To the University of Wyoming:

The members of the Committee approve the thesis of A. Stone Olguin presented on April 3, 2024.

Dr. James Caldwell, Chairperson

Dr. Mike Borowczak, Co-Chairperson

Dr. Dan Stanescu, Outside Member

Dr. Bryan Shader

,

APPROVED:

Dr. Bryan Shader, Head, Department of Electrical Engineering and Computer Science

Dr. Cameron Wright, Dean, College of Engineering

Olguin, A. Stone, On Evaluating Substitution-Box Constructions for AES Under Side-Channel Analysis, M.S., Department of Electrical Engineering and Computer Science, May, 2024.

The Advanced Encryption Standard (AES) is a widely-used cryptographic algorithm. AES has been constructed to be secure against different cryptanalysis attacks such as linear and differential cryptanalysis. Power-based side-channel attacks, which involve using the power consumption of a device, are one of the biggest vulnerabilities of AES.

An important aspect of the AES algorithm is the Substitution-box (S-box). The S-box is a lookup table that adds security to AES against both linear and differential attacks. There are several different variations on an S-box, each with different lookup values. Every S-box has individual properties that can reflect the cryptographic strength of using that S-box.

This work presents the results of experiments that gather power-based data from AES using different S-boxes. Our hypothesis for these experiments was that certain properties of an S-box factors into the resilience against power-based side-channel attacks. Experiments performed indicate that the S-box properties of nonlinearity, differential probability, and linear probability most significantly affect side-channel resilience in AES. The analysis of this power-based data is used to determine how each property of an S-box affects AES's power-based side-channel resistance. Using each significant property for power-based side-channel resistance, an S-box can be theoretically constructed for use in AES to increase the security of AES against power-based side-channel attacks.

ON EVALUATING SUBSTITUTION-BOX CONSTRUCTIONS FOR AES UNDER SIDE-CHANNEL ANALYSIS

by

A. Stone Olguin, B.S. Mathematics

A thesis submitted to the
Department of Electrical Engineering and Computer Science
and the
University of Wyoming
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE
in
COMPUTER SCIENCE

Laramie, Wyoming
May 2024

Dedicated to my amazing family, whose support helped make this work possible, and my wife Megan, who is the most amazing and supportive person I could've had.

Contents

List of Figures

List of Tables

List of Algorithms

List of Source Codes

Acknowledgments

Throughout this journey of completing this work, I received help and support from many different people and groups. I would like to sincerely thank everyone who has helped contribute to this thesis, whether directly or indirectly.

I first would like to extend my thanks to the commitee members: Dr. James Caldwell, Dr. Mike Borowczak, Dr. Bryan Shader, and outside member Dr. Dan Stanescu. Each one of these members have contributed greatly toward inspiring my pursuit of a graduate degree. Dr. Stanescu and Dr. Shader helped me realize my love for computer science when I was working for my mathematics degree, and Dr. Caldwell and Dr. Borowczak both welcomed me with open arms when I joined the Computer Science department. I would also like to thank Dr. Caldwell for his time in helping me create the best work I could present in this thesis.

I would like to thank IOG and the Kraken cryptocurrency exchange for their financial contributions towards me for this degree. The assistance allowed me to focus on research and not have to worry about financial responsibilities.

I would also like to thank the members of the Cybersecurity EDucation And Research (CEDAR) lab for all of their support. In particular, I would like to formally thank Clay Carper, Jarek Brown, and Andey Robins for assisting me in navigating the world of Computer Science when coming in from a Mathematics background. I have benefited greatly from each of your time and efforts, and I would truly not have made any research progress (or publications) without any of your help.

Next, I would like to thank my friends and family for supporting me throughout this entire process. To my parents and grandparents: thank you so much for supporting me

throughout not just this work, but my undergraduate work as well. Thank you for your patience, support, and faith in my capabilities for this work. To my friends and extended family: thank you for cheering me on and giving me support even when the workload increased.

Finally, I would like to extend as much appreciation as I can to my wife, Megan Olguin. You have been there for me through the toughest times during this work, and your support gave me the strength to finish this thesis. It has been the greatest pleasure to have you by my side during this journey.

A. Stone Olguin

University of Wyoming
May 2024

Chapter 1

Introduction

The technology that makes secure communication possible is cryptography (Chapter 2) [1]. This reliance on cryptography has led to many different cryptographic algorithms being constructed for different uses. One such widely-used cryptographic algorithm is the Advanced Encryption Standard (or AES) (Chapter 3). This protocol is widely-used throughout the world today as a secure means of protecting secret or private data [2].

Despite the diverse applications of AES, it is known to be vulnerable to power-based side-channel attacks (Chapter 4) [3, 4], While this form of attack requires an adversary to have physical access to the device to successfully perform the attack, disregarding power-based side-channel attacks could prove a disastrous assumption for the overall security of AES. As a result of this, AES can be improved by determining a way to be made secure (or at least resistant) against power-based side-channel attacks.

This work aims to demonstrate how work towards creating a more side-channel-resistant AES can be made by changing one of its important features: its Substitution-box (S-box). Analyzing S-boxes and the cryptographic properties of these S-boxes yields a way to predict how each property affects the side-channel resistance of AES (Chapter 5). In this thesis, we experimentally validate the hypothesis that the choice of S-box does have an effect on the side-channel resistance of AES. Using these changes, an ideal S-box can be constructed to improve AES by using the most significant S-box properties that affect side-channel resistance (Chapter 6).

1.1 Prior Work and Inspiration

Previous work had emphasis placed on the refinement of a proof concerning the semantic security of the ElGamal encryption scheme. The original proof [5] was performed in the theorem prover LEAN 3 by Joey Lupo [6] which was developed by Leonardo de Moura from Microsoft Research in 2015. LEAN 4 is a significant upgrade of LEAN 3, but it is not backward-compatible with LEAN 3. Lupo's proof of the semantic security of ElGamal in LEAN 3 depended on a variety of previously-proved theorems from the LEAN 3 mathlib library [7]. To upgrade Lupo's proof to LEAN 4, conversions from the original mathlib to the updated one for LEAN 4 needed to be made. In addition, some of the proofs needed to be modified to ensure that each claim for semantic security was upheld. The code for the LEAN 4 version of the semantic security of ElGamal is given at the following link: `https://github.com/A-Stone-Olguin/cryptolib4`.

Additionally, collaborative research was performed in the research field of power-based side-channel analysis in conjunction with members from the Cybersecurity EDucation And Research (CEDAR) lab. This work involved determining how to correctly use power-based side-channel tools, co-mentoring state educators on the topic, and assisting other researchers in determining the statistical assumptions made in this field [8].

Both of these works demonstrated an engagement with formal correctness proofs for cryptographic protocols and the field of side-channel analysis. It is currently not known how to account for side-channel attacks in formal models [9]. While these two conceptual domains are traditionally opposing in views of formality, a combination of both domains were determined as a worthwhile endeavor for research. In particular, these two concepts can complement each other. In this thesis, we gather side-channel results experimentally and show which formal properties of S-boxes contribute to side-channel resistance of AES.

This document was prepared using LaTeX on the website Overleaf [10], and the figures used in Chapters 2-4 were constructed using the program Draw.io [11].

Chapter 2

Cryptography

Cryptography is the art and science of secure communication of information to prevent unintended or unauthorized adversaries from determining the shared information [1]. The original form of the messages are called the plaintext, and the resulting secret message is the ciphertext. The processes of transforming plaintext into ciphertext are done through cryptographic protocols (or cryptosystems), which can involve algorithms ran on a computer [12], physical objects such as belts and sticks [13], or simply changing the letters of the alphabet [14]. The study of the recovery of the plaintext from the ciphertext is called cryptanalysis [15]. In particular, the process of converting plaintext into ciphertext is called encryption, while the reverse process of returning ciphertext into the original plaintext is called decryption.

The process of communication involving cryptography usually involves the sender of a message to encrypt the plaintext and communicating the ciphertext. This process involves using a secret key that will ensure that only the intended recipient can decrypt the ciphertext to recover the original plaintext. The usage of the secret key prevents unintended recipients of the ciphertext, also known as attackers or adversaries, from decrypting the ciphertext. A brute-force attack is when an adversary attempts to guess the secret key by guessing all possible values of the secret key. Using this definition, a cryptographic protocol has a break when a weakness in the protocol can be exploited to recover the secret key in less complexity

than an average brute-force attack [16]. Communication using cryptography is shown[1] in Figure 2.1, derived from the figure by Sharma and Gupta [17].

Figure 2.1: Cryptographic Process

Cryptography has been used as early as 100 BCE [15]. The Caesar Cipher was attributed to Julius Caesar in 100 BCE as using the alphabet but shifting each letter a fixed number. This cipher is simple to grasp, and even children can use cipher wheels which can allow easy encryption and decryption of messages. A cipher wheel is shown in Figure 2.2, and this shows how for encryption, A is translated to T, and for decryption T is translated to A. The shift of the letters would be the secret key for this cryptographic protocol. However, this ease of use allows for unintended recipients of the ciphertext to retrieve the original plaintext from the ciphertext. One such method for an attacker is to perform frequency analysis [18] on the number of letters used in the text compared to the number of words in the original language that contain those letters. For example, in the English alphabet the most common letter used is 'E', while 'Z' is the least frequently used letter [19]. Additionally, since each letter is shifted a constant amount in the Caesar Cipher, if an attacker can determine which letter is associated with which cipher letter, the rest of the cipher can be broken. As a generalization of a Caesar Cipher, Substitution Ciphers [15] involve permuting the letters of the alphabet to create the new cipher alphabet. While this does make it harder for an attacker to determine the encryption method used for the plaintext by requiring a knowledge of each mapping of the permutation, all Substitution Ciphers can still be attacked by using frequency analysis [15]. In fact, only an average of twenty-eight letters of the cipher text are needed to break a given substitution cipher [20].

[1]The Encryption process used in this figure is AES-128 (Chapter 3).

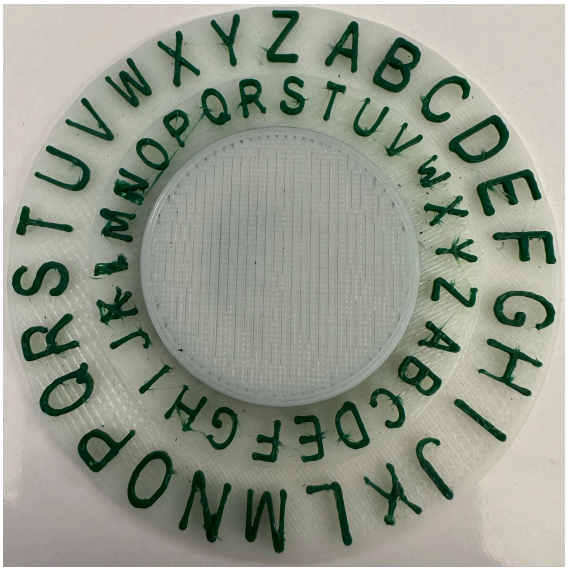


Figure 2.2: Cipher wheel.

2.1 Types of Cryptography

The Caesar Cipher and its generalization, the Substitution Cipher are both known as symmetric cryptographic protocols [21]. This means that the secret key used by both the sender and receiver are the same. Common symmetric cryptographic protocols are the Caesar Cipher and general Substitution Ciphers mentioned above, along with the Scytale Cipher [13], and even modern cryptographic protocols such as the Advanced Encryption Standard [12] (Chapter 3) and Skipjack [22]. Symmetric encryption can be visualized as shown in Figure 2.1 where the secret keys are the same.

Asymmetric cryptographic protocols (also known as public-key cryptosystems) [21] involve the recipient and the sender each having different secret keys. In contrast to symmetric cryptosystems, both the recipient and the sender also have publicly-available public keys derived from the secret keys. A popular example of an asymmetric cryptographic protocol is RSA [23], an algorithm that involves computation using two very large prime numbers. Asymmetric encryption can add authenticity verification of the sender by encrypting the sender's secret key and the recipient's public key and adding them to the message. Since only the recipient can decrypt the message (using their secret key), they gain access to the sender's public key which can be used to verify that they were the actual sender.

This can be represented symbolically as follows: m^K is the message m encrypted by key K and m^{-K} is the decryption of message m using key K. For each participant (say A) in a protocol, we assume they have both a secret key (S_A) and a *published*[2] public key (P_A). The keys S_K and P_K are inverses of one another, *i.e.*,

$$(m^{P_K})^{-S_K} = m = (m^{S_K})^{-P_K}.$$

The left-hand side of the above equation may seem initially unintuitive; if a sender wanted to send a message to K, P_K could be used to encrypt the message and then only agent K has the secret key to decrypt it. The right-hand side is perhaps clearer, agent K encrypts message m using the secret key S_K and that message can be decrypted by anyone using the published public key. This can be useful if K wants to *sign* a message. Suppose the sender is labeled A (for Alice) and the receiver is labeled B (for Bob), then asymmetric encryption and decryption are defined by the following functions:

$$\begin{aligned} encrypt_A\, m\, B &: (m^{S_A})^{P_B} = c \\ decrypt_B\, c\, A &: (c^{-S_B})^{-P_A} = m \end{aligned} \tag{2.1}$$

Note that each participant A (and B) has its own encryption and decryption algorithms which build in their secret keys denoted as $encrypt_A$ and $decrypt_A$. Figure 2.3 shows how both public and secret keys from both parties are used to encrypt and decrypt plaintext.

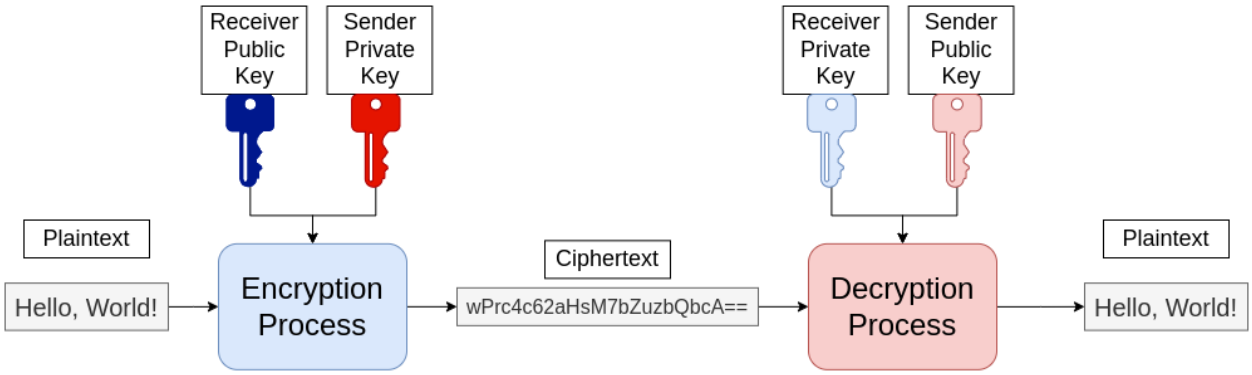


Figure 2.3: Asymmetric Encryption.

[2]There are a number of publicly-accessible key registries such as Microsoft Azure Key Vault, GnuPG, Seahorse, Oracle Cloud Infrastructure Vault, Google Cloud Key Management, and Hashicorp Vault, among others.

Another way to classify a cryptographic protocol is by the way the plaintext is encrypted. A popular method for encryption of plaintexts of varying length are by splitting the plaintext into fixed-length blocks. Each plaintext block is encrypted into resulting ciphertext blocks resulting in the full ciphertext when each block is appended together. Symmetric cryptographic protocols that use this method are called block ciphers [24]. Examples of such block ciphers are DES [25], AES [12], and Twofish [26]. Alternatively, stream ciphers are another form of symmetric cryptographic protocol. These protocols involve encrypting each character of the plaintext with a pseudorandom stream (a number that looks random, but is generated deterministically) through the XOR (Boolean exclusive or) operation [27]. Notable examples of stream ciphers are Scream [28], SNOW [29], and Turing [30].

2.2 Importance of Cryptography

Cryptographic protocols were originally intended for military and diplomatic uses for communication without fear of the confidential messages falling into adversarial hands [15, 31]. However, the rise of the Internet as a public communication channel [32] has increased the need for cryptographic communication to ensure secure transmission of personal or private data.

One popular use of cryptography is in electronic commerce. When purchasing items on Internet stores such as Amazon, credit card information is sent over the Internet, and this information should be encrypted to avoid potential attackers from accessing the credit card information [33]. As a generalization of this, Figure 2.4 demonstrates how credit card information is sent to a online webstore to make a purchase while an adversary grabs the encrypted data. By encrypting the credit card information sent over the Internet for the purchase, even if an attacker could grab the encrypted data, it would be hard to gather the original unencrypted data by nature of the difficulty of breaking a cryptographic protocol (Section 2.3).

