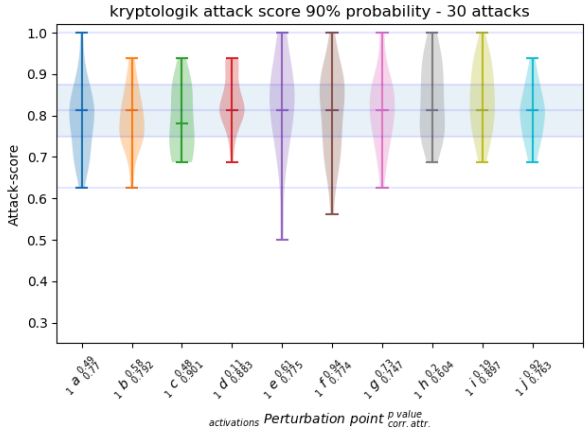

Figure 6.22: Attack score over 30 attacks for the 10 points showing highest correctness attraction at 90%, here shown with a correctness attraction of 90%

T-test Between Reference- and Perturbed- Attack Scores p-values

See p-values derived from the attack score between the original Kryptologik white box and the Kryptologik white boxes with inserted perturbations at different probabilities in table 6.2

Point	10%	50%	90%
a	0.49	0.49	0.49
b	0.58	0.58	0.58
c	0.48	0.22	0.48
d	0.11	0.11	0.11
e	0.61	0.61	0.61
f	0.94	0.94	0.94
g	0.73	0.73	0.73
h	0.2	0.2	0.2
i	0.19	0.19	0.19
j	0.92	0.92	0.92

Table 6.2: P-values for t-test on different perturbation probabilities on the Kryptologik white box

6.3.4 NSC2013

Attacks at 10% Perturbation Probability

T-test on NSC2013 at perturbation probability of 10% shows all but one with no significant difference in populations between the reference attacks and the perturbation attacks for any of the ten perturbation points. Perturbation point i with a p-value of 0.03 is the lowest p value with a positive increase in mean attack score between the populations. A complete table over p-values is displayed in section 6.3.4.

Attacks at 50% Perturbation Probability

T-test on NSC2013 at perturbation probability of 50% shows there is no but four significant differences in populations between the reference attacks and the perturbation attacks for any of the ten perturbation points. Perturbation points e, f, g and j all with p-values in the range of 0.00 to 0.07 with both positive and negative changing means. A complete table over p-values is displayed in section 6.3.4.

Attacks at 90% Perturbation Probability

T-test on Kryptologik at perturbation probability of 90% shows there is no significant difference in populations between the reference attacks and the perturbation attacks for all but two points, e and i. P values both at 0.0 with a positive

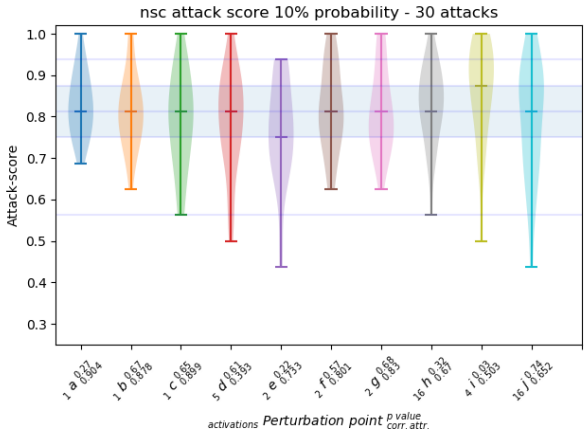

Figure 6.23: Attack score over 30 attacks for the 10 points showing highest correctness attraction at 90%, here shown with a correctness attraction of 10%

and negative increase in mean attack score between the populations. A complete table over p-values is displayed in section 6.3.4.