The authenticity of data is another important aspect of cryptography. Cryptography can ensure important authenticity of the data, authenticity of the sender, and authenticity of

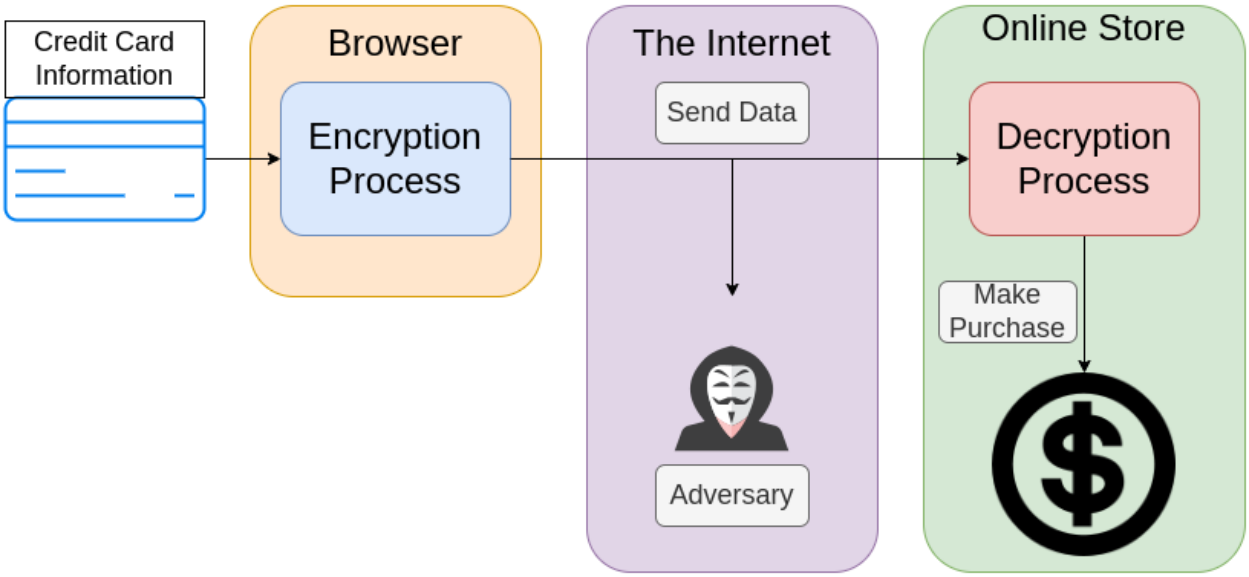


Figure 2.4: Credit card information transmitted online.

the receiver [34]. By ensuring the authenticity of the data, a sender and receiver of encrypted data can ensure that the data they received was correctly sent. If the sender and receiver are both authenticated through cryptographic means, then each party is sure that the data comes from and is sent to the intended party. RSA demonstrates the authenticity of the sender and receiver of the cryptographic data by using both the public and secret keys of each user by nature of being an asymmetric encryption algorithm as shown in Equation 2.1.

2.3 Difficulty of Breaking a Cryptographic Protocol

Cryptographic protocols are assumed to be difficult to break; however, oftentimes this comes at the cost of ease of computing the encrypted data. As mentioned earlier in the chapter, the Caesar Cipher is simple to encrypt, but it is very easy to break by only requiring a knowledge of the shift. The definition of hardness of a cryptosystem context depends on the difficulty of the underlying problem that encrypts the data. The Caesar cipher relies upon determining the shift of the letters which is a constant value (1 through 26). As a result, the secret key can be 26 possible values. The general Substitution Cipher was mentioned to be more difficult than the Caesar Cipher, and this is reflected by the number of possible

keys at $26 \times 25 \times \cdots \times 2 \times 1 = 26!$. The attacks on these ciphers are brute-force attacks (see introduction to chapter).

To determine the difference between easy and hard problems to solve, we introduce P and NP. P is the set of all problems that can be solved in a polynomial amount of time with regards to the input size [35]. NP is the set of all problems that, given a potential solution, that solution can be checked in polynomial time with regards to the input size [35]. Using these definitions, easy problems are considered to be in P, whereas hard problems are considered to be in NP. In some ways, it may be surprising that problems in P are considered "easy" because for example some problems with complexity n^{1000} is in P, but it turns out that practical problems typically have complexity n^3 or less. Even the general Substitution Cipher, which has 26! different keys is in P because regardless of the input size of the plaintext, 26! is a constant.

Clearly P $\subseteq$ NP, this is the case because a correct solution found in polynomial time takes no time at all to check. If NP $\subseteq$ P, then we would know P = NP, and if NP $\not\subseteq$ P, then P $\neq$ NP. This problem of P $\stackrel{?}{=}$ NP has turned out to be one of the most difficult problems in mathematics today. In fact, it is a millennium prize problem [36]. The Clay Mathematics Institute will reward anyone that can prove this problem with one million dollars. A visual representation of P $\stackrel{?}{=}$ NP is shown in Figure 2.5. Most cryptologists (and complexity theorists) assume that P $\neq$ NP, and as a result, some problems must be in NP that are not in P. These definitions are surprisingly robust. There is the possibility (yet to be proved) that there are problems solvable in polynomial time on a quantum computer that take exponential time on a traditional model of computation *e.g.* a Turing Machine or a Register-Transfer Machine.

Returning to the discussion about cryptography, the assumption that P $\neq$ NP is the cornerstone of modern cryptography. By using these definitions, the security of a cryptographic protocol is proportional to the difficulty of the problem it takes to brute-force guess the secret key. With these definitions of P and NP problems, we can classify the difficulty of protocol attacks. In particular, problems in P are "easy" protocols to attack, whereas problems in NP are "hard" protocols to attack. Since both the Caesar Cipher and general

Substitution Cipher are not dependent on the length of the plaintext or the secret key, both of these problems are in P. By contrast, RSA is dependent on the size of the modulus used for the key to be factored, so it is in NP because the factoring problem is in NP [35]. By our definitions of P and NP, it follows that successfully attacking the Caesar Cipher and general Substitution Cipher are "easy" whereas attacking RSA is "hard", as long as P $\neq$ NP.

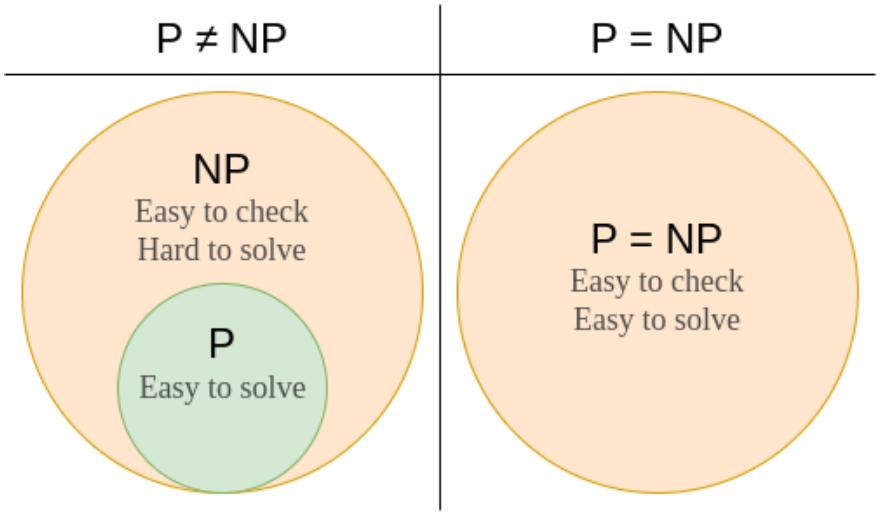


Figure 2.5: Diagram showcasing either case of P = NP or P $\neq$ NP.

The difficulty of breaking a cryptographic protocol is also dependent on the assumptions made when using the cryptosystem. As P and NP are defined, they characterize the complexity of problems running on traditional models of computation. Quantum computers, however, do not run computations the same way traditional computers do. As a result, using a Quantum computer, RSA can be broken more easily because factoring can be performed in polynomial time on the size of the modulus by using Shor's algorithm [37]. However, Shor's algorithm is not proven to have quantum supremacy, so it remains to show whether factoring has a polynomial time solution on traditional computational models.

2.4 Limitations of Cryptography

As mentioned in Section 2.3, the security of a cryptographic protocol is proportional to the difficulty of the problem to brute-force a correct guess for the secret key. Despite this, the

security of a cryptographic algorithm is not decided solely by the difficulty of the brute-force attack. In fact, some problems become easier to solve as computation power increases. The Data Encryption Standard (DES) was once secure enough to be used by the government for all unclassified data [38], but this protocol has since been seen as insecure because of the increase in computational power (following Moore's Law) which allows for the key to be extracted quick enough [39, 40]. There are protocols that are known to be insecure against attacks using a quantum computer. There are quantum-safe cryptographic protocols[3], *i.e.* protocols that are resistant to attacks used by a quantum computer.

Most cryptographic protocols rely on the assumption that P $\neq$ NP. This assumption allows cryptographers to assume, for example, that there are one-way functions. One-way functions are functions f that are easy to compute with an input x, such that if $f(x) = z$, $f^{-1}(z)$ is hard to compute [41]. Common examples of these are cryptographic hash functions such as SHA-256 [42]. By assuming that SHA-256 is a one-way function, the cryptographic protocol assumes that P $\neq$ NP. If it turns out that P = NP, then the strength of SHA-256 would no longer be valid because the hardness of computing the inverse of the hash function would be computationally easy.

Resources are another possible limitation in cryptography. Some devices such as micro-controllers have fewer computational resources than normal computers. As a result, certain cryptographic algorithms have to make concessions on speed or power to be used on different devices, such as AES using tiny-AES-c [43]. An example of such computationally-limited devices would include microchips installed on credit cards. Adding additional security could increase the time it takes to encrypt and decrypt, or it could come at the cost of additional power usage [44]. Figure 2.6, inspired by Mark Ciampa [44], shows the resource tradeoffs that must be made to ensure higher security in terms of time and energy of the cryptosystem. A concrete example of this are two modes of encryption for AES: ECB and CBC (see Chapter 3 for more details). Almuhammadi and Al-Hejri [45] demonstrate that ECB can encrypt and decrypt faster with an encryption throughput of 85.75 KB/ms as opposed to CBC which has an encryption throughput of 73.92 KB/ms. While ECB can encrypt faster than CBC, CBC

[3] https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions

is found to be more secure because the same block of plaintext is not encrypted the same every time in CBC as opposed to ECB [46].

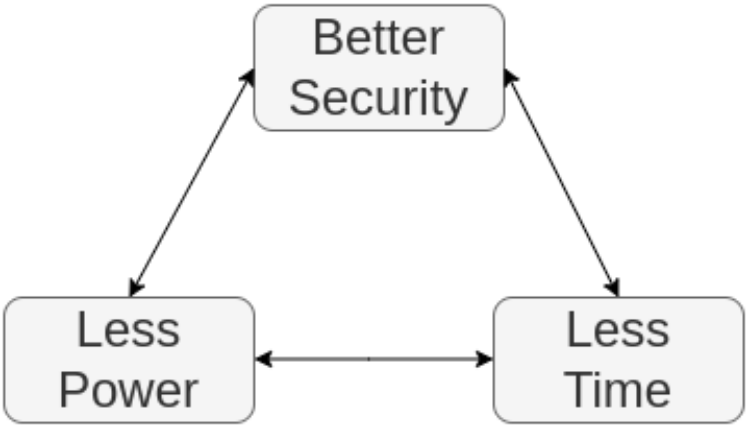


Figure 2.6: Tradeoffs of cryptographic protocols.

Additionally, there are other methods of attack that can make certain cryptographic protocols less secure. Linear cryptanalysis and differential cryptanalysis are two popular methods of attacking cryptographic algorithms [47]. As an example, if a cryptographic protocol is proven to be resistant to linear cryptanalysis, then the protocol is seen as secure against linear cryptanalysis. This doesn't mean the protocol isn't seen as secure as a whole, but it is secure against a certain attack. Conversely, if a protocol is not secure against a certain attack, that does not mean the protocol as a whole is not secure. As an example, AES is constructed to be secure under linear and differential cryptanalysis [48], but it is vulnerable to having its secret key extracted from power-based side-channel analysis (Chapter 4). Despite this, AES is commonly used in many applications (see Chapter 3) and is even stated to be secure by the National Security Agency (NSA) [49]. This is because certain attacks also make assumptions that in turn, limit their effect on the security of a cryptographic protocol.

In an effort to balance the assumptions that can limit the application or security of a cryptosystem, indistinguishability obfuscation (iO) [50] is a process that limits the assumptions made by a process. iO aims to ensure that if two protocols compute the same result, then their obfuscation should be indistinguishable. In particular, Barak et al. [51] demonstrate that constructing a program that satisfies iO can be performed if P = NP even though cryptographic protocols can not be made [52]. However, if P $\neq$ NP, then iO would be much

harder to produce, as demonstrated by Garg et al. [53].

2.5 Provability of Cryptographic Protocols

Provability of a cryptographic protocol can relate to whether the protocol runs as intended (provable correctness) or if it is secure by only being able to have the secret key extracted by brute-force attack (provable security). Certain cryptographic protocols, such as one-time pad [54], have been proved secure by being shown to be unbreakable in a very strong sense. Unfortunately, the secret key to a one-time pad must be as long as the plaintext itself, and this is an impractical constraint that leads to it not being useful in almost all applications. However, provable security as a blanket statement is difficult to achieve by nature of how security is defined for a cryptographic protocol. As mentioned in Section 2.4, protocols can be secure against certain attacks but vulnerable to others. This in turn, makes it difficult to prove the security of a cryptosystem. For example, unless P $\neq$ NP (a mathematical theorem yet to proved or disproved), AES cannot be proved secure [55]. This is why P $\neq$ NP is a standard assumption made by most cryptographic protocols.

Historically, there has been significant tension between the more theoretical community involved in correctness proofs and the more practical community of engineers who try to break protocols as they are running on physical devices. Koblitz and Menenzes [9] argue that proofs of security in cryptographic protocols have "limited relevance". Koblitz and Menenzes state that the proofs tend to not have the true formal rigor of mathematical proofs, and entire cryptographic protocols cannot be formalized. Attacks such as side-channel attacks (Chapter 4) are not accounted for by "the usual security definitions" [9, pp. 8] because the properties of the device running a protocol is difficult to generally formalize. They also deny the usefulness of certain security proofs (such as semantic security and its equivalent idea of indistinguishability [56]).

In a scathing response to Koblitz and Menenzes, Goldreich [57] argues that denying the value of formal mathematical models to reason about cryptographic protocols is to deny the value of science. In fact, Goldreich's paper is titled "On Post-Modern Cryptography",

referring to post-modernism which is considered by many to be unscientific. Goldreich states that Koblitz and Menenzes make a "conceptual mistake" [57, pp. 10] by confusing modern cryptography with complexity theory. Goldreich claims that arguments based on polynomial-time computation are "inessential" [57, pp. 10] and can be replaced by arguments about resource bounds, including computational time and space. Goldreich argues that Koblitz and Menenzes's argument against mathematical approaches to cryptography is based on objections about "contrived" counterexamples (versus "natural" ones). He points out that even if a "contrived" counterexample leads to failure of a cryptographic protocol, then indeed it would be foolish to depend on the security of that protocol.

The argument about the provability of a cryptographic protocol (in terms of security or correctness) mimics the arguments of whether programs can be proven correct. Fetzer argued against proofs of formal correctness for programs [58]. Similarly, but with a different argument, Lipton, DeMillo, and Perlis [59] also argued against proofs of program correctness. Koblitz and Menenzes [9] mimic these arguments against formal modeling and proofs of correctness for cryptographic protocols. Goldreich's response to Koblitz and Menenzes is not terribly different from responses to arguments against formal program correctness.

Chapter 3

Advanced Encryption Standard (AES)

In order to develop a secure encryption algorithm to enter the twenty-first century, The National Institute of Science and Technology (NIST) aimed to develop an Advanced Encryption Standard (AES) with contributions from the cryptographic community starting in 1997 [49]. In particular, the intentions of AES were to be used by the U.S. Government. The intention of finding a new encryption standard arose after it became clear that the Data Encryption Standard (DES) [25] was vulnerable to cryptanalysis attacks [40]. The Rijndael cryptographic algorithm was created by Joan Daemen and Vincent Rijmen [60], and it was selected as the winner for the proposal for AES in 2000 [12]. As a result, AES is also known as the Rijndael encryption scheme. Since its acceptance, AES has been approved by the Federal Information Processing Standards (FIPS) [12].

Additionally, the National Security Agency (NSA) determined that AES was secure enough for non-classified data [49]. Rijndael's implementation used key lengths of 128, 192, or 256 bits which were all longer than DES's 56-bit key length. Because of this, AES was determined to be secure for top-secret government information by the NSA for key lengths of 192 bits and 256 bits [49].

AES is a symmetric block cipher of 128 bits, but can have varying key lengths of 128, 192, or 256 bits. To accomodate different key lengths, different number of rounds of encryption are used for each key length; namely, 10, 12, and 14 rounds for 128, 192, and 256-bit key lengths; respectively. Shorthands for AES with the different key lengths are AES-128, AES-

192, and AES-256 to differentiate what key length is being used for the AES implementation. An overview of the AES algorithm along with what occurs in each round of encryption will be elaborated in Section 3.2. Since its development, AES has become the most widely-used symmetric encryption algorithm [2]. It is used in diverse applications including Wi-Fi networks [61], Google Cloud [62], Facebook Messenger [63], and many others.

The security of AES has varied as cryptanalysis techniques have evolved over time. Side-Channel Analysis (Chapter 4) has used AES as a common attack target [64–67] and has been able to recover the secret key used by AES. These attacks are based on a number of assumptions that may not hold in many scenarios (Section 2.3). Related-key attacks [68] have also been shown to exploit the key schedule to have a recovery of the secret key [69]. However, despite attacks being made against AES, it has been shown to be quantum resistant [70]; however this is only the case for AES-256.

3.1 The Substitution-Box (S-box)

One fundamental aspect of AES is the substitution-box (S-box) which is used to obscure the secret key's relationship with the ciphertext. S-boxes aim to be resistant to linear and differential cryptanalysis [40, 47, 71], and the measure of their strength against these attacks can be demonstrated by the nonlinearity [72] and their differential uniformity [73]; respectively. The Rijndael S-box was constructed to be resistant against linear and differential cryptanalysis attacks, so AES is resistant to each of these common attack methods [48]. Rijndael's S-box maps an 8-bit input to an 8-bit output, both represented as polynomials over the Galois Field[1] of size 2 (GF(2)). Using the reducing polynomial of $x^8 + x^4 + x^3 + x + 1$, each element in the size 256 field ($\mathrm{GF}(2^8)$) is mapped to another element in $\mathrm{GF}(2^8)$, with the identity zero mapped to itself.

$$s' = s \oplus (s \lll 1) \oplus (s \lll 2) \oplus (s \lll 3) \oplus (s \lll 4) \oplus 63_{16} \tag{3.1a}$$

$$s' = (s \lll 1) \oplus (s \lll 3) \oplus (s \lll 6) \oplus 5_{16} \tag{3.1b}$$

[1] A Galois Field is a finite field [74].

Since the S-box is a lookup table, an inverse would be needed in order to decrypt data. To ensure that the S-box will have an inverse for decryption, an affine transformation is applied to each element of the S-box. An affine transformation is any function that takes input vectors and transforms it using geometric contraction, expansion, dilation, reflection, rotation, shear, translation, or stretching along with any combinations of these transformations [75]. To express this using vector spaces and other transformations, the input and output are expressed as bitvectors, over the field GF(2). For example, the bitstring 10001111 is expressed as $[1,0,0,0,1,1,1,1]^T$, where T means the vector is transposed. As a result, an affine transformation is performed on these vectors over GF(2) by using rotations (bitwise shifts) and addition (XOR). The affine transformation applied to element s of the S-box yields an output s' as defined in Equation 3.1a, where $\oplus$ is the XOR operator, $\lll$ is the left circular bitwise shift, and X_{16} is the number X in hexadecimal, a base-16 number. The inverse affine transformation to construct an inverse S-box is shown in Equation 3.1b. Alternatively, the matrix version of the affine transformation in Equation 3.2 and the inverse S-box affine transformation is shown in Equation 3.3.

$$\begin{bmatrix} s'_0 \\ s'_1 \\ s'_2 \\ s'_3 \\ s'_4 \\ s'_5 \\ s'_6 \\ s'_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \tag{3.2}$$

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} s'_0 \\ s'_1 \\ s'_2 \\ s'_3 \\ s'_4 \\ s'_5 \\ s'_6 \\ s'_7 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} \tag{3.3}$$

The original S-box proposed by Rijndael is given in Table 3.1. From an affine transformation shown in Equation 3.1b, an S-box's inverse is guaranteed to exist. The inverse of the original Rijndael S-box is given in Table 3.2. Notice that element (0,0) in the S-box (Table 3.1, bolded) has a value of 0x63, and the element (6,3) in the inverse S-box (Table 3.2, bolded) has a value of 0x00, where (X,Y) is (row,column) of the table. Also, the entry

(0,0) in Table 3.2 is 0x52 while the entry (5,2) in Table 3.1 is 0x00. For every pair (X,Y) $\in \{0, 1, \cdots, f\} \times \{0, 1, \cdots, f\}$, the entry for (X,Y), say 0xWZ, in one of the tables encodes the row and column in the other table at entry (W,Z) and vice-versa.

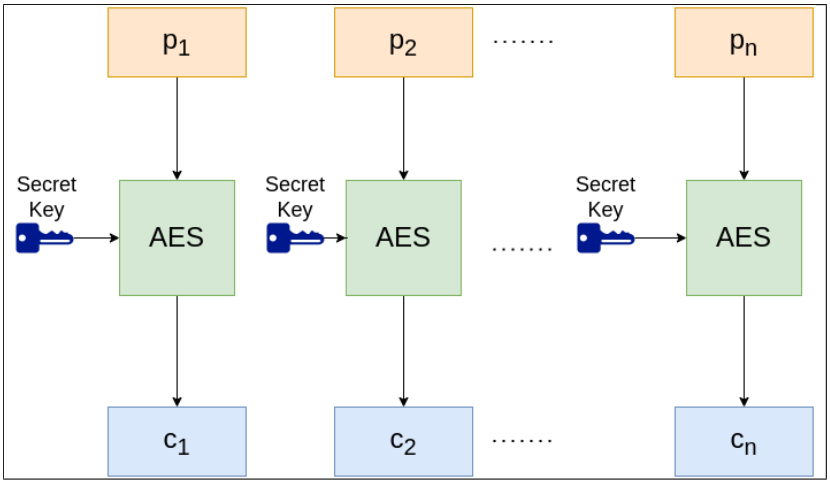
	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	**0x63**	0x7c	0x77	0x7b	0xf2	0x6b	0x6f	0xc5	0x30	0x01	0x67	0x2b	0xfe	0xd7	0xab	0x76
1	0xca	0x82	0xc9	0x7d	0xfa	0x59	0x47	0xf0	0xad	0xd4	0xa2	0xaf	0x9c	0xa4	0x72	0xc0
2	0xb7	0xfd	0x93	0x26	0x36	0x3f	0xf7	0xcc	0x34	0xa5	0xe5	0xf1	0x71	0xd8	0x31	0x15
3	0x04	0xc7	0x23	0xc3	0x18	0x96	0x05	0x9a	0x07	0x12	0x80	0xe2	0xeb	0x27	0xb2	0x75
4	0x09	0x83	0x2c	0x1a	0x1b	0x6e	0x5a	0xa0	0x52	0x3b	0xd6	0xb3	0x29	0xe3	0x2f	0x84
5	0x53	0xd1	0x00	0xed	0x20	0xfc	0xb1	0x5b	0x6a	0xcb	0xbe	0x39	0x4a	0x4c	0x58	0xcf
6	0xd0	0xef	0xaa	0xfb	0x43	0x4d	0x33	0x85	0x45	0xf9	0x02	0x7f	0x50	0x3c	0x9f	0xa8
7	0x51	0xa3	0x40	0x8f	0x92	0x9d	0x38	0xf5	0xbc	0xb6	0xda	0x21	0x10	0xff	0xf3	0xd2
8	0xcd	0x0c	0x13	0xec	0x5f	0x97	0x44	0x17	0xc4	0xa7	0x7e	0x3d	0x64	0x5d	0x19	0x73
9	0x60	0x81	0x4f	0xdc	0x22	0x2a	0x90	0x88	0x46	0xee	0xb8	0x14	0xde	0x5e	0x0b	0xdb
a	0xe0	0x32	0x3a	0x0a	0x49	0x06	0x24	0x5c	0xc2	0xd3	0xac	0x62	0x91	0x95	0xe4	0x79
b	0xe7	0xc8	0x37	0x6d	0x8d	0xd5	0x4e	0xa9	0x6c	0x56	0xf4	0xea	0x65	0x7a	0xae	0x08
c	0xba	0x78	0x25	0x2e	0x1c	0xa6	0xb4	0xc6	0xe8	0xdd	0x74	0x1f	0x4b	0xbd	0x8b	0x8a
d	0x70	0x3e	0xb5	0x66	0x48	0x03	0xf6	0x0e	0x61	0x35	0x57	0xb9	0x86	0xc1	0x1d	0x9e
e	0xe1	0xf8	0x98	0x11	0x69	0xd9	0x8e	0x94	0x9b	0x1e	0x87	0xe9	0xce	0x55	0x28	0xdf
f	0x8c	0xa1	0x89	0x0d	0xbf	0xe6	0x42	0x68	0x41	0x99	0x2d	0x0f	0xb0	0x54	0xbb	0x16

Table 3.1: Rijndael S-box in hexadecimal. The (0,0) element is bolded.

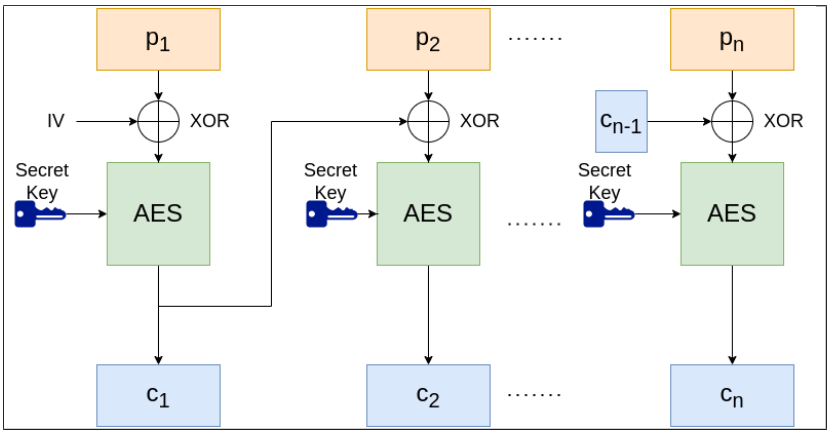
	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	0x52	0x09	0x6a	0xd5	0x30	0x36	0xa5	0x38	0xbf	0x40	0xa3	0x9e	0x81	0xf3	0xd7	0xfb
1	0x7c	0xe3	0x39	0x82	0x9b	0x2f	0xff	0x87	0x34	0x8e	0x43	0x44	0xc4	0xde	0xe9	0xcb
2	0x54	0x7b	0x94	0x32	0xa6	0xc2	0x23	0x3d	0xee	0x4c	0x95	0x0b	0x42	0xfa	0xc3	0x4e
3	0x08	0x2e	0xa1	0x66	0x28	0xd9	0x24	0xb2	0x76	0x5b	0xa2	0x49	0x6d	0x8b	0xd1	0x25
4	0x72	0xf8	0xf6	0x64	0x86	0x68	0x98	0x16	0xd4	0xa4	0x5c	0xcc	0x5d	0x65	0xb6	0x92
5	0x6c	0x70	0x48	0x50	0xfd	0xed	0xb9	0xda	0x5e	0x15	0x46	0x57	0xa7	0x8d	0x9d	0x84
6	0x90	0xd8	0xab	**0x00**	0x8c	0xbc	0xd3	0x0a	0xf7	0xe4	0x58	0x05	0xb8	0xb3	0x45	0x06
7	0xd0	0x2c	0x1e	0x8f	0xca	0x3f	0x0f	0x02	0xc1	0xaf	0xbd	0x03	0x01	0x13	0x8a	0x6b
8	0x3a	0x91	0x11	0x41	0x4f	0x67	0xdc	0xea	0x97	0xf2	0xcf	0xce	0xf0	0xb4	0xe6	0x73
9	0x96	0xac	0x74	0x22	0xe7	0xad	0x35	0x85	0xe2	0xf9	0x37	0xe8	0x1c	0x75	0xdf	0x6e
a	0x47	0xf1	0x1a	0x71	0x1d	0x29	0xc5	0x89	0x6f	0xb7	0x62	0x0e	0xaa	0x18	0xbe	0x1b
b	0xfc	0x56	0x3e	0x4b	0xc6	0xd2	0x79	0x20	0x9a	0xdb	0xc0	0xfe	0x78	0xcd	0x5a	0xf4
c	0x1f	0xdd	0xa8	0x33	0x88	0x07	0xc7	0x31	0xb1	0x12	0x10	0x59	0x27	0x80	0xec	0x5f
d	0x60	0x51	0x7f	0xa9	0x19	0xb5	0x4a	0x0d	0x2d	0xe5	0x7a	0x9f	0x93	0xc9	0x9c	0xef
e	0xa0	0xe0	0x3b	0x4d	0xae	0x2a	0xf5	0xb0	0xc8	0xeb	0xbb	0x3c	0x83	0x53	0x99	0x61
f	0x17	0x2b	0x04	0x7e	0xba	0x77	0xd6	0x26	0xe1	0x69	0x14	0x63	0x55	0x21	0x0c	0x7d

Table 3.2: Inverse Rijndael S-box in hexadecimal. The (6,3) element is bolded.

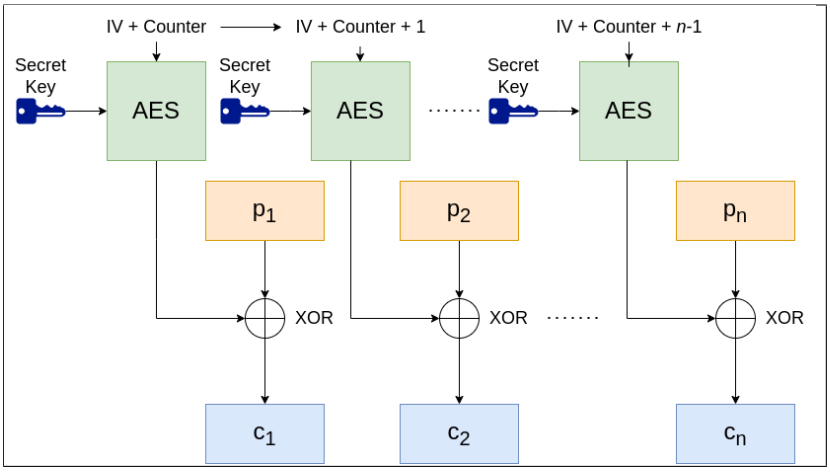
There are three main modes used for AES: Electronic Code Block (ECB); Cipher Block Chaining (CBC); and Counter (CTR) [46]. ECB is the simplest of the modes by simply running each block of the plaintext through the encryption algorithm; and as a result, it is the fastest of the encryption modes. However, this is vulnerable to attacks since the same block of plaintext will always produce the same ciphertext, regardless of where it occurs in the



(a) AES in ECB Mode.



(b) AES in CBC Mode.



(c) AES in CTR Mode.

Figure 3.1: Visualization of AES run in different encryption modes.

plaintext. CBC involves using an Initialization Vector (IV) that is a random, fixed number to "chain" the blocks so that there is a relationship between the entire plaintext. CTR also adds a relationship between the entire plaintext through each block by incrementing a counter which acts as the IV for each block of the plaintext. Figure 3.1 visualizes the encryption modes of ECB (Figure 3.1a), CBC (Figure 3.1b), and CTR (Figure 3.1c) where the plaintext $p = p_1 p_2 \dots p_n$ is broken into n blocks to produce the blocked ciphertext $c = c_1 c_2 \dots c_n$, and IV is the initialization vector. These visualizations were inspired from the figures designed by Shawn Wang [76].

3.2 AES Algorithm

AES involves using 10, 12, or 14 rounds for AES-128, AES-192, and AES-256 respectively. For this section, the number of rounds will be denoted as N to show the generalization of AES, regardless of the key length used for the algorithm. A visual representation of AES is shown in Figure 3.2, as inspired by Arrag et al [77].

At the start of the first round, the keys used for each round are derived from a key schedule. This expansion of keys allows for a different key for each round, and this was made to ensure it met FIPS standards [48]. Afterwards, the first `AddRoundKey` function is called. Rounds 1 through (N-1) use the functions `SubBytes`, `ShiftRows`, `MixColumns`, and `AddRoundKey`, in that order. On the final round, the `MixColumns` function call is not used.

Each step (or function) in AES operates on an intermediate result, the state. Let p be the input plaintext block, and let $b =$ the block length. Then, the definition of the state, S, is a $4 \times \frac{b}{32}$ matrix where each element in row i and column j ($s_{i,j}$) is

$$s_{i,j} = p_{i+4j}, \qquad 0 \leq i < 4, \quad 0 \leq j < b,$$

as defined by Daemen and Rijmen [48, pp. 32]. For the following sections, the state will be visualized as a 4×4 matrix (AES-128 state), as demonstrated in Equation 3.4. Additionally,

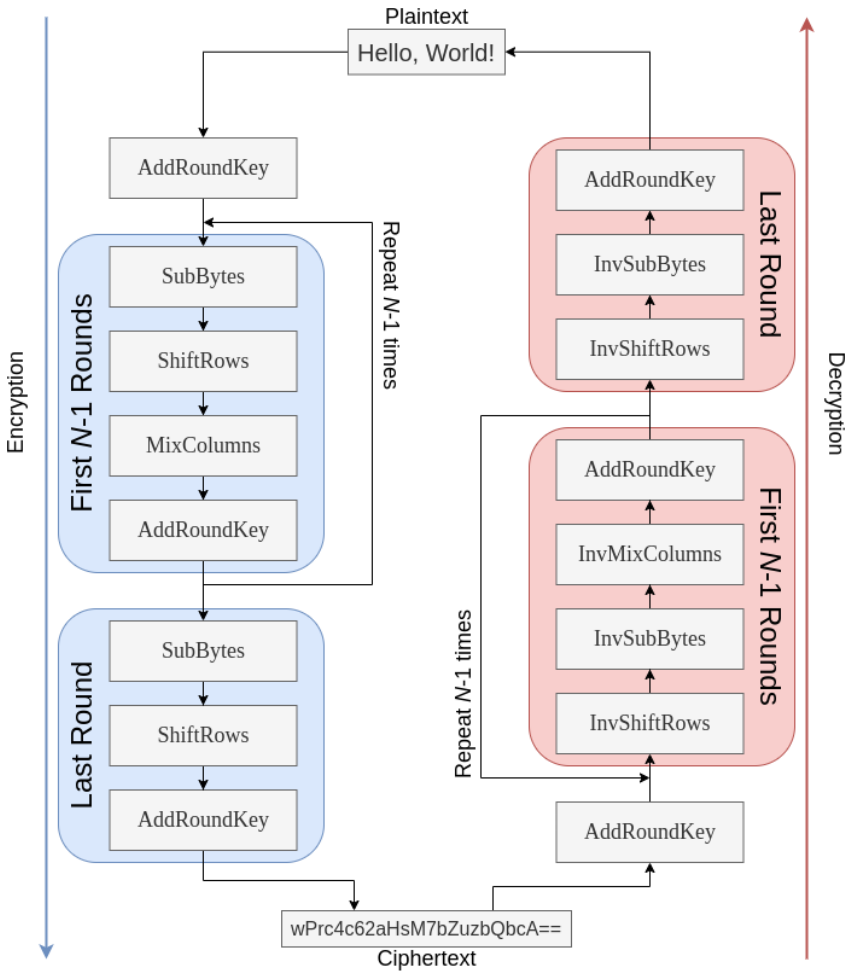


Figure 3.2: AES algorithm.

each graphic is inspired from the graphics presented by Arrag et al [77].

$$S = \begin{bmatrix} p_0 & p_4 & p_8 & p_{12} \\ p_1 & p_5 & p_9 & p_{13} \\ p_2 & p_6 & p_{10} & p_{14} \\ p_3 & p_7 & p_{11} & p_{15} \end{bmatrix} \tag{3.4}$$

3.2.1 AddRoundKey

This step of the algorithm involves using the round key, which is created from the main key using the AES key schedule. This round key has a byte XOR'ed with each byte in the state. This step aims to add differing keys for each round from the original key, and a visualization

of this step is shown in Figure 3.3.