T-test Between Reference- and Perturbed- Attack Scores P-values

See p-values derived from the attack score between the original Kryptologik white box and the Kryptologik white boxes with inserted perturbations at different probabilities in table 6.3

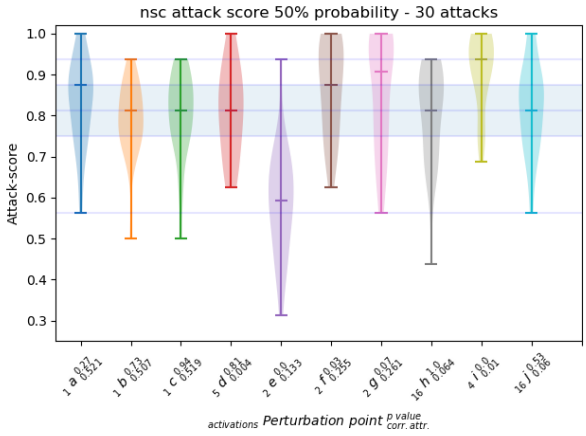

Figure 6.24: Attack score over 30 attacks for the 10 points showing highest correctness attraction at 90%, here shown with a correctness attraction of 50%

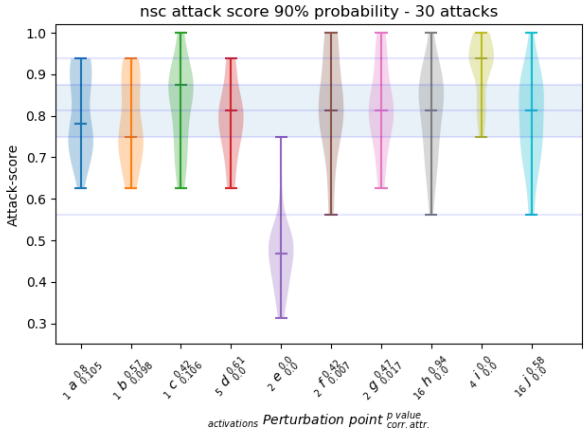

Figure 6.25: Attack score over 30 attacks for the 10 points showing highest correctness attraction at 90%, here shown with a correctness attraction of 90%

Point	10%	50%	90%
a	0.27	0.27	**0.8**
b	0.67	0.73	0.57
c	0.65	0.94	0.42
d	0.61	0.81	0.61
e	0.22	$\mathbf{1.30 \cdot 10^{-9}}$	$\mathbf{1.73 \cdot 10^{-20}}$
f	0.57	**0.03**	0.42
g	0.68	**0.07**	0.47
h	0.32	1.0	0.94
i	**0.03**	$\mathbf{5.21 \cdot 10^{-5}}$	$\mathbf{5.41 \cdot 10^{-6}}$
j	0.74	0.53	0.58

Table 6.3: P-values for t-test on different perturbation probabilities on the NSC2013 white box

6.4 Key Takeaways

For the Ches2016 and Kryptologik white boxes the perturbations did not influence the attack score. The third white box, NSC2013, had five perturbation points that affected the attack score. The correctness attraction for these points are however very low and yields the implementation unusable in most scenarios. The attack score has in one of the cases decreased and in the four other cases increased. The probability of finding the correct secret encryption key embedded inside a white box can therefore in some cases decrease or increase, when attacking the white box under random perturbation using the 'Differential Computation Analysis' side channel attack.

Chapter 7

Discussion

7.1 Comments on Correctness Attraction of Perturbed Cryptographic White Box Implementations

It is shown in studies by Danglot et al. [36] that correctness attraction for algorithms within cryptography shows less correction attraction than the average algorithm under perturbation. The results in this study further strengthens those findings. The three white box implementations, Ches2016, Kryptologik and NSC2013 all behave poorly with respect to correctness attraction at high perturbation probabilities with no anti fragile perturbation points and very few robust perturbation points.

Further on one can see that even though the three white boxes are implementations of the same AES algorithm, their specific implementations are still vastly different from one another and produces different correctness attraction. The correctness attractions for the white boxes seen in this study can be grouped into two different profiles. The first profile producing only correctness attraction with values higher or equal to the perturbation probability. The second profile producing correctness attraction with values lower or equal to the perturbation probability. The correctness attraction for Ches2016 and Kryptologik falls into the first profile. The results for research questions two and three further show that Ches2016 and Kryptologik will have similar behaviors regarding execution time- and attack score under perturbation which could be indicated by them having similar perturbation profiles.

The graphs 7.1, 7.2 and 7.3 shows the span of where correctness attraction is expected to be found for the different white boxes and different perturbation

probabilities. The highest, lowest and middle correctness attraction values are displayed inside the blue span as shaded blue lines. The two types of perturbation profiles can be distinguished where the first profile is with figures 7.1 and 7.2, while profile two is in figure 7.3.

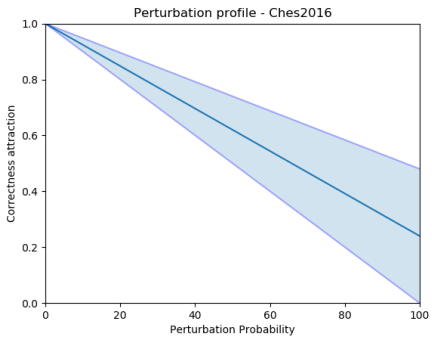

Figure 7.1: Span of correctness attraction for Ches2016

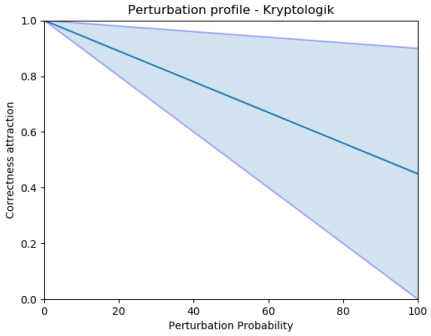

Figure 7.2: Span of correctness attraction for Kryptologik

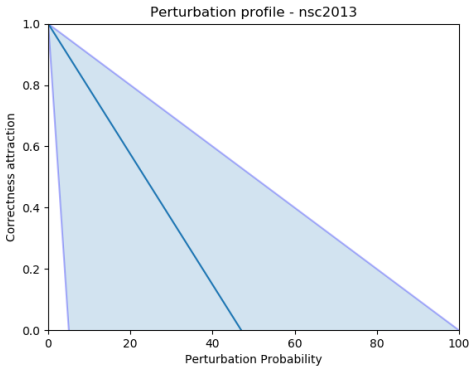

Figure 7.3: Span of correctness attraction for NSC2013

7.2 Comments on Execution Time of Perturbed White Box Implementations

Here the first perturbation profile consisting of Ches2016 and Kryptologik showed no significant increase in execution time with any of the perturbation points inserted into the program, regardless of perturbation probability. The

second perturbation profile from NSC2013 showed the same increase in execution time in all the tested perturbation points for all perturbation probabilities. But the increase in execution time is so small, on average 25%, so it would not give any noticeable delay to the end user of the white box.

The underlying cause to an increase in execution time is that there are more instructions that needs to be executed. Most of the points in NSC2013 were activated more than once each execution. Recall the mean number of activations over all perturbation points in NSC2013 were 6294 activations. The median number of activations per perturbation point were 40. In NSC2013 the points showing the highest correctness are the points among the ones that has been activated 16 times or less. 16 activations is a number significantly lower than both the mean and median activations for the white box implementation. The increase in execution time on NSC2013 is significantly different than the execution times obtained from the Ches2016 and Kryptologik and therefore is worth mentioning. But the increase is so small it should not affect any use case of the algorithm.

7.3 Perturbed Cryptographic Implementations Resistance Against Side Channel Attacks

Ches2016 and Kryptologik belonging to the first correctness attraction profile shows no significant change in the attack scores regardless of any of the inserted perturbation points, in comparison to the attack score of the original white box. Kryptologik showed no difference in attack score regardless of perturbation probability. Out of the 30 attacks conducted on Kryptologik perturbation point c, with perturbation probability 50%, in figure 6.21 we can see a statistically significant difference in the attack score for c, in comparison to the reference attack score. One attack on the perturbed white box got an attack score of 0. This value is an outlier and if it were to be removed the remaining attack scores would produce a p-value as expected compared to the p-value obtained from perturbation probability 10% and 90% on the same Kryptologik white box implementation.

NSC2013 is the white box implementation where perturbations have managed to have an impact on the attack score. Significant perturbation points are a, e, f, g and i where the mean of the attack score is moved compared to the reference attack score. P-values seems to be lower for perturbation probability 50% compared to 10% and 90% when examining the table 6.3 for points e, f, g, and i. Previous research has focused on creating alternative execution

paths inside the white box and from that making it harder to derive statistical data from its execution since the program can follow different paths. With a perturbation probability of 50% two equally likely execution paths are made available. A higher or lower perturbation probability makes the program tend to choose one execution path over the other and statistical similarities between the executions are therefore easier to derive.