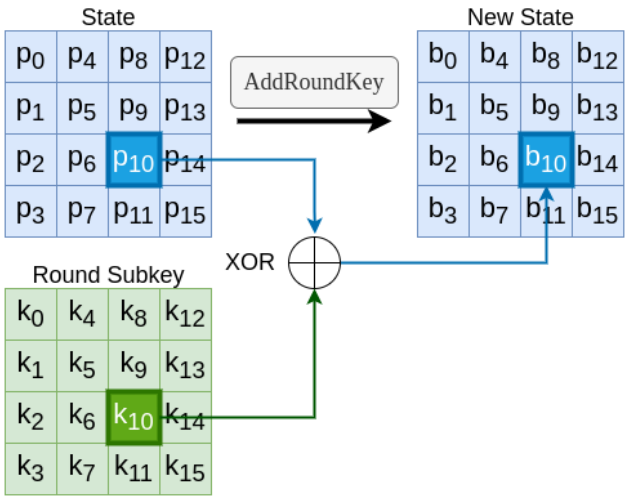


Figure 3.3: `AddRoundKey` step of AES.

3.2.2 `SubBytes`

This step involves using substitution of the bytes for each byte in the state. In particular, this substitution is performed using the S-Box (Table 3.1) to add nonlinearity to the algorithm, as shown in Figure 3.4. The addition of nonlinearity in this step adds to the linear and differential cryptanalysis security of the algorithm. The inverse of this step is performed during decryption, which involves using the inverse S-box as shown in Table 3.2.

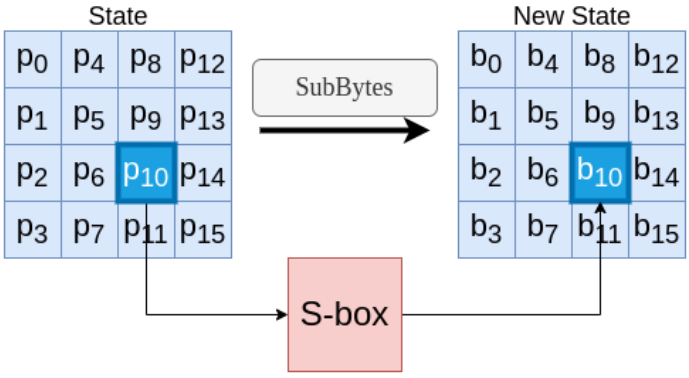


Figure 3.4: `SubBytes` step of AES.

3.2.3 `ShiftRows`

To ensure that each column in the state is not four independent block ciphers (as a result of always being four rows, regardless of key size), the `ShiftRows` step is needed. This step involves shifting all except the first row's elements, as shown visually in Figure 3.5. For the inverse of this step used in the decryption, the rows are shifted back to their original locations.

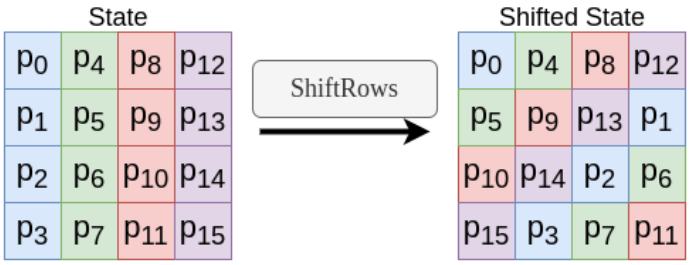


Figure 3.5: `ShiftRows` step of AES.

3.2.4 `MixColumns`

The final and most computationally expensive step [78] is called `MixColumns`. In combination with `ShiftRows`, diffusion (changing a single bit of the plaintext changes about half of the bits of the ciphertext [79]) is present in AES to improve its resilience to cryptanalysis. A column in the state is multiplied with a fixed polynomial $p(x)$ to create an updated state for each column, as demonstrated in Figure 3.6. To compute the polynomial in a lower-level representation, a matrix representation of $p(x)$ is multiplied with a column from the state as shown in Equation 3.5. The larger computational expense of this step results from the matrix multiplication performed, which is $\mathcal{O}(n^3)$ with better time complexities shown to be $\mathcal{O}(n^{2.376})$ [80].

$$\begin{bmatrix} b_{4j} \\ b_{4j+1} \\ b_{4j+2} \\ b_{4j+3} \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \begin{bmatrix} p_{4j} \\ p_{4j+1} \\ p_{4j+2} \\ p_{4j+3} \end{bmatrix} \tag{3.5}$$

For the inverse of this step to be used in decryption, the columns are reversed from their mixed columns using the inverse of $p(x)$. The inverse of $p(x)$ is represented using the inverse of its matrix. Note that the determinant of the representative matrix of $p(x)$ is -35. Note that any matrix with a nonzero determinant has an inverse.

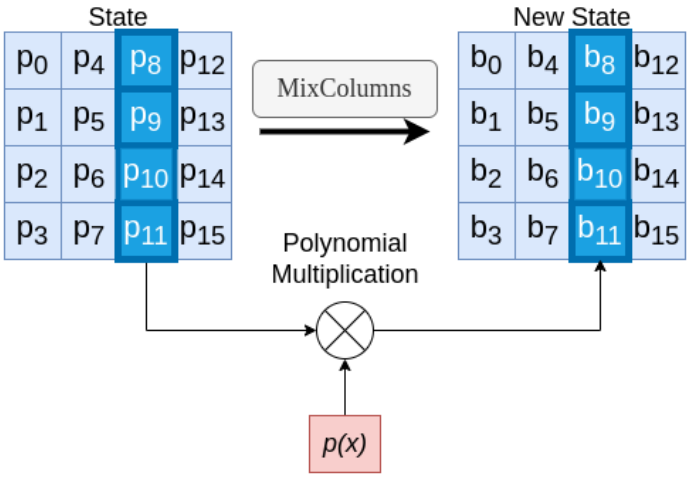


Figure 3.6: `MixColumns` step of AES.

Chapter 4

Power-Based Side-Channel Analysis

Encryption systems such as AES have been shown to be secure against certain attacks such as linear [40] and differential cryptanalysis [71, 81]; however, these protocols are still vulnerable to a class of attacks called side-channel attacks. Research has demonstrated that there are relationships between the encryption performed by a protocol and physical artifacts generated by evaluating the protocol on an actual computational device.

These artifacts are *intentional* properties of an implementation in the real world. These properties are distinguished from the *extensional* properties which are purely mathematical. The extensional behavior of a protocol can be used to formally prove correctness of the protocol with respect to certain mathematical models of correctness [9, pp. 8].

Examples of the physical artifacts that may have a bearing on the actual security of a protocol are: the noise generated by a Central Processing Unit (CPU) running a protocol [82], the electromagnetic radiation [83], the timing of the operations [84], the cache access [85], and the power consumption of a device [86]. Each of these leakage avenues of determining relationships between the device operations are called side-channels, and the study of these forms of leakage is called Side-Channel Analysis (SCA). SCA has risen in popularity over the past quarter-century as a meaningful field of research on the side of cryptanalysis to find vulnerabilities in encryption schemes, such as AES, that are strong against other attacks [85, 86].

One of the often utilized and stronger methods of side-channel attacks are power-based

side-channel attacks [87,88]. These attacks measure the small changes in power consumption of a device as it runs an encryption protocol. Information gathered from these observations can be used to devise attacks. In the following sections, we explore how these attacks can be formulated. As described by Randolph et al. [89, pp. 2], when an encryption protocol manipulates ones and zeroes in an electronic device, a current is "applied or removed from each transistor" in the device. As a result, when computations occur in a cryptographic algorithm within a device, power is being used by the device to make the algorithm run.

$$I = \frac{V}{R} \tag{4.1}$$

As a result of Ohm's law (demonstrated in Equation 4.1, where V = voltage, I = current, and R = resistance) [90], high quality measurements of the current can be obtained, but power-based noise from other computations on a device can obscure these power measurements. To counteract this noise, Randolph et al. [89] also mention that if the measurements are gathered as close as possible to the circuit that performs the cryptographic operations, the noise will be lower. Additionally, power consumption of a device changes during each step in time; what may have a high amount of power used at one time step can have little to no power consumed the next time step. Thus, power-based side-channel analysis assumes that an attacker can gain physical access to the computational device and knows which encryption protocol is being used. The collection of these power measurements over a fixed period of time is called a power trace.

Although only a single side-channel avenue, power-based side-channel analysis has a varied history behind it. There are several methods of utilizing the power leakage of a device as an attack, such as Simple Power Analysis (SPA), Differential Power Analysis (DPA), Correlation Power Analysis (CPA), and Test Vector Leakage Assessment (TVLA). Each of these methods have their strengths and weaknesses, and they vary on assumptions based on the information that a user might have. The following four subsections summarize each of these methods as well as describe how the each methodology developed.

4.1 Simple Power Analysis (SPA)

The eponymous, simple approach to analyzing algorithms is to directly observe the power traces. This method was introduced by Kocher et al. [86] to demonstrate how to visualize power traces in an effort to gather meaningful information about a running algorithm. As shown in Figure 4.1 [89, pp. 3], the 10 rounds of 128-bit AES are visually distinct from other power data, but each round looks similar in power consumption. This information can help an attacker identify which algorithm is being used by the device, but SPA information only provides an overall view of the power consumed by a device. Visualization of the data is easy to implement, but it does not reveal any information about the algorithm from an attacking standpoint. The visual observation of SPA shown in Figure 4.1does not reveal the secret key used in this AES operation[1].

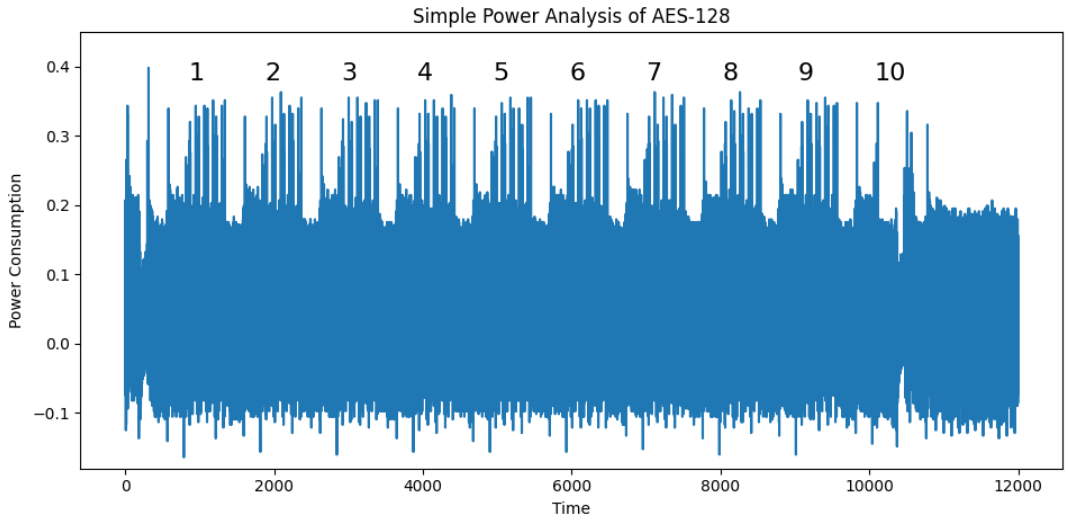


Figure 4.1: Simple Power Analysis Trace for an AES Operation.

Additionally, SPA is vulnerable to countermeasures against side-channel attacks and other power observations such as random noise in a device. While this attack is up to interpretation from the attacker to determine if leakage occurred, Mayer-Sommer [91] demonstrated that SPA is still a viable and simple side-channel attack method that can still be utilized even with other, stronger methods available.

[1]Provided by Alicia Thoney of the CEDAR lab.

4.2 Differential Power Analysis (DPA)

In an effort to add more empirical results from gathered power traces, DPA was proposed by Kocher et al. [86] in 1999. The influence of this work on power-based side-channel analysis has been called "the bedrock for research involving SCA at the device level" [89]. This method exploits the Hamming Weight of the input, as discussed in more detail in Section 4.3. This method can be used to automate the process of SCA on a device while also removing the interpretation bias of observing the traces. Additionally, DPA's techniques can be leveraged to retrieve the secret key from an encryption scheme such as DES [86],AES [3,4], and Camellia [92,93].

The process of DPA involves first gathering the power traces from a device running an encryption algorithm. The arithmetic mean is computed between each trace to yield an $N \times 1$ master trace vector, where N is the number of timesteps used in each power trace. The master trace is then compared to each point in the trace to determine if statistically significant differences are present.

Each possible input value is used in an attack along with each ciphertext as the comparison to the master trace. An attacker will probe for each subpart of the secret key by using this differential method on every possible value of the subkey. The correct subkey guess will be mathematically different in terms of its average difference in trace, but it also will look visually distinct from the incorrect guesses for the subkey value. Figure 4.2 demonstrates how the correct subkey can be visually represented against the incorrect subkey guesses, with the traces gathered using the ChipWhisperer-Nano device (Section 4.6). Each color represents a different guess for the subkey, and the magnitude of each difference indicates the best possible guess.

Despite the success of recovering the secret key from this technique, DPA still has countermeasures. Two countermeasures against DPA are masking and shuffling, as demonstrated by Rivain et al [94]. Masking is the process of randomly splitting every sensitive variable so that it seems not valuable to an attacker, and shuffling involves spreading the information of a sensitive variable's signals over a set amount of different time steps. Rivain et al. [94] demonstrates that these countermeasures can nullify DPA at the cost of efficiency.

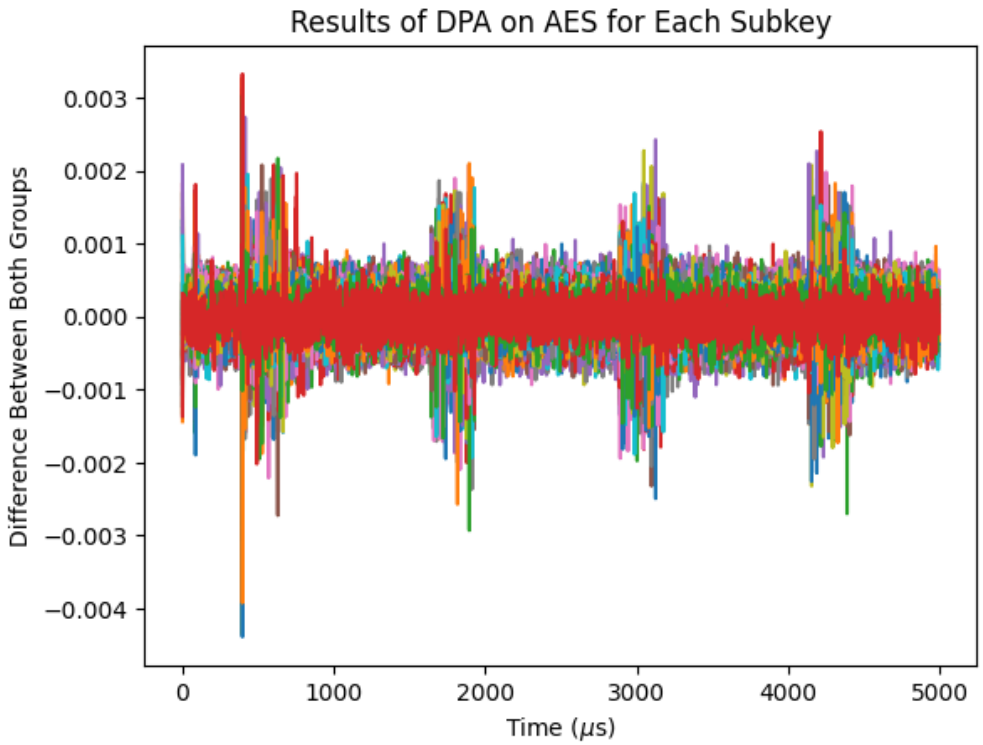


Figure 4.2: Demonstration of subkey guess on AES using DPA.

Additionally, boolean masking was described as a counterexample by Maghrebi et al. [95] by having sensitive data in an encryption protocol XOR with a random word. Finally, hardware countermeasures have been demonstrated by Ueno et al. [66] and Soares et al. [96] as ways to have encryption protocols resist DPA attacks. Despite such countermeasures, DPA can still find ways to find the secret information of the encryption protocol. Clavier et al. [97] details how to attack devices that have countermeasures such as random process interrupts and random noise power consumption. Even though these countermeasures aim to disguise the pattern of power usage, the cited papers [66, 95–97] demonstrate the pattern will still be present through the obfuscated computation in the device .

4.3 Correlation Power Analysis (CPA)

DPA exploits the Hamming Weight [98, 99], or the number of 1's in a binary representation, of computations [100]. Alternatively, we define the Hamming Weight of a natural number v

as $wt(v)$ defined in Equation 4.2. However, as mentioned in the introduction to this chapter, the power measured from a side-channel attack involve changes in bits being flipped from on to off. Using this knowledge, Brier et al [100] proposed the side-channel attack method of CPA. This attack involves using using the Hamming Distance between the input to a cryptographic device and the output produced within the cryptographic protocol. Using this, we can define Hamming Distance as shown in Equation 4.2 as the Hamming Weight of the XOR between two natural numbers in the function $hd()$. Using the XOR to calculate the Hamming Distance between the input and output of a cryptographic protocol, it can be seen what bits are changed through computation.

$$\begin{aligned} wt(v) &= \sum_{i=1}^{\text{bits in } v} v[i] \\ hd(u,v) &= wt(u \oplus v), \text{ For all } u, v \in \mathbb{N} \end{aligned} \tag{4.2}$$

For each of the possible subkey guesses, an attacker will correlate the input data and output data with the possible byte for the subkey. In particular, the correlation will be calculated using Pearson's Correlation Coefficient [101] which is given in Equation 4.3, where x, y are samples having bitlength n each, and $\bar{x}$ denotes the arithmetic mean of x. In statistical terms, the numerator of the equation is the covariance of x and y, and the denominator is the product of the standard deviation of x and the standard deviation of y.