Therefore, one needs to be careful if perturbations are to be used as a mitigation technique solely based on the perturbation point showing the highest correctness attraction as this study does. Even though five out of the ten chosen perturbation points managed to produce a significant shift in attack score only one perturbation point produced a lower attack score than the reference attack score. In the four other cases the attack score increased when inserting the perturbation point making the white box easier to attack. Inserting perturbations does not automatically lead to the implementation becoming more secure, in most cases it does not affect the security of the program, and in some cases the program is made more vulnerable to the DCA side channel attack as the higher attack score indicated.

7.4 Security and Correctness

A countermeasure to an attack is only any good if it does not break the correctness of the output from the original implementation. But even if a countermeasure breaks the correctness of a program down to 50% there are still methods of figuring out the correct output with high probability that would satisfy most needs and use cases.

When conducing a t-test a difference in means is considered significant if the p-value gets below 0.1. The p-values obtained from the experiments were in many cases very high indicating that there was no significant difference between the mean of the reference white box attack scores and the mean of the perturbed white box attack score. The difference in mean was be both positive and negative.

Furthermore, the points a, e, f, g and i in the NSC2013 white box has both broken correctness and a shift in attack scores means compared to the reference scores. The highest correctness is of 0.521 while the other points has a correctness of 0.261 or lower which yields the implementation in those cases useless. There seems to be no correlation between broken correctness and the attack score when analyzing any of the three white boxes.

The ideal scenario is displayed in the figure 7.4. A perturbation should produce correctness attraction over 0.5 within the graphs green area to the right

in order for the correct output to be easy identifiable, otherwise the perturbed implementation is deemed useless with respect to its original use case. Perturbation points which has managed to produce a positive change in attack score mean is plotted on the top part of the chart which indicates that the perturbed white box as a whole was more secure towards the conducted attack. Markers on the lower part of the chart managed to produce a negative difference in mean which indicates that the perturbed white box as a whole is more vulnerable towards the conducted attack. The perturbation point should also produce attack scores where the p-value is less than 0.1, derived from the t-test between the attacks on the perturbed white box and its reference white box. The blue areas indicate significant p-values. The marker has three different sizes indicating perturbation probability. The biggest marker corresponds to perturbation probability 90%, while the smallest marker corresponds to perturbation probability 10%. The sweet spot for a perturbation point in order to be used as a mitigator for a side channel attack would be at the top rightmost part of the graph where the green and the blue areas meet. The green and blue area displays high correctness attraction and a low p-value.

Unfortunately this study did not produce any perturbation points inside the range fulfilling both criteria of high correctness attraction and a low p-value for any of the tested implementations. Points within the lower rightmost red and blue square indicates that the white box is still usable and is significantly more vulnerable towards the attack. This study did not manage to produce any perturbation points with those features either.

7.5 Limitations

There are 2^{128} possible inputs and equally many outputs to any AES block cipher implementation and the three white boxes in this study is no exception. The experiment was conducted with 1000 randomly generated inputs. Because of the AES diffusion property, a change in any one bit of the input will cause half of the output bits to change [42], every new input gives a vastly different ciphertext. A larger search space could therefore be coveted to see if the results obtained in these studies are significant enough to draw general assumptions from.

Even though the whitebox is only used with a specific set of inputs the cipher text from the algorithm will vary widely since even a small change in input results in a large change in output. Therefore it will be hard to tweak the choice of perturbation to better fit an pre-defined set of outputs.

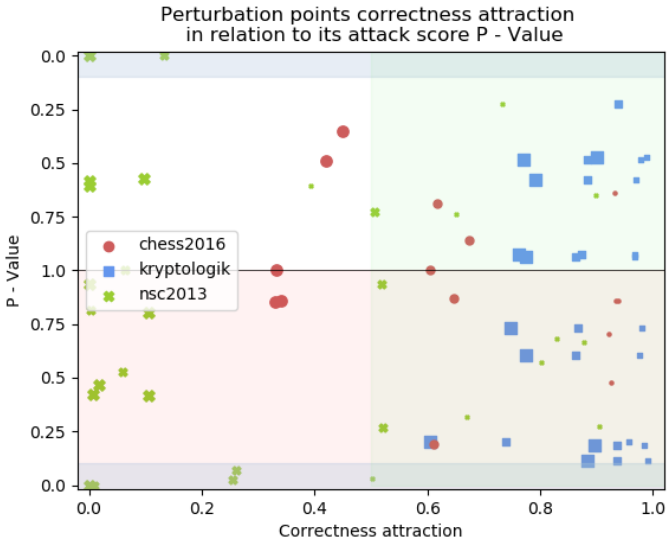

Figure 7.4: Correctness attraction in relation to attack scores for all three tested white box implementations. Values inside the colored areas, lower to the left, is better

The worst-case scenario in the time it takes to generate a white box is for the Ches2016. One randomly chosen generation with one inserted perturbation point at 50% perturbation probability took 21.072 seconds to complete on one CPU core. The compile time to convert between the C language to LLVM and back to an executable is very time consuming on these large implementations. The large number of compilations needed to generate all possible binaries, in combination with the long compilation time of each binary, makes it so it is not possible to compile and investigate all combinations of perturbation points and perturbation probabilities in reasonable time. Therefore, only a selection of perturbation probabilities and a fraction of the possible perturbation points are investigated for each implementation. The intuition to evenly spread the perturbation points among all the possible perturbation points from top to bottom in the binary has shown not to produce perturbation points with the desired properties to mitigate the side channel attack. There might be other perturbation points not investigated that have other properties which can be identified with more sophisticated approaches.

Chapter 8

Conclusions

The attempt in creating a technique to mitigate side channel attacks using random perturbations has in its current state shown not to be able to do so while still maintaining a high level of correctness. The prerequisites for being able to use the random perturbation as a mitigating technique were present though. We were still, to some extent, able to find acceptable correctness attraction for a couple of perturbation points in the white boxes. The execution time of running the white box with inserted perturbations did also show no overhead. In most cases though the mitigation technique of inserting perturbation points did not have any effect on the conducted side channel attack. One of the tested white boxes for a few of its perturbation points did show a change in the hardness of conducting a side channel DCA attack. The change in some cases made it easier to conduct the attack and in one case harder. The potential of using perturbation points to affect the difficulty of conducting a DCA side channel attack has been demonstrated in this study but it needs more research on finding the right perturbation point at the right place in the binary to make it have the desired mitigating effect.

8.1 Contributions

This research has contributed to:

- The science of correctness attraction by further examining the correctness attraction on cryptographic implementations.

No earlier research has been done on the correctness attraction with focus on algorithms solely within cryptography. Possible implementations of perturbations in programs related to cryptography is also a novel area of research.

- Providing a tool to insert perturbations into LLVM IR.

The tool can be found in the github repository [43] and is an extension of the suite of programs used to insert perturbations into other programs without the need to have any prior knowledge about the target program subject for perturbation.

- Introducing the concept of side channel mitigation techniques on white boxes which breaks the correctness of the original program.

Previous research on cryptography has only focused on attack mitigation techniques and tools which maintains the original correctness of the white box implementation. This thesis introduces the concept of applying mitigation techniques which breaks correctness. Correctness breaking techniques for mitigating side channel attack is a novel area of research.

8.2 Future Research

The intuition of choosing the perturbation points with the highest correctness attraction showed to give scantly results. Another approach of selecting perturbation points may be examined according to other criteria. Similarities between the Ches2016 and the Kryptologik implementations correctness attraction profiles exists. Classifying white boxes according to their correctness attraction profile and conducting research focused on one correctness attraction profile are scopes which could be applied in future research. Each correctness attraction profile might display its own characteristics in where the best perturbation points are located, increase in execution time and effect on attack resistance.

The only perturbation model used in this study was adding the value of one (1) to any value found in the white box. With using other perturbation models such as the subtraction of one (1) or setting the value under perturbation to zero (0), might cause other behaviors regarding correctness, execution time or attack resistance worth studying as a continuation of this study.

The tool to insert perturbations does not differentiate between perturbation points inside a loop or the once outside of loops. Will the NSC2013 correctness attraction profile be more similar to the profile of Ches2016 and Kryptologik if the tool was extended to only execute a perturbation once, even if the perturbation point is activated many times more inside a loop? The intuition is that implementing this scheme for the perturbation points which managed

to show a change in attack score will still do so with this new scheme, but the points will produce higher correctness.