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{4.3}$$

Rather than DPA using a separation of bits, CPA relies upon using a statistic in the form of Pearson's Correlation Coefficient. By adding this statistical power, CPA has more success in correctly recovering the secret key on an attacked cryptographic protocol, and it can recover the same information as DPA more accurately and with fewer samples [100]. As a result, CPA has more statistical strength than DPA, but it comes at the cost of the strength of the assumptions that must be made. In addition to knowing what cryptographic protocol is being implemented, an attacker must also assume it knows what sort of operations are being utilized by that protocol. Pearson's Correlation Coefficient also requires an assumption

that the data input is normally-distributed [8,102]. An attacker could attack AES's `SubBytes` portion of the algorithm (see Section 3.2.2) by determining the correlation between the input text and the output text through the S-box after using a secret key [4,89], or they could find correlations between the input text and the first instance of the `AddRoundKey` function [100] (see Section 3.2.1). This attack is also not limited to AES, as Xia et al. [103] demonstrated that the encryption protocol LiCi can also be attacked with CPA on analyzing the input and output of the S-box used in LiCi. Despite these successful attacks, this requires that an attacker knows what S-box is being utilized by the algorithm to make a correct conclusion. Figure 4.3 demonstrates the workflow of CPA being performed on AES.

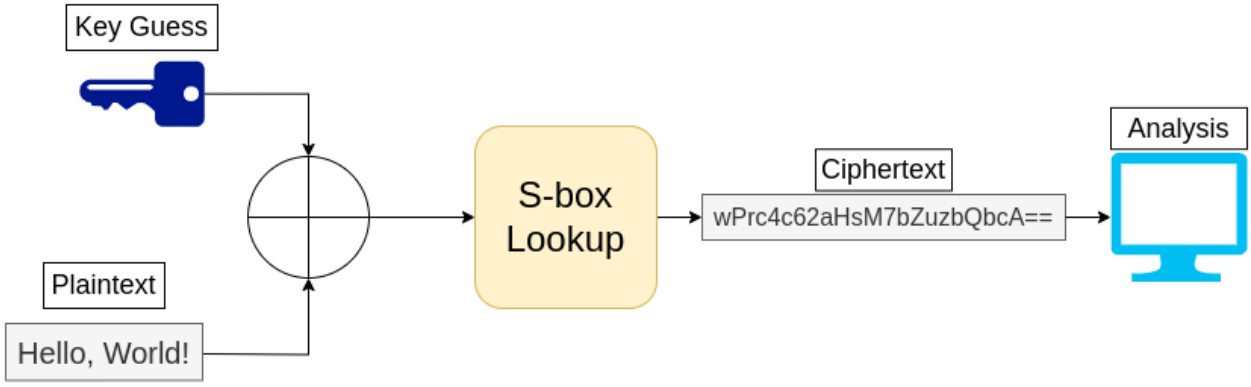


Figure 4.3: Outline for CPA attack on the `SubBytes` step of AES.

This attack can make the correct key more apparent, even from a visual standpoint. Figure 4.4 demonstrates the result of attacking AES using CPA, and the peaks of the correlation are the same color, demonstrating the correct subkey for the secret key. Using this powerful technique not only mathematically differentiates the correct subkey guess from the incorrect ones, but it is also visually distinct from incorrect guesses.

Despite the strength of CPA, there are countermeasures against this attack method. One such countermeasure involves the protocol using masking techniques that work as a countermeasures against DPA attacks [104]. In particular, Brier et al. [100] mentioned that the same countermeasures used to strengthen DPA also work against CPA. This is because both attacks rely on "side-channel observability." Despite countermeasures being in place, CPA can circumvent these by utilizing other techniques such as merging patterns from each round [105]. Thus, CPA is a strong alternative to DPA that has its strengths in recovery

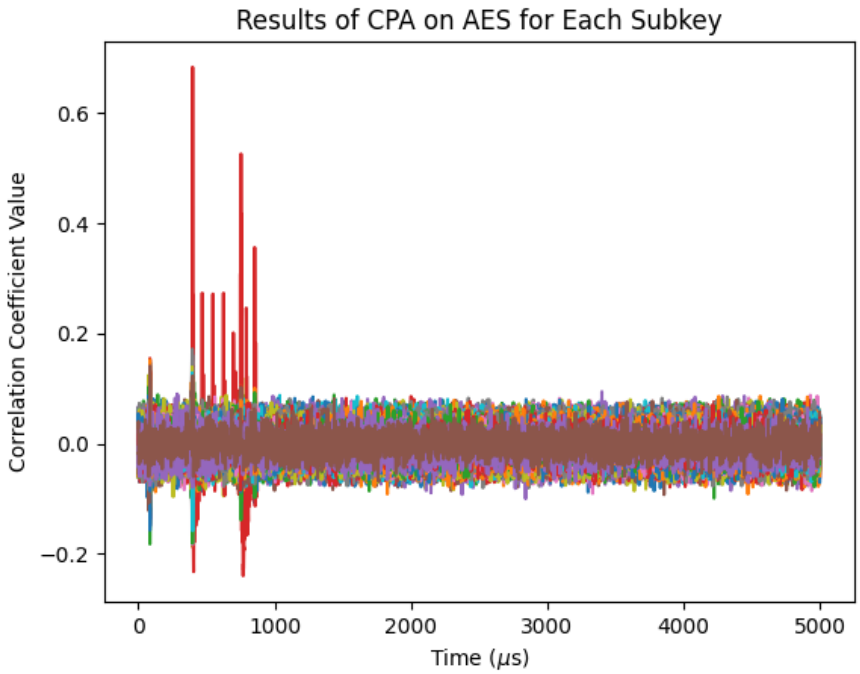


Figure 4.4: Results of a subkey guess on AES using CPA.

of secret keys, but it must make assumptions about the normality of the data and the implementation of the S-box being used [8]. If the assumption of the normality of the data is false, then the conclusion drawn from performing CPA does not have the statistical justification required to use Pearson's Correlation Coefficient. If the S-box is unknown, then CPA cannot be performed to attack the `SubBytes` step because that requires knowledge of the S-box.

4.4 Test Vector Leakage Assessment (TVLA)

As a generalization of specific power-based side-channel attacks, Goodwill et al. [106] proposed the method of TVLA to NIST. This side-channel analysis method differentiates itself from SPA, DPA, and CPA by analyzing the leakage detection of a Device Under Test (DUT) rather than leakage exploitation. This process involves taking power traces gathered from a device running a cryptographic protocol and separating the samples into two different groups using a selection function. Figure 4.5 demonstrates how TVLA can be used for side-channel

analysis. A common way of separating samples of TVLA into two groups is to gather trace information of fixed-plaintext versus random-plaintext inputs. By doing the fixed versus random separation, the two groups can then be compared to each other to find any statistical differences. Tests that can be used for statistical differences involve Welch's t-test [106, 107], Pearson's χ^2 test [108, 109], or the Wilcoxon Signed-Rank Test [8, 110]. These tests determine if there is significant evidence to support the claim that there was leakage from the DUT from its power traces. Welch's t-test assumes the normality of the data [107], and Pearson's χ^2 test assumes data independence [111], while the Wilcoxon Signed-Rank test is a non-parametric test; meaning that there are no assumptions made about the distribution of the data.

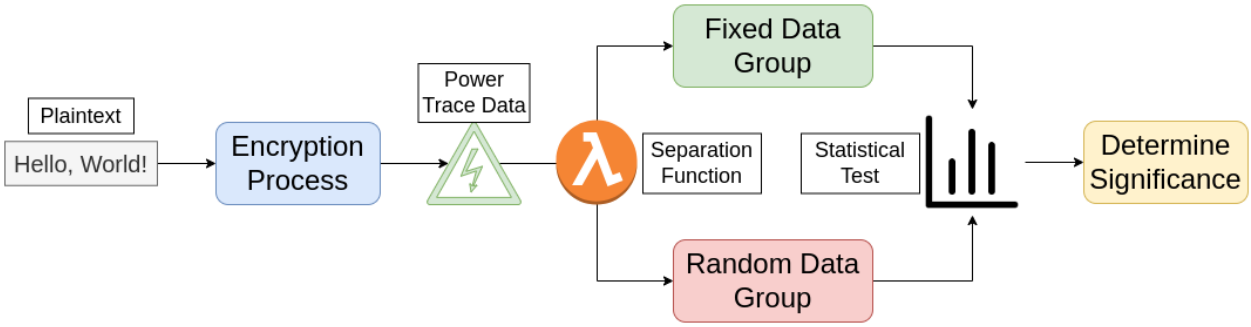


Figure 4.5: TVLA Attack.

If a DUT does not yield significant leakage, then it can be assumed at a certain confidence level that no leakage occurred. However, even if there is leakage, that does not accurately portray the effort of a side-channel attack [112]. Trace gatherings could be very difficult to gather and still have significant leakage, so the result could yield more wariness about the security of the DUT than is necessary. Additionally, even if leakage occurs, it does not provide information on how to attack the cryptographic device. Even if leakage does occur, the power-based side-channel attack could be difficult to formulate because statistical significance does not correlate with any particular attack method [113].

The usage of TVLA has been extended to work for block ciphers such as GIFT [114], asymmetric cryptographic algorithms [115], and even AES [89, 106]. This switch from the specific exploitation of previous side-channel attacks to a more generalized leakage analysis allows for more flexibility in attacks, and it provides more empirical results about how

resistant a cryptographic protocol is to power-based side-channel attacks. Papagiannapolis et al. [112] demonstrate that TVLA is an effective metric to analyze a cryptographic protocol.

4.5 AES as an SCA Target

AES has been an attack target in each of the previous sections. This is to demonstrate the common usage of power-based side-channel attacks on AES. AES is targeted by SPA [65], DPA [3,4,66,116], CPA [4,100], and TVLA [106,115,117]. The precedence of attacking AES using a variety of side-channel attacks validate the research interest of what aspects of AES are more vulnerable to side-channel attack.

As mentioned in Section 4.3, CPA can be used on different steps of AES such as the `SubBytes` step (Section 3.2.2). In particular, this function utilizes the S-box along with the secret key to start encrypting the plaintext. However, there have been different implementations of S-boxes that can be used with AES because they are 8-bit S-boxes. A suite of these S-boxes are presented by SageMath [118] that can be easily assessed for their properties along with the S-boxes themselves. Table 4.1 displays the name of each of these S-boxes together with the papers where they originate.

Since all of these S-boxes are designed with different properties such as nonlinearity, and because each S-box is an 8-bit S-box, we can put each S-box in the AES algorithm and determine what properties affect side-channel resistance in AES. Thus, the precedence of using AES as an SCA target combined with many S-boxes with varying properties being readily available lends itself to being a useful encryption scheme to research.

4.6 ChipWhisperer

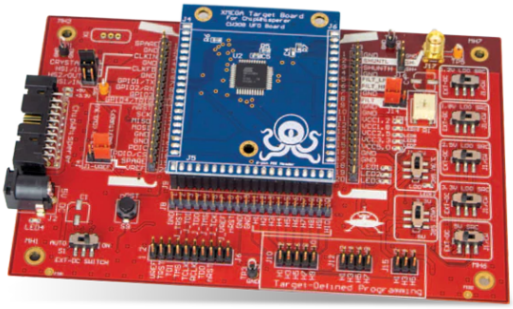
The ChipWhisperer is a platform for power-based side-channel analysis that was created by Colin O'Flynn [155]. The goal of the ChipWhisperer was to create an open-source accessible hardware and software for power-based side-channel analysis intended for research. Some of the open-source software gives examples of using DPA and CPA to attack AES's secret

Substitution Box (S-box) Name	Authors
ANUBIS	P. Barreto & V. Rijmen [119]
ARIA_s2	D. Kwon et al. [120]
BelT	Belarusian State University [121]
Camellia	K. Aoki et al. [93]
CMEA	D. Wagner et al. [122]
Chiasmus	J. Schejbal [123]
CLEFIA_S0, CLEFIA_S1	T. Shirai et al. [124]
Crypton_0_5	C.Lim [125]
Crypton_1_SX (X $\in \{0, \ldots, 4\}$)	C.Lim [126]
CS_cipher	J. Stern & S. Vaudenay [127]
CSA	R. Weinmann & K. Wirt [128]
CSS	M. Becker& A. Desoky [129]
DBlock	W. Wu et al. [130]
E2	M. Kanda et al. [131]
Enocoro	D. Watanabe et al. [132]
Fantomas	V. Grosso et al. [133]
Fox	S. Vaudenay & P. Junod [134]
Iceberg	F.-X. Standaert et al. [135]
Iraqi	Anonymous [136]
Scream, iScream	V. Grosso et al. [28]
Kalyna_piX (X $\in \{1, \ldots, 4\}$)	R. Oliynykov et al. [137]
Khazad	P. Barreto & V. Rijmen [138]
Kuznyechik, Kuznechik, Streebog, Stribog	V. Dolmatov [139]
Lilliput-AE	A. Adomnicai et al. [140]
MD2	B. Kaliski [141]
newDES	R. Scott [142]
Picaro	G. Piret et al. [143]
Safer	J. L. Massey [144]
SEED_S0, SEED_S1	H. J. Lee et al. [145]
SKINNY_8	C. Beierle et al. [146]
ForkSkinny_8	E. Andreeva et al. [147]
Remus_8	T. Iwata et al. [148]
Romulus_8	T. Iwata et al. [149]
S_X (X $\in \{1, \ldots, 8\}$)	Siddiqui et al. [150]
Skipjack	NIST [22]
SNOW_3G_sq	ETSI/Sage [29]
SMS4	F. Liu et al. [151]
Turing	G. G. Rose & P. Hawkes [30]
Twofish_p0, Twofish_p1	B. Schneier et al. [26]
Whirlpool	P. Barreto & V. Rijmen [152]
Zorro	B. Gerard et al. [153]
ZUC_S0, ZUC_S1	ETSI/Sage [154]

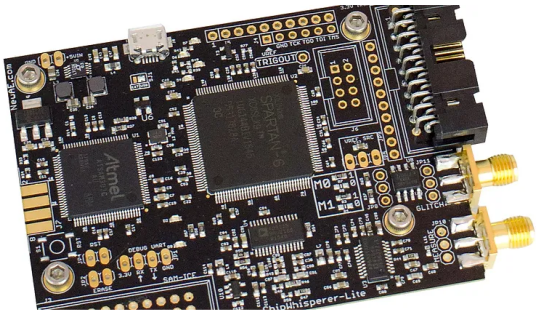
Table 4.1: S-boxes from SageMath.

key [156]. Additionally, integration of the ChipWhisperer platform is compatible with the lightweight-implementation of tiny-AES-c [157].

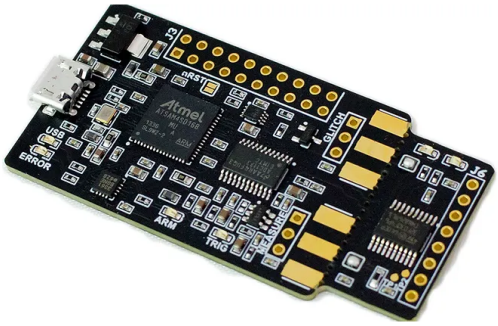
Two platforms of interest for collecting the power-trace data are the ChipWhisperer-Nano (CW-Nano) [158] and the ChipWhisperer-Lite (CW-Lite) [159]. Currently costing only $50, the CW-Nano is a low-cost alternative to the ChipWhisperer Starter Pack which currently costs $690; the starter pack provides the components CW-Lite. The CW-Lite device can be used as a capture-only device that integrates with the UFO Target Board [160] which can run with Atmel XMEGA [161] or STM32F Arm programmers [162]. As a result of this, there is flexibility in choosing which programmer will be used to run a particular encryption scheme. The main microchip that will be used by the CW-Lite will be the STM32F303 ARM microchip [163]. Figure 4.6a, taken from its datasheet [160] shows the UFO Board and Figure 4.6b shows the CW-Lite Capture device as displayed in its datasheet [159]. The CW-Nano can gather more data points per power trace than the



(a) ChipWhisperer-Lite UFO Board [160].



(b) ChipWhisperer-Lite Capture [159].



(c) ChipWhisperer-Nano [158].

Figure 4.6: Figures of ChipWhisperer targets from their datasheets.

CW-Lite (50,000 as opposed to 24,400); however, the quality of those data points are not as precise (8 bits for CW-Nano and 10 bits for CW-Lite) [158, 159]. The number of bits precision used in the data points for the CW-Nano's gathered power traces are less than the number of bits in the CW-Lite's power traces. Despite this difference, the CW-Nano can capture and run a target on the single board. Figure 4.6c displays the CW-Nano device from its datasheet [158], where the smaller portion of it is the target device, and the larger portion is the capture device.

Both the CW-Lite and the CW-Nano can gather power traces that lead to successful DPA and CPA attacks. Thus, these devices are cost-effective tools for researching power-based side-channel attacks and analysis.

Chapter 5

Substitution-Box (S-Box) Analysis for Side-Channel Resistance

The previous chapters established the Advanced Encryption Standard (AES) as widely-used (Chapter 3), but it is also vulnerable to different methods of side-channel attacks (Chapter 4). In particular, the Substitution-box (S-box) used in AES is an important component in the strength of the algorithm. As a result, the strength of AES can be affected by what S-box is used in AES. As demonstrated in Table 4.1, there are many S-boxes that can be used in AES without changing the rest of the algorithm. The choice of S-box can determine the susceptibility of an implementation of AES to side-channel attacks. The analysis of S-boxes and their contribution to the cryptographic strength of AES is the main subject of this thesis. The methods of analysis of S-boxes and their properties' effects on side-channel attack resistance is described in this chapter. The code used for this experiment is publicly available on GitHub at `https://github.com/A-Stone-Olguin/S_Box_Analysis`.