The other way around can also be investigated by inserting multiple perturbations to be activated during the same execution into the Ches2016 and Kryptologik white box. Does this produce any new major changes to the attack score? A change would then indicate that inserting only one perturbation, as in this study, alongside many tens of thousands of regular program instructions is not enough supplied randomness in order to mitigate a side channel attack.

Research needs to be done in finding the perturbation points who affects the programs control flow graph the most in order to generate the most diverse control flow graph in comparison to running the program without activating the perturbation point to fool the attacker.

Bibliography

[1] Joppe W Bos et al. "Differential computation analysis: Hiding your white-box designs is not enough". In: *International Conference on Cryptographic Hardware and Embedded Systems*. Springer. 2016, pp. 215–236.

[2] Paul Kocher, Joshua Jaffe, and Benjamin Jun. "Differential power analysis". In: *Annual International Cryptology Conference*. Springer. 1999, pp. 388–397.

[3] Estuardo Alpirez Bock et al. "White-Box Cryptography: Don't Forget About Grey-Box Attacks". In: *Journal of Cryptology* (2019), pp. 1–49.

[4] Boaz Barak et al. "On the (im) possibility of obfuscating programs". In: *Annual international cryptology conference*. Springer. 2001, pp. 1–18.

[5] Paul Kocher, Joshua Jaffe, Benjamin Jun, et al. *Introduction to differential power analysis and related attacks*. 1998.

[6] Paul Kocher et al. "Introduction to differential power analysis". In: *Journal of Cryptographic Engineering* 1.1 (2011), pp. 5–27.

[7] Hyunjin Ahn and Dong-Guk Han. "Multilateral White-Box Cryptanalysis: Case study on WB-AES of CHES Challenge 2016." In: *IACR Cryptology ePrint Archive* 2016 (2016), p. 807.

[8] Olivier Billet and Henri Gilbert. "A Traceable Block Cipher". In: *Advances in Cryptology - ASIACRYPT 2003*. Ed. by Chi-Sung Laih. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 331–346. ISBN: 978-3-540-40061-5.

[9] Nicolas Belleville et al. "Automated software protection for the masses against side-channel attacks". In: *ACM Transactions on Architecture and Code Optimization (TACO)* 15.4 (2018), p. 47.

[10] *Challenges — CHES 2016 Capture the Flag*. `http://ctf.newae.com/flags/`. (Accessed on 09/05/2019).

[11] *KryptoPlus White-box Demo*. `http://kryptologik.com/`. (Accessed on 09/05/2019).

[12] *Full Disclosure: Oppida/NoSuchCon challenge has been released*. `https://seclists.org/fulldisclosure/2013/Apr/133`. (Accessed on 09/05/2019).

[13] Julien Bringer, Herve Chabanne, and Emmanuelle Dottax. "White Box Cryptography: Another Attempt." In: *IACR Cryptology ePrint Archive* 2006 (Jan. 2006), p. 468.

[14] Julien Bringer, Hervé Chabanne, and Emmanuelle Dottax. "Perturbing and protecting a traceable block cipher". In: *IFIP International Conference on Communications and Multimedia Security*. Springer. 2006, pp. 109–119.

[15] Emmanuel Prouff, Matthieu Rivain, and Régis Bevan. "Statistical analysis of second order differential power analysis". In: *IEEE Transactions on computers* 58.6 (2009), pp. 799–811.

[16] Timo Bartkewitz and Kerstin Lemke-Rust. "A high-performance implementation of differential power analysis on graphics cards". In: *International Conference on Smart Card Research and Advanced Applications*. Springer. 2011, pp. 252–265.

[17] Paul C Kocher. "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems". In: *Annual International Cryptology Conference*. Springer. 1996, pp. 104–113.

[18] Yoni De Mulder, Brecht Wyseur, and Bart Preneel. "Cryptanalysis of a Perturbated White-Box AES Implementation". In: *Progress in Cryptology - INDOCRYPT 2010*. Ed. by Guang Gong and Kishan Chand Gupta. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 292–310. ISBN: 978-3-642-17401-8.

[19] Stanley Chow et al. "White-box cryptography and an AES implementation". In: *International Workshop on Selected Areas in Cryptography*. Springer. 2002, pp. 250–270.

[20] Stanley Chow et al. "A white-box DES implementation for DRM applications". In: *ACM Workshop on Digital Rights Management*. Springer. 2002, pp. 1–15.