A high-level overview of the methods used is provided in Figure 5.1. This process first involved determining what S-boxes would be used and what properties would be measured from them (Section 5.1). Additionally, the ChipWhisperer software required modifications so that AES could be changed to have a different S-box implemented, and the ChipWhisperer devices would then be programmed with the modified AES (Section 5.2). Metrics used for evaluating the side-channel strength of AES for each S-box were implemented and

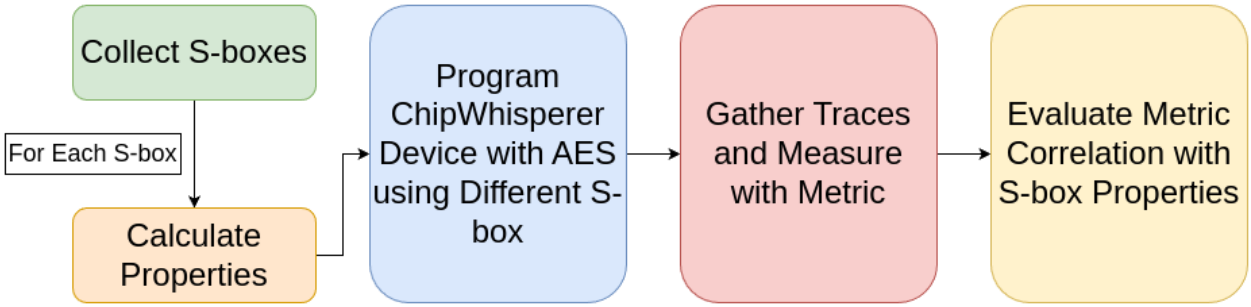


Figure 5.1: Workflow for Experimental Methods.

computed using the traces gathered from a ChipWhisperer device (Section 5.3). Finally, given each S-box's results for each metric, analytical results were made to determine what properties measured from the S-boxes had statistically significant effects towards the side-channel resistance of AES (Section 5.4). This process involved ten total experiments on each different S-box. Each experiment applied two metrics to measure side-channel resistance of AES using that particular S-box.

5.1 Collecting S-boxes

As mentioned in Chapter 4, SageMath contains a suite of S-boxes defined in Table 4.1. These S-boxes are all of the 8-bit input and 8-bit output S-boxes that are defined in SageMath Version 9.3 [118]. SageMath is run on Python version 3.10.12 by using a file named `sboxes_info.py`. This Python script computed each S-box's properties and put the results in a pickle file. Additionally, eight more S-boxes ($S_1, \cdots, S_8$) were gathered from Siddiqui et al. [150]. These additional eight S-boxes were added for comparison with Siddiqui et al.'s exploitation of different S-box properties.

Each S-box from SageMath is represented by a class called `SBox`. This class has its own member functions to calculate different properties of each S-box. Each property of the S-box is detailed in Section 5.1.1. These properties each are a measure of cryptographic strength for each S-box, and each of these were selected to see if any property could have a significant effect on the side-channel leakage of AES.

ChipWhisperer expects S-boxes to be as lists in Python, not SBox types. In order

to use the S-boxes in SageMath with the ChipWhisperer software, the values of each S-box and its inverse were put into lists of integers. This was done in a separate script (`sboxes_info.py`) because the ChipWhisperer software uses Python version 3.9.5, and this version is not compatible with SageMath. This conversion of S-boxes in list datatypes allowed for the ChipWhisperer to run correctly with each different S-box. This method worked because only the values were needed for the S-box to run on ChipWhisperer, but the results for each property were stored in a pickle file to keep track of each S-box's properties.

5.1.1 S-box Properties

The desired properties of S-boxes were inspired by Siddiqui et al. [150]. Each S-box from Siddiqui et al. had its Strict Avalanche Criterion (SAC), Bit Independence Criterion (BIC), Linear Approximation Probability, Differential Approximation Probability, and nonlinearity calculated.

Nonlinearity is the measure of how well the algebraic structure of the S-box is concealed. A high nonlinearity means that the algebraic structure is well concealed. The nonlinearity is a calculation of using the Walsh-Hadamard matrix XOR-ed as a boolean formula. Additionally, the **linearity** is the measure of the "lack of nonlinearity." The linearity is computed by computing the nonlinearity and subtracting the nonlinearity from the largest possible nonlinearity that could be calculated. The nonlinearity property measures how well the algebraic structure of the S-box is concealed. SageMath calculated these values for an S-box with the `nonlinearity` and `linearity` functions.

Differential Probability and **Linear Probability** are both measures of algebraic structures of an S-box. Differential probability measures the differential uniformity. Linear probability measures how much algebraic bias an S-box has. A low linear probability means that there is a low probability of a single element of an S-box demonstrating the algebraic structure. Similarly, a low differential probability indicates that there is a lower probability that there is uniform changes in input and output of an S-box. Linear probability was computed by the SageMath function `maximal_linear_bias_relative` and differential probability was computed using SageMath's using `maximal_difference_probability`.

Two measures of diffusion in the S-box are the **linear branch number** and **differential branch number** [48]. SageMath can compute the linear branch number of an S-box with the `linear_branch_number` function and the differential branch number with the function `differential_branch_number`.

The **boomerang uniformity** property, or the measure against boomerang-style attacks [164],was computed using SageMath's `boomerang_uniformity` function.

The final two metrics, **BIC** and **SAC** do not have functions available in SageMath. BIC is a measurement of the variables of the input and output of an S-box. SAC is the measure of how changing a single input bit changes half of the output bits through an S-box with an ideal average SAC being 0.5. Although not available through SageMath, the BIC and SAC were computed with modified code from Muhammad Abrar's block-cipher-testing repository on Github [165].

The measure of side-channel resistance was predicted using the properties shown in Table 5.1. Some properties could be less significant than others in terms of side-channel resistance for a given metric, but that does not mean the property itself is insignificant. An example of the properties for the Camellia S-box and their values is shown in Table 5.1. The full results for each S-box is given in Chapter 6.

Property	Value
BIC	0.131696
Boomerang Uniformity	6.000000
Differential Branch Number	2.000000
Differential Probability	0.015625
Linear Branch Number	2.000000
Linear Probability	0.062500
Linearity	32.000000
Nonlinearity	112.000000
SAC	0.498291

Table 5.1: S-box Properties and their values for the Camellia S-box.

5.2 ChipWhisperer Setup

The original open-source GitHub repository for ChipWhisperer [166] contains many additional files that are not needed to test AES on different S-boxes. Automating each test would involve running multiple Jupyter notebooks that use global variables, and this would be space-inefficient for gathering power traces. To efficiently use the ChipWhisperer platform and not have extraneous files, a minimal working build was developed. Automation of programming a ChipWhisperer device with a modified AES with a different S-box would allow for testing to be run in an iterative fashion without requiring a user to manually change the S-box used in AES (See Algorithm 2 on Page 50).

5.2.1 ChipWhisperer Minimal Working Build

To make a more automated and specific version of the ChipWhisperer software, a directory named `chipwhisperer_minimal` was created as a minimal build of the ChipWhisperer GitHub repository. Most of the original ChipWhisperer repository was removed for the minimal build, with notable removals being the software and firmware directories. These two directories are used for the setup of ChipWhisperer, and they are not needed after having a system set up for use with ChipWhisperer devices. The hardware directory was also modified to only include using files that would be needed to work with the ChipWhisperer-Lite (CW-Lite) and ChipWhisperer-Nano (CW-Nano). The subdirectories of hardware in the original repository were `capture`, `common/hd`, `tools`, and `victims`. Of these sub-directories, only `victims` was kept, but the subdirectories of `victims` were also modified. The only kept subdirectory of `victims` was `firmware`. This was done because the only targets used for both CW-Lite and CW-Nano are in their firmware. As a result of this, the `chipwhisperer_minimal` directory only includes the `firmware` subdirectory at the top level. Figure 5.2 shows the reduction from the original repository to the minimal build folder.

The `firmware` directory contains the code that is compiled and executed on the target device. In particular, the `simpleserial-aes` and `simpleserial` subdirectories establish serial communication between the target device and the capture device. The `simpleserial-aes`

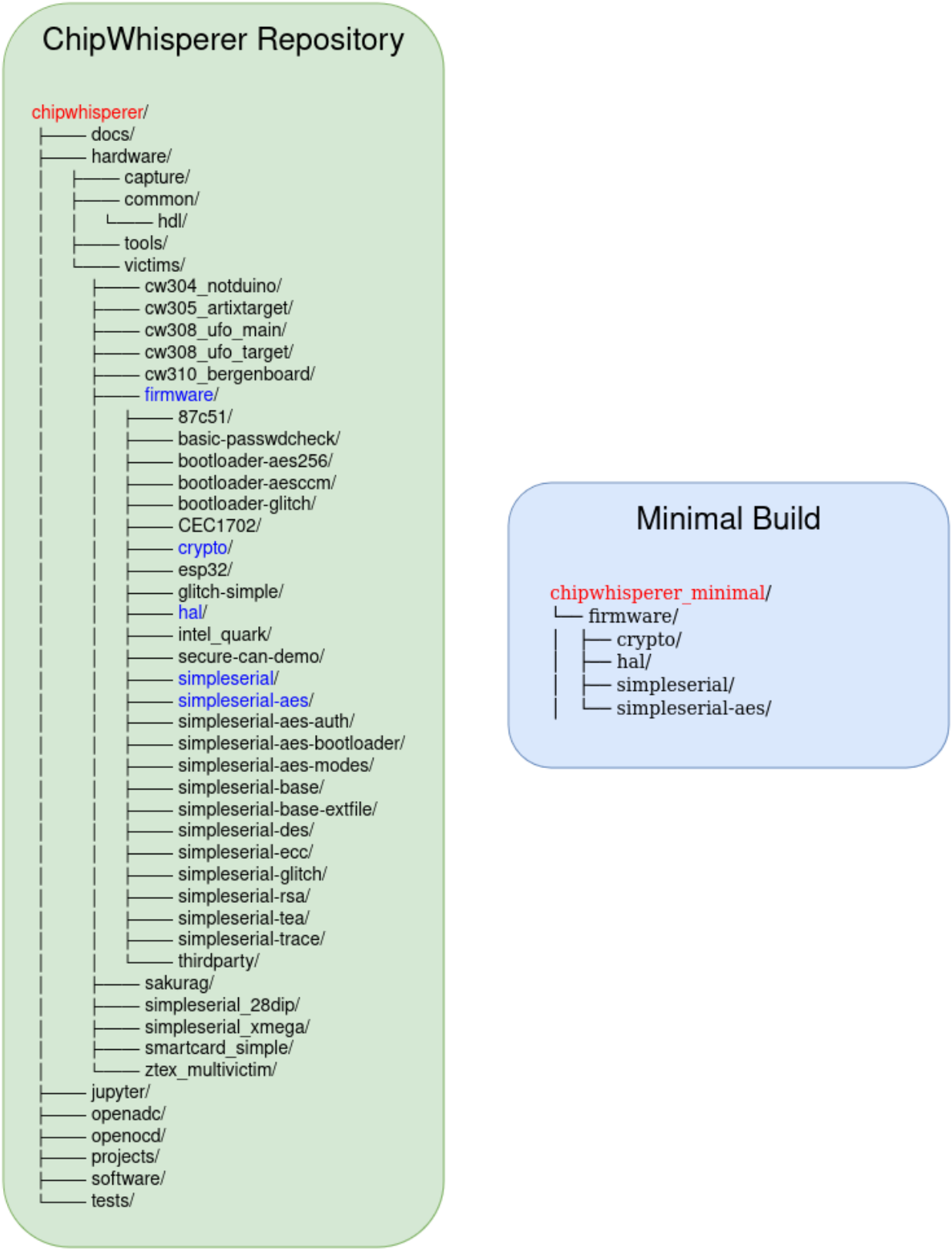


Figure 5.2: Comparison between minimal build and original ChipWhisperer repository.

subdirectory runs AES on the target device using the C file of the same name. The type of AES can be varied and specified using Makefile flags, but the default version of AES used is Tiny-AES-C[1] [43]. Because the only version of AES that would be used for this experiment is Tiny-AES-c, the `chipwhisperer/hardware/victims/firmware/crypto` directory was simplified to only contain the `tiny-AES128-C` folder and the `aes-independant.c` [sic][2] and `aes-independent.h` files. The `aes-independent.c` and associated `.h` files interact with Tiny-AES128-C's `aes.c` and `aes.h` files to program the target Tiny-AES-C.

Finally, the only other directory in `chipwhisperer/hardware/victims/firmware` kept in the minimal build was the `hal` subdirectory. This subdirectory contained the necessary files to program all of the different ChipWhisperer devices. However, the only devices used were the CW-Lite with the STM32F3 programmer, and the CW-Nano with the STM32F0 programmer. As a result, the only subdirectories that were not deleted for the minimal build were `stm32f0, stm32f0_nano, stm32f3`, and the only files used were the header file `hal.h` and `Makefile.hal`. Keeping these subdirectories allowed for each target to be programmed correctly with their programmers, and `Makefile.hal` in conjunction with `hal.h` enables the `make` utility to choose which programmer to use based on the `MAKEFLAGS`. Figure 5.3 demonstrates how the `hal` and `crypto` subdirectories of `firmware` were simplified to only require files for the CW-Nano and CW-Lite for Tiny-AES-C.

By using the `tree` command and analyzing the properties of the `chipwhisperer` repository, the original repository uses about 3.8 GB of space with 1923 subdirectories and 10790 total files. By contrast, the `chipwhisperer_minimal` directory uses 22.2 MB of space with 35 subdirectories and a total of 201 files. In terms of size, the `chipwhisperer_minimal` uses about 0.6% of the space, only about 1.8% as many subdirectores, and about 1.9% as many files. This minimal working build for the software for ChipWhisperer makes it clearer that it will only work for the intended devices, and looking through all files takes significantly less time

[1] `https://github.com/kokke/tiny-AES-c`.

[2] The misspelling of "independent" as "independant" finds its way into filenames and function names in the ChipWhisperer software implementation of AES. Rather than point out this error each time it appears in this thesis, we simply correct the spelling error. However, the spelling error remains in the actual code. I considered fixing the error, but this would require a significant refactoring of the code including makefiles, directory names, and code.

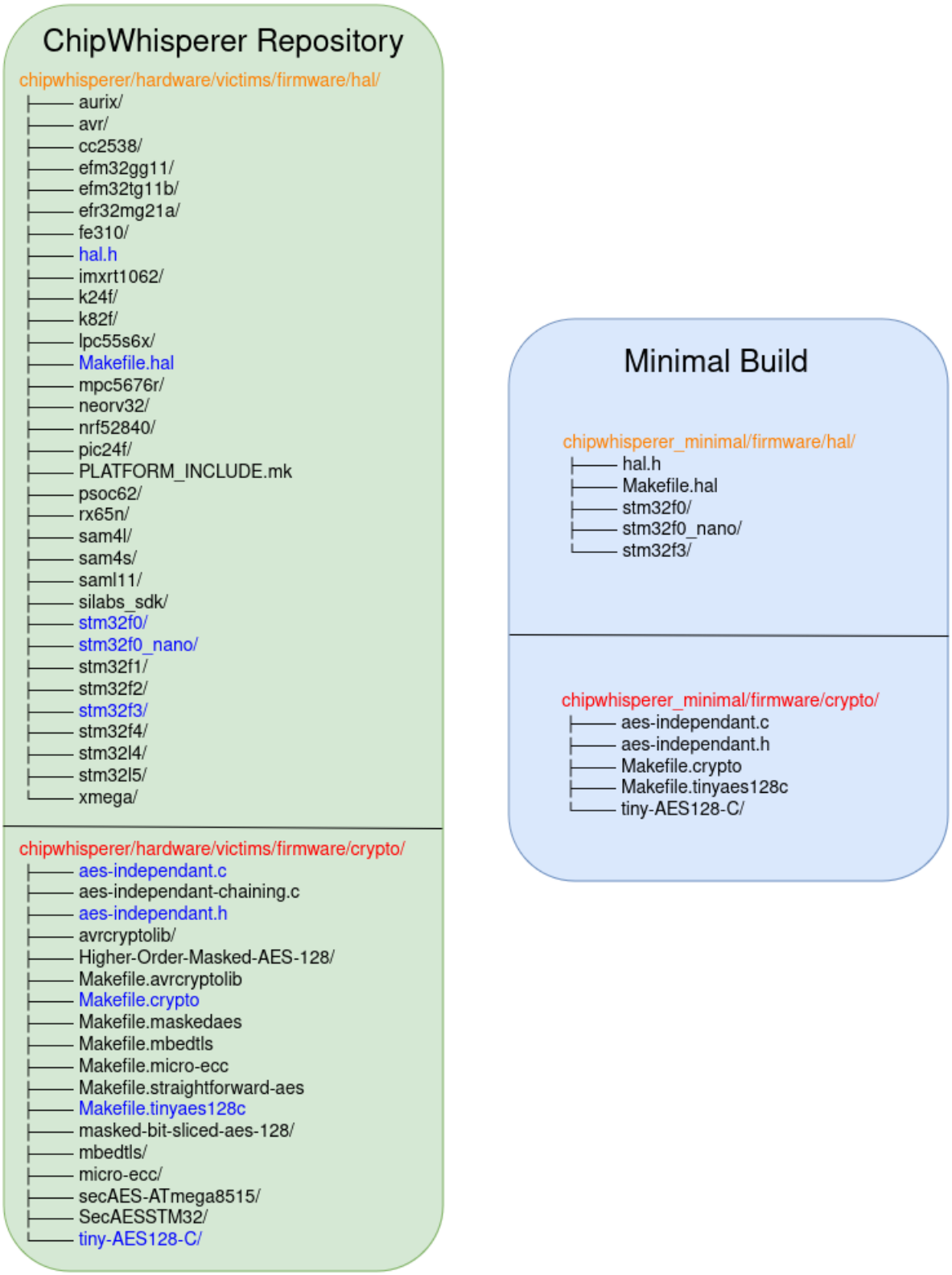


Figure 5.3: Visualization of minimization of `hal` and `crypto` directories.

and effort.

5.2.2 Updating Tiny-AES-C

The ChipWhisperer contains an implementation of Tiny-AES-C that only works on AES-128 in ECB mode. As mentioned in Chapter 3, two other frequently-used modes are CBC and CTR, and other key-lengths of 192 and 256 bits can be used for AES. In fact, the Tiny-AES-C code used by the ChipWhisperer software was added in April 2017. Since that date, Tiny-AES-C has been modified to be used on CBC and CTR modes and on different key lengths.

To determine if different modes would affect leakage of data from side-channel attacks (described later in Section 5.3.2), each of the three modes needed to be implemented to run on the ChipWhisperer software. First, this involved changing the files `aes.c` and the associated `aes.h` file in the `firmware/crypto/tiny-AES128-C` directory to be the updated versions from the up-to-date Tiny-AES-C repository that was last updated in December 2021. This was done by simply replacing `aes.c` and `aes.h` with their updated versions.

However, the up-to-date version of Tiny-AES-C accounts for specifying each different mode by flags to the compiler (or through `MAKEFLAGS`). To account for this, an extra makeflag was specified in `chiphwisperer_minimal/firmware/simpleserial-aes/makefile` named `AES_MODE`. Based on the value of `AES_MODE` specified during the making of the firmware, the `CFLAGS` will be updated to add what mode of AES will be used during compilation of `aes.c`.

To accurately use each mode, the AES encryption function from `aes-independent.c` from the `chipwhisperer_minimal/firmware/crypto/` directory had to be modified to accurately run AES in a specified mode. The original version of this file involved using different AES implementations such as the Hardware's AES, AVR Crypto Library, DPAV4, and Tiny-AES-C. This was modified to only use the Tiny-AES-C implementation to maintain consistency as the default AES implementation used by ChipWhisperer. The functions to set the key and encrypt using AES were modified to work with the up-to-date version of Tiny-AES-C. Originally, `aes-independent.c` would use a separate function called `aes_indep_key` to set the key and `aes_indep_enc` to encrypt the plaintext. Each of these function calls used the

old version of Tiny-AES-C that would specify to use only AES-128 in ECB mode. One feature that was retained from the original `aes-independent.c` was the secret key defined in hexadecimal as

$$key = \texttt{[2b, 7e, 15, 16, 28, ae, d2, a6, ab, f7, 15, 88, 09, cf, 4f, 3c]}.$$

The pseudocode of `aes-independent.c` is displayed in Algorithm 1. The updated version of Tiny-AES-C involves using a structure type to give context for AES named `AES_ctx`. This structure stores the secret key after being initialized, and if the mode needs it, the context will also store the Initialization Vector. As a result, an `AES_ctx` structure named `ctx` is defined outside of the key initialization and encryption functions in `aes-independent.c`. Initializing the key was only slightly changed to abstract for each mode of AES. If the AES mode was defined as CBC or CTR, then a fixed but arbitrary Initialization Vector is defined as `[0x00, ..., 0x0f]`, or an array of one through sixteen in hexadecimal. Then, `ctx` is intialized using Tiny-AES-C's function `AES_init_ctx` using the secret key and Initialization Vector. The encryption using each mode in `aes_indep_enc` was modified to case based on the specified mode by using each mode's encryption function from Tiny-AES-C.

These changes allow for a user to simply state what mode will be used as a makeflag when making the firmware, and the associated mode will be programmed on the target device. This ease of use allows for easy automation of using different S-boxes through AES on different modes by simply changing a makeflag.

5.2.3 Automating AES with Different S-boxes

The original ChipWhisperer software has implementations to run CPA and DPA on AES by using their tutorial Juptyer notebooks [156]. However, these notebooks use global variables that can be inefficient from a memory standpoint. Additionally, some of the implementations for these tutorials use several Jupyter notebooks that pass global variables between each other, and this can lead to confusing results on what data is being stored in memory. One of these global variables is the original Rijndael S-box, and this cannot be used when the S-box is being varied. Also, the Jupyter notebook tutorials only demonstrate how to break

Algorithm 1 Modified AES Independent code

```
1: ctx ← AES_ctx
2:
3: function aes_indep_key(key)
4:     if AES mode is CBC or CTR then
5:         iv ← {0x00, 0x01, 0x02, 0x03,
6:               0x04, 0x05, 0x06, 0x07,
7:               0x08, 0x09, 0x0a, 0x0b,
8:               0x0c, 0x0d, 0x0e, 0x0f}
9:
10:        AES_init_ctx(ctx, key, iv)          ▷ Initialize with key and IV
11:    else
12:        AES_init_ctx(ctx, key)              ▷ ECB does not need IV
13:    end if
14: end function
15:
16: function aes_indep_enc(plaintext)
17:    if AES mode is CBC then
18:        AES_CBC_encrypt(ctx, plaintext)     ▷ Encrypt in CBC
19:    else if AES mode is CTR then
20:        AES_CTR_encrypt(ctx, plaintext)     ▷ Encrypt in CTR
21:    else
22:        AES_ECB_encrypt(ctx, plaintext)     ▷ Encrypt in ECB
23:    end if
24: end function
```

a key, and they do not provide any measurement of the side-channel resistance of AES.

The first step to automating the process of running AES on different S-boxes involved changing Tiny-AES-C's `aes.c` file to have an acceptable S-box that could be changed. This was achieved by taking the original `aes.c` and only changing the file at the location where the `sbox` variable (and its inverse) are defined. The subdirectory `generate_c` achieves this using its Python script, `generate_c_files.py`. This directory contains `.c.part` files that are portions of `aes.c`, and each filename is described in Table 5.2.

.c.part file name	Contains
`aes_prelude`	The part of `aes.c` before the `sbox` declaration
`aes_inverse`	The inverse (`rsbox`) of the original Rijndael S-box
`aes_postlude`	The part of `aes.c` after the `rsbox` declaration

Table 5.2: Description of part files for generating `aes.c`

A function in `generate_c_files.py` named `generate_c_files` has a single input being the name of the desired S-box. The pseudocode for `generate_c_files` is shown in Algorithm 2. This name is checked to see if it is one of the S-boxes from Table 4.1, and if it is, the list that was mentioned as stored in Section 5.1 is taken from the pickle file. If an inverse for this S-box name exists, then that list is also retrieved from the stored data. The list S-box is converted to a string that shows the S-box as an array in C syntax. If an inverse exists, that list is also converted to a string, otherwise, the file `aes_inverse.c.part` is read in as a string. Finally, both `aes_prelude.c.part` and `aes_postlude.c.part` are read into strings. All of these strings are concatenated into a single string that overwrites `aes.c`. As a result, given a name of a desired S-box, the result will generate a modified `aes.c` where the `sbox` variable has been changed to the desired S-box.

Algorithm 2 Generate `aes.c` with a differing S-box

```
1:  function generate_c_files(sbox_name)
2:      box ← S-box of sbox_name
3:      inv ← inverse S-box of sbox_name
4:
5:      if inv is not empty then
6:          c_inv ← read content of aes_inverse.c.part and concatenate into a string
7:      else
8:          c_inv ← string of inv as a c-array
9:      end if
10:
11:     c_box ← string of box as a c-array
12:     prelude ← read content of aes_prelude.c.part and concatenate into a string
13:     postlude ← read content of aes_postlude.c.part and concatenate into a string
14:
15:     write (prelude + c_box + c_inv + postlude) to aes.c
16: end function
```

With the AES code created for the target device, the next step is to create the `.hex` firmware file that will be used to program the device. This firmware file can be created using the Makefile defined in `chipwhisperer_minimal/firmware/simpleserial-aes`. As mentioned in Section 5.2.2, this Makefile was changed to have an additional makeflag that will change the mode AES will run on. To incorporate all of this in a single file as well as ensure that the correct S-box is loaded in `aes.c`, a function named `make_firmware` is called. This function

takes as input a name for an S-box, the device that will be used (either the CW-Lite or CW-Nano) and the AES mode which defaults to ECB if not specified. The name of the S-box is used as input to `generate_c_files` to update `aes.c` with the desired S-box to program the ChipWhisperer device. After this has been executed, a subprocess is run that calls `make` as a command to the `simpleserial-aes` Makefile with the `MAKEFLAGS` specifying which device and AES mode are being used. Additionally, a makeflag of `CRYPTO_TARGET` is specified as `TINYAES128C` to use the Tiny-AES-C implementation of AES.

After the firmware has been generated, all that remains is to program the device. However, because a subprocess was called in `make_firmware`, the program sleeps for one second to ensure that the subprocess can finish before continuing. The ChipWhisperer tutorial would start the process of programming the ChipWhisperer device by using a separate Jupyter notebook that was called the "BUILD" script. The setup of the device is created in `setup_scope_prog` where the input taken is the device name. The device name is used to determine which programmer will be used (STM or ARM). Additionally, the `scope` (also known as what will be collected for the traces as a reference to an oscilloscope) is established. The default setup for `scope` is created, and then the number of timestep datapoints collected for each trace is set to 2500. These results are then returned to be used in the function `cw.program_target`, which is ChipWhisperer's programming function for the device. After this function is called, the ChipWhisperer device is programmed to run AES with a specified S-box.

5.3 Metrics on S-box Analysis

With a way to automatically program a ChipWhisperer device to run AES with a specified S-box, all The final data collection surrounds how to measure side-channel resistance.. As mentioned in Chapter 2, an adequate definition of security is difficult to formalize. Similarly, side-channel attacks are difficult to characterize formally. Despite this, Papagiannopoulos et al. [112] developed a "cheat sheet" of different metrics to measure the side-channel vulnerability of a given algorithm. Two significant metrics mentioned are the "Number of Traces" and

"T-Statistic" metrics. The latter of these two metrics is also known as "Detected Leakage."

Other metrics are specified by Papagiannopoulos et al. [112], but these two were the most stand-out from the other metrics. The other detailed metrics of "Signal to Noise Ratio", "Score", "Rank and Guessing Entropy", "Success Rate", "Success Rate of order o", and "Full Key Rank and Guessing Entropy" are different ways to reason about the "Number of Traces" metric, but additional assumptions such as knowing attack results are needed. The "Correlation" metric is used by CPA, so it is too specific to a certain test, and the other metrics of "Mutual Information", "Hypothetical Information", and "Perceived Information" involve hypothetical knowledge of the leakage distribution. Previously, this could be assumed to be normally-distributed, but previous work demonstrated that the power traces (or leakage) is not necessarily normally-distributed [8]. As a result, the "Number of Traces" and "Significant Leakage" metrics are the only two metrics that require the least amount of assumptions. DPA (Section 4.2) and CPA (Section 4.3) attacks can be detected using the "Number of Traces" metric. The "Detected Leakage" metric is a direct application of TVLA (Section 4.4) in that it uses TVLA as a metric to determine the leakage between two groups of power traces.

Using two analysis methods as opposed to one for determining the side-channel strength of an S-box used in AES limits the bias of one test over another. In particular, the "Number of Traces" metric measures the average difficulty to break the secret key with a particular S-box, but the result can be affected by whether a particular experimental run had traces that gathered more significant information. In contrast to "Number of Traces", the "Detected Leakage" metric will directly measure what leakage is significant, but this does not say whether a secret key can be extracted in a large number of traces or not. In fact, the "Detected Leakage" metric uses the t-test to determine whether there is a significant difference between groups. However, if significant leakage is detected, this result does not provide information on how to attack AES and recover the secret key. Analyzing side-channel vulnerability using a combination of these two metrics aims to balance out the bias of either metric used alone.

The following subsections detail the "Number of Traces" and "Leakage Detection" met-

rics. Both of these metrics involve knowing how to set initial values for certain functions and constants dependent on which side-channel attack (DPA, CPA, or TVLA) is used. As a result, a structure called "metric_values" was constructed to store each value. These constant values are displayed in Table 5.3. The Python dictionaries in python syntax are shown

Attack Method	METRIC_HIGH	attack_function	TOTAL_RUNS
DPA	4000	dpa_run	10
CPA	300	cpa_run	30
DPA	500	tvla_run	30

Table 5.3: Metric values for each side-channel attack method.

below in Figure 5.4.

```
dpa_metrics = {
      "METRIC_HIGH" : 4000,
      "attack_function" : dpa_run,
      "TOTAL_RUNS" : 10,
}
cpa_metrics = {
      "METRIC_HIGH" : 300,
      "attack_function" : cpa_run,
      "TOTAL_RUNS" : 30,
}
tvla_metrics = {
      "METRIC_HIGH" : 500,
      "attack_function" : tvla_run,
      "TOTAL_RUNS" : 30,
}
```

Figure 5.4: Metric Values Dictionaries

The METRIC_HIGH is a value used to establish the starting metric's high value for the number of traces. The attack_function is the function used by each attack method to attack, each of these being the traditional attack models for DPA, CPA, and TVLA. Finally, the TOTAL_RUNS is a variable that will demonstrate how many repetitions of each S-box's metric will be computed. The choice of TOTAL_RUNS having a value of 30 for CPA and TVLA is to ensure that the Central Limit Theorem can be more accurately applied to assume each trial is sampled randomly from the normal distribution. However, due to the much larger computational

cost of using DPA, only 10 repetitions could be performed in the interest of time.

5.3.1 Number of Traces Metric

The "Number of Traces" metric involves finding the minimum number of traces needed to recover the secret key a majority of the time. Because of this process' reliance on recovering the secret key, CPA and DPA are natural side-channel methods that can be measured with this metric as a result of both methods resulting in a guess for the secret key. This process allows for slight flexibility in terms of how to determine the minimum number of traces, and it also allows for different choices on what is determined on the majority. For the purposes of this experiment, this metric aimed to determine the minimum number of traces needed to recover the secret key 90% of the time, as demonstrated later in Algorithm 3.

The reliance on recovering the secret key makes it difficult to directly work with the AES modes CBC and CTR. This is because the ChipWhisperer has CPA and DPA attacks designed for AES in ECB mode. As a result, this experiment did not evaluate the "Number of Traces" metric for CBC or CTR because the methods were not implemented in the ChipWhisperer. DPA and CPA attacks on these modes to recover the secret key could be constructed, but the computational time cost of rerunning these modes of AES for "Number of Traces" was too expensive for this experiment.

Algorithm 3 shows the pseudocode used for the "Number of Traces" metric. This metric is performed using the function named `num_traces`. The function begins from lines by programming the device with the specified `platform`, whether it be the CW-Nano or CW-Lite. Additionally, `generate_c_files` (Algorithm 2) is called using the `sbox_name`. The rest of the algorithm determines the minimum number of traces needed by using the bisection method.

As a result, the initial lower bound for the bisection method, `lo` is set to 0, while the initial high bound `hi` is set to the `metric_values.METRIC_HIGH`. This value was defined in Table 5.3, dependent on the attack method of either CPA or DPA. The bisection method was implemented to determine the minimum number of traces because of its low time complexity of $\mathcal{O}(\log_2 n)$, where n is `metric_values.METRIC_HIGH`.

Algorithm 3 Number of Traces Metric

```
1: function num_traces(sbox_name, platform, metric_values)
2:     call generate_c_files(sbox_name)
3:     setup and program platform with AES on ECB mode
4:
5:     // Determining the initial bounds for the bisection method
6:     hi ← metric_values.METRIC_HIGH
7:     lo ← 0
8:
9:     // While loop is the bisection method
10:    while |hi − lo| > 1 do
11:        midpoint ← (hi + lo)//2
12:        counter ← 0
13:
14:        // For loop checks if breaks majority of time
15:        for i in 0, . . . , metric_values.TOTAL_RUNS do
16:            Gather midpoint number of traces of running AES with sbox_name
17:            key_guess ← key guess from metric_values.attack_function
18:
19:            // If there is a break, increment the number of breaks
20:            if key_guess == secret_key then
21:                counter ← counter + 1
22:            end if
23:        end for
24:        // If majority were breaks, look if can break using less traces
25:        if counter ≥ 0.9 ∗ metric_values.TOTAL_RUNS then
26:            hi ← midpoint
27:            // Otherwise, look if we can break using more traces
28:        else
29:            lo ← midpoint
30:        end if
31:    end while
32:
33:    Print hi to file with sbox_name
34: end function
```

The while loop from lines 10 through 31 implements the bisection method. Each iteration of the while loop computes the midpoint between the high and low bound, rounded down by integer division. A counter variable is also initialized to account for the number of successful power-based side-channel attacks on AES using the given S-box.

The for loop from lines 15 through 23 in Algorithm 3 checks to determine if the 90%

threshold is maintained for breaking AES a majority of the time. The total number of iterations performed for this loop is `metric_values.TOTAL_RUNS`. As mentioned earlier, these values are chosen to ensure a better random sampling of the results to gather an accurate measurement for the "Number of Traces" metric. Each of these iterations compute a guess for the secret key using either DPA or CPA, which is determined from `metric_values.attack_function`. If this key is the secret key, then the counter is incremented.

After the `metric_values.TOTAL_RUNS`-th iteration, the counter is compared to 90% of `metric_values.TOTAL_RUNS`. If the counter is greater than or equal to this value, then there was a successful break a majority of the time. In this case, the new high bound would be the midpoint to see if the minimum number of traces could be smaller. On the other hand, if the counter is less than 90% of `metric_values.TOTAL_RUNS`, then the low bound is set to the midpoint to check if more traces are needed to recover the secret key a majority of the time.

Once the high bound and the low bound have a distance of less than or equal to one, the while loop ends. This results in a final value, `hi`, that will be the result for the minimum number of traces needed to successfully break the secret key using either CPA or DPA more than 90% of the time. The final result is print to a text file that details the results of the minimum number of traces needed to break AES with the given `sbox_name`. This function was called for each of the sixty-six S-boxes in Table 4.1. The metric results were computed using both CW-Nano each using DPA and CPA totaling four total metric results.

One of the major strengths of this metric is that it provides an estimate of the time cost required to recover the secret key for each S-box used in AES. Additionally, this method is simple to implement because it involves performing repeated CPA or DPA attacks `metric_values.TOTAL_RUNS` number of times and performing a simple check to see whether the secret key was recovered or not. Additionally, this process can be parallelized using many identical devices each performing the same operations, and by random sampling, the results would be similar. For example, rather than using `metric_values.TOTAL_RUNS` iterations on a single ChipWhisperer device, `metric_values.TOTAL_RUNS` identical ChipWhisperer devices could each run a single time, and the next iterations of the number of traces in the bisection method would be used based on the number of devices that recovered the secret key.

Despite its strengths, this method has some limitations that make it difficult to use as a sole metric. In particular, this method is based on probabilistic results in a "majority of the time". If the experiment is ran again, the values for each minimum number of traces could shift slightly based on whether the secret key was recovered exactly each time. A single byte of the secret key could be incorrect on a sequential run of the experiment, and that would determine the secret key as not recovered, even if it was recovered as a majority. Additionally, the results for this metric are all positive integers as a count of traces. This results in discrete results, there could not be a minimum number of traces needed to break being a non-integer rational number. As a result, the results of this metric could be seen as categorical results rather than numeric values as a range, and this idea could affect how each result should be interpreted. Finally, when using CPA as an attack method, the power traces gathered are assumed to be from a normal distribution which is not always the case [8].

5.3.2 Detected Leakage Metric

This metric involves gathering a fixed number of power traces from AES with a given S-box and separating the values into two groups. The ChipWhisperer already has a methodology for this, which is shown as pseudocode in Algorithm 4. The TVLA process performed in `tvla_gather_n_traces` is also shown in Section 4.4 by Figure 4.5 The fixed number of power traces was set to $N =$ `tvla_metric_values.METRIC_HIGH` $= 500$ for each group, making 1000 power traces total gathered. This number was chosen to be large enough to have more traces to compare the amount of leakage so that the values would be easy to visualize. In particular, N plaintext values given to AES are a fixed plaintext, and the other N are randomly generated plaintext inputs for AES. As a result, the two groups will contain N elements each, one with the power traces of a fixed plaintext and the other with the power traces of random plaintext.

The goal of creating these two groups is to determine if there is a significant difference between a random plaintext and a fixed plaintext's traces when recovered from an algorithm. If one of the 2500 timesteps (defined in the programming of the device) in a trace has a significant difference from the random plaintext, then it is determined that leakage occurred.

Algorithm 4 TVLA Gather N Traces

```
1: // This function is based on a template provided by ChipWhisperer
2: function tvla_gather_n_traces(Num_traces)
3:     // Initialize all key, text pairs with ChipWhisperer software
4:     ktp ← cw.ktp.TVLATest.init(Num_traces)
5:
6:     fixed_group ← [ ]
7:     random_group ← [ ]
8:     fixed_text ← [0xda, 0x39, 0xa3, 0xee,
9:                   0x5e, 0x6b, 0x4b, 0x0d,
10:                  0x32, 0x55, 0xbf, 0xef,
11:                  0x95, 0x60, 0x18, 0x90]
12:
13:    for i in 0, ..., (2 * Num_traces) do
14:        key ← key from ktp
15:        text ← text from ktp
16:        trace ← captured power trace with key and text
17:
18:        // Determine whether the text was the fixed text
19:        if text == fixed_text then
20:            append trace to fixed_group
21:        else
22:            append trace to random_group
23:        end if
24:        ktp = ktp.next()
25:    end for
26:    return (fixed_group, random_group)
27: end function
```

Welch's t-test [107] is tested on the two groups to determine if there is a statistically significant difference between the two groups. In particular, sciPy's `ttest_ind` function [167] can compute this value. Between both groups, this test will be comparing the difference between two $N \times 2500$ matrices containing the power consumption of N traces at 2500 timesteps. Using `ttest_ind` will result in a list of 2500 $t-$statistics, each representing a statistic to determine the likelihood that leakage occurred at any of the timesteps. If a $t-$statistic has its absolute value greater than 4.5, this means that significant leakage is detected at that timestep [106]. We will denote N_t as the number of t-statistics out of the 2500 calculated from `ttest_ind` result that has its absolute value greater than 4.5. A ratio computed as

$r = \frac{N_t}{2500}$ is determine as the ratio of leakage.

Algorithm 5 demonstrates the pseudocode used to calculate the average leakage ratio needed for this metric over `tvla_metric_values.TOTAL_RUNS` = 30 iterations. The beginning of this algorithm (lines 2 through 6) involves first modifying Tiny-AES-C to work with `sbox_name` sbox and programming the `platform` device to run AES in `aes_mode` mode. An array `rations` is initialized as an empty array to eventually store each iteration's r value.

The for loop from lines 7 through 16 perform the actual leakage detection a total of `num_runs` times. First, the two groups (random and fixed) of power traces are computed using the `TVLA_Gather_N_Traces` function (Algorithm 4). The number of traces used in this function was $N =$ `tvla_metric_values.METRIC_HIGH`. A list of 2500 $t-$statistics are calculated and stored in `tvalues` using the function `ttest_ind` on the fixed and random groups of power traces. This process is actually a one-line implementation of the `tvla_run` function.

Finally, each ratio value, r is computed and stored in `ratios`. This array will end up with a total of `tvla_metric_values.TOTAL_RUNS` r ratios of leakage. The return of this function will be the average of all the r values in `ratios`. This result was then wrote to a text file along with the S-box name that was used in AES for this result. This `tvla` function was called for each S-box to compute the "Detected Leakage" metric value for each S-box.

Because this test involves gathering power traces and not recovering a secret key, each mode of AES can be used for this metric. The three different modes with two ChipWhisperer devices resulted in six metric results in total. An example of the results file that used AES in CBC mode with the CW-Nano device would be named `avg_leaks_cwnano_CBC.txt`.

The biggest strength of this approach is in its ease of collecting the necessary information. All this method requires is collecting the power trace information from the cryptographic device. In particular, this adds to the flexibility of testing the different modes of AES because recovering a secret key is not necessary for this metric. Additionally, calculating the t-test statistic runs much faster as it does not require exhaustively guessing through all possible bytes for a subkey like DPA and CPA attacks do in the "Number of Traces" metric. While the "Number of Traces" metric relies on probability, the "Detected Leakage" metric uses statistical hypothesis testing to have significant evidence for leakage detection. As a

Algorithm 5 Average Leakage Detection Metric Function (TVLA)

```
1: function tvla(sbox_name, platform, tvla_metric_values, aes_mode)
2:     call generate_c_files(sbox_name)
3:     setup and program platform with AES on aes_mode mode
4:
5:     num_runs ← tvla_metric_values.TOTAL_RUNS
6:     // Array to store each ratio for 30 runs
7:     ratios[num_runs]
8:     for i in 0, . . . , num_runs do
9:         // Get two groups of power traces for 500 traces in each group
10:        N ← tvla_metric_values.METRIC_HIGH
11:        (fixed_t, random_t) ← tvla_gather_n_traces(N)
12:
13:        // Compute the t-value for each timestep (tvla_run function)
14:        tvalues ← tvla_metric_values.attack_function(fixed_t, random_t)
15:
16:        // Determine the ratio of leaked timesteps
17:        append (Number of values in tvalues ≥ 4.5)/2500 to ratios
18:    end for
19:
20:    return The average of ratios
21: end function
```

result, the statistical strength of result is greater with this metric. The results of this metric are also continuous in that the resulting average leakage ratio would be any possible rational value in the interval [0,1].

Despite the statistical strength being greater in the "Detected Leakage" metric compared to the "Number of Traces" metric, the empirical strength is lessened. While it is useful to quantify the leakage, the "Detected Leakage" metric does not provide a measure of how easy it would be to recover the secret key used in AES with the given S-box. Because the value of the average leakage ratio is in the empirically small interval of [0,1], the averaging of each trace could obscure some small differences in leakage by nature of the computation of the arithmetic mean.

5.4 Analyzing S-box Results

With each test result gathered, the final analysis compiled all of the data into a single data structure. A DataFrame from the Pandas Python library [168,169] was used to include data from each S-box and its results. This dataframe had sixty-six rows (one for each S-box) and twenty-two columns. The first ten columns contain the names of each S-box and their measured properties as in Section 5.1.1. The remaining columns are each results for each metric. Pandas has a function named `to_csv` that converts a DataFrame to a `.csv` file named `sboxes_results.csv`.

This `.csv` file was used by the loaded into an R program [170] for statistical analysis. R would read the `sboxes_results.csv` file into a DataFrame in R. This data structure in R allowed for ease of usage in methods such as regression analysis, plotting of the data, and making new DataFrames from the original DataFrame.

Regression analysis was performed to predict the results from each metric based on the properties of the S-boxes. The goal of this analysis was to see what properties of S-boxes have significant effects on the side-channel metric result. By doing this analysis on each result, some properties had significant effects on these results. As a result, the most significant properties were determined from the regression analysis results.

The process to determine the significant results involved creating a base formula. Each of the S-box properties from Section 5.1.1 were used as the predictors for this formula for each metric, and this model was represented in R syntax as

```
formula <- result ~ (nonlinearity + linearity + differential_probability +
        boomerang_uniformity + diff_branch + linear_branch +
        linear_probability + bic + sac)^2.
```

This formula involved using the `^2` around each of the S-box properties which would create a base formula containing each linear S-box property along with each pairwise interaction for a total of 54 predictor terms. Linear terms are the values such as `nonlinearity` or `differential_probability`, which both are single terms. Interaction terms are any of the pairwise interactions of the S-box properties such as `diff_branch * linear_branch`. From

this base formula, a linear regression model was constructed using the linear model function (`lm`).

The `summary` function in R determines what properties have a "goodness of fit" for each result. With each of these properties comes an associated p-value in terms of its significant effect on predicting the result. The higher the p-value, the less probability that a given term predicts the result. In particular, a significance level of 5% (0.05) was chosen to determine if a term had a significant effect. This level of significance was chosen to maintain consistency with the standard choice of significance level for regression analysis. If a p-value was less than the significance level of 0.05, then the term associated with that p-value was determined to be significant.

The process of creating the best linear regression model by modifying the original base formula is given in Algorithm 6. This function aimed to create the best model by having each term in the final model have p-values less than 0.05. However, to add to the confidence of these results, the p-values were used to construct 95% Confidence Intervals and see if they contain zero for each term. This involved taking the base model and removing a term one at a time in a step-wise function. In particular, lines 15 through 17 of Algorithm 6 create the new linear model without the term with the largest width confidence interval. This function was called recursively until the width of the confidence interval did not contain zero (there was a significant slope) as shown in lines 11 through 13.This function was called for each metric as a `dependent_var`, and the original `linear_model` was the model using the formula containing all linear and interaction terms.

After the regression results had been determined for each metric, it remained to analyze them. A count of each result's significant predictors were represented in a barplot to show how many metrics have these terms as significant predictors using the library `ggplot2` [171]. That is, if a property had a large value in the barplot, then that means that property is used more often in predicting the results of each metric. The barplot will determine what S-box property is the most significant predictor of side-channel resistance by counting its frequency as a significant term in each regression model.

For each metric's result, the fitted regression model returned from `regression_step` was

Algorithm 6 Regression Step Function

```
function regression_step(linear_model, dependent_var)
    // Get terms and their p-values
    summ_fit ← Summary of linear_model
    terms ← Attributes of linear_model terms
    pvals ← Filtered pvals based on non_na

    // Calculate 95% Confidence Interval, find largest Interval
    max_ci ← Index of maximum Confidence Interval width for each term

    // If largest width confidence interval does not contain zero, return
    if !(max_ci.lower_bound <= 0 && max_ci.upper_bound >= 0) then
        return linear_model
    end if
    // Update formula without the largest p-value term
    remove_term ← term with largest confidence interval
    new_formula ← formula without remove_term, keeping dependent_var
    new_lm ← Update linear_model with new_formula

    // Recursively call function with updated regression model
    return regression_step(new_lm, dependent_var)
end function
```

represented as a linear function based on the properties of the S-box. In addition, by the step-wise construction of the regression model, each term for each model was statistically significant in predicting the metric's result.

5.4.1 A Short Regression on Regression Models

This section gives an example of the regression methodology using the Penguins dataset modeldata.

As an example of a linear regression formula, we will use the Penguins dataset in Equation 5.1. In particular, this example predicts the body mass of a penguin in grams based on the bill length and bill depth measurements of a penguin, each in millimeters.

$$\begin{aligned} \texttt{body\_mass\_g} = &-25718.181 + 718.853 * \texttt{bill\_length\_mm} \\ &+ 1493.496 * \texttt{bill\_depth\_mm} \\ &- 36.317 * \texttt{bill\_length\_mm*bill\_depth\_mm} \end{aligned} \tag{5.1}$$

The linear terms of Equation 5.1 are the terms of bill length and bill depth. The interaction term is present in the model as `bill_length_mm*bill_depth_mm`. This regression equation's coefficient values demonstrate how changing a term yields a change in the predicted body mass. For example, this means that as `bill_length_mm` increases by one, the predicted body mass of the penguin increases by 718.853 grams while other terms in the regression equation remain constant. The interaction term is a bit more complicated, it demonstrates that as both bill length and bill depth increase together, the body mass of a penguin decreases.

An easy way to visualize the effect of the regression model was made by plotting the predicted values for a metric against the actual values. These constructed plots are denoted as Predicted values vs. Actual values plots. This was done by computing the predicted values for each metric using the equation from the regression equation and plotting the result on the x axis. The y axis was the actual values for the predicted metric, and an identity line ($y = x$) was drawn on the plot. If the trend of the datapoints of this plot are close to the identity line, the overall regression model is shown to be a good predictor for the metric.

An example of this using the Penguin dataset is shown in Figure 5.5. This figure demonstrates how a constructed graph using a penguin's bill length and bill depth can predict the body mass of a penguin. The datapoints follow the trend of the identity line, so this model was a good fit to predict the body mass of a penguin.

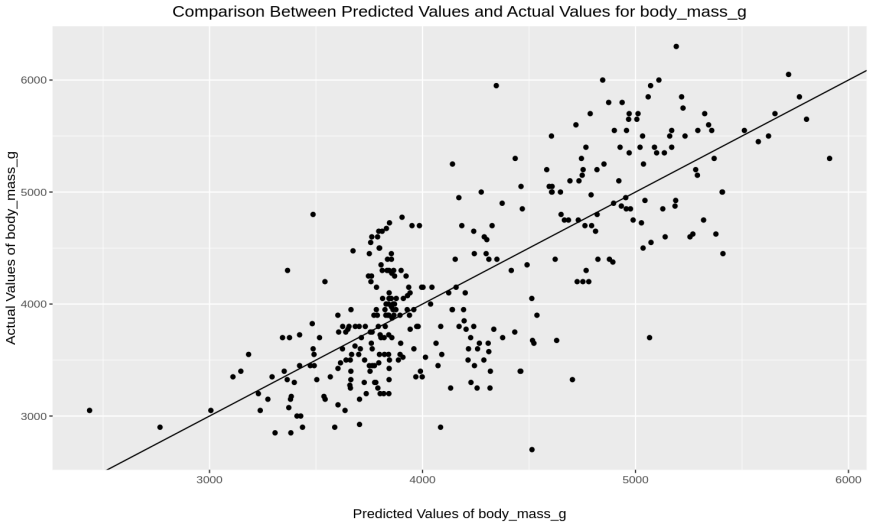


Figure 5.5: Example of Predicted Values vs. Actual Values Plot

These regression formulae were also represented visually using plots from the car package for R [172]. This package would plot Component and Residual (C+R) plots for each term of a linear model. For linear predictor terms (no interactions), a two-dimensional C+R plot demonstrates the effect of a term on the x axis of the plot for a predicted variable on the y axis. If a C+R plot does not have a horizontal line, that indicates the term on the x axis is a significant term for the model. In fact, the dashed line's slope on the C+R plot is the coefficient associated with the term in the linear regression model. An example of a C+R plot for the Penguin dataset is shown in Figure 5.6. Because each plot shows the dashed line was not horizontal, the terms of bill length and bill depth are statistically significant in predicting the body mass of a penguin.

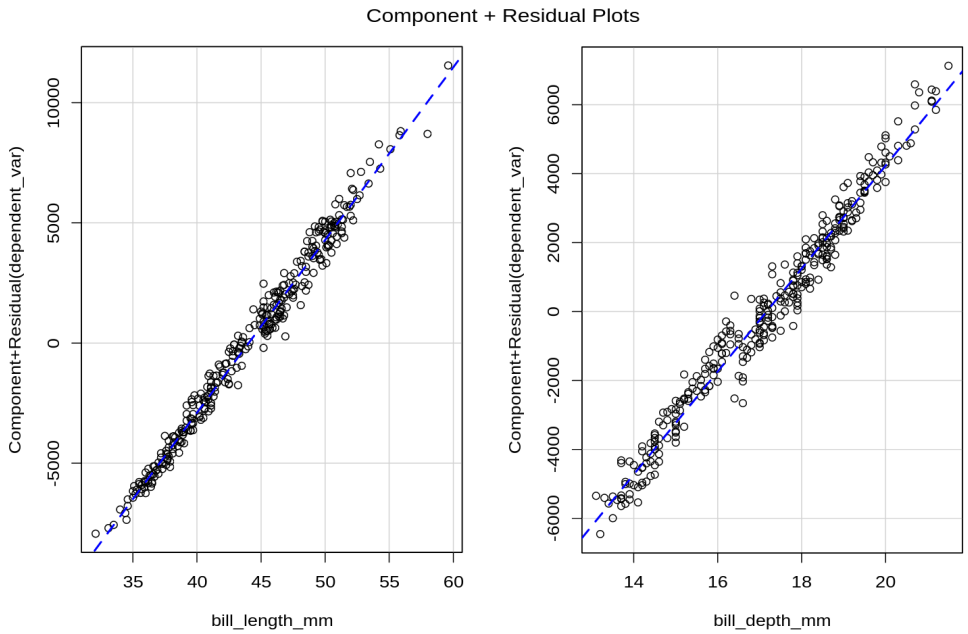