[21] Federico Scrinzi. "Behavioral analysis of obfuscated code". MA thesis. University of Twente, 2015.

[22] National Institute of Standards and Technology. “Advanced Encryption Standard”. In: *NIST FIPS PUB 197* (2001).

[23] U.S. Department of Commerce and NIST. *FIPS 46-3, Data Encryption Standard (DES) (withdrawn May 19, 2005)*. https://csrc.nist.gov/csrc/media/publications/fips/46/3/archive/1999-10-25/documents/fips46-3.pdf. (Accessed on 09/05/2019). 1999.

[24] Nigel P Smart. “Physical side-channel attacks on cryptographic systems”. In: *Software Focus* 1.2 (2000), pp. 6–13.

[25] David Broman. *Lecture 1: IS1500 HT17-1 Datorteknik och komponenter*. https://kth.instructure.com/courses/2535/pages/lectures. (Accessed on 09/05/2019). 2017.

[26] Paul Bottinelli and Joppe W Bos. “Computational aspects of correlation power analysis”. In: *Journal of Cryptographic Engineering* 7.3 (2017), pp. 167–181.

[27] Chi-Keung Luk et al. “Pin: building customized program analysis tools with dynamic instrumentation”. In: *Acm sigplan notices*. Vol. 40. 6. ACM. 2005, pp. 190–200.

[28] Nicholas Nethercote and Julian Seward. “Valgrind: a framework for heavyweight dynamic binary instrumentation”. In: *ACM Sigplan notices*. Vol. 42. 6. ACM. 2007, pp. 89–100.

[29] *GitHub - SideChannelMarvels/Deadpool*. https://github.com/SideChannelMarvels/Deadpool. (Accessed on 09/05/2019).

[30] *GitHub - SideChannelMarvels/Daredevil*. https://github.com/SideChannelMarvels/Daredevil. (Accessed on 09/05/2019).

[31] *Conference on Cryptographic Hardware and Embedded Systems 2016 (CHES 2016)*. https://iacr.org/workshops/ches/ches2016/start.php. (Accessed on 09/05/2019). 2016.

[32] Phillipe Teuwen and Charles Hubain. *Differential Fault Analysis on White-box AES Implementations*. https://blog.quarkslab.com/differential-fault-analysis-on-white-box-aes-implementations.html. (Accessed on 09/05/2019). 2016.

[33] *NSC Writeups - YobiWiki*. http://wiki.yobi.be/wiki/NSC_Writeups. (Accessed on 09/05/2019). 2016.

[34] Jia-Rui C Cook. “Engineers Diagnosing Voyager 2 Data System – Update”. In: *Press Release p* 41 (2010).

[35] Keith Bedingfield, Richard D Leach, Margaret B Alexander, et al. "Spacecraft system failures and anomalies attributed to the natural space environment". In: (1996).

[36] Benjamin Danglot et al. "Correctness attraction: a study of stability of software behavior under runtime perturbation". In: *Empirical Software Engineering* 23.4 (2018), pp. 2086–2119.

[37] Udo Voges. "Software diversity". In: *Reliability Engineering & System Safety* 43.2 (1994), pp. 103–110.

[38] Jude Angelo Ambrose, Roshan G Ragel, and Sri Parameswaran. "RIJID: random code injection to mask power analysis based side channel attacks". In: *Proceedings of the 44th annual Design Automation Conference*. ACM. 2007, pp. 489–492.

[39] Subhadeep Banik et al. "Analysis of software countermeasures for whitebox encryption". In: *IACR Transactions on Symmetric Cryptology* (2017), pp. 307–328.

[40] Benoit Baudry and Martin Monperrus. "The multiple facets of software diversity: Recent developments in year 2000 and beyond". In: *ACM Computing Surveys (CSUR)* 48.1 (2015), p. 16.

[41] *LLVM Language Reference Manual — LLVM 10 documentation*. `https://llvm.org/docs/LangRef.html`. (Accessed on 09/05/2019).

[42] William Stallings. *Cryptography and Network Security Principles and Practices, Fourth Edition*. Upper Saddle River, NJ 07458: Prentice Hall, Nov. 2005. ISBN: 0-13-187316-4.

[43] *GitHub - KTH/xPerturb*. `https://https://github.com/KTH/xPerturb`. (Accessed on 09/05/2019).

TRITA -EECS-EX-2020:80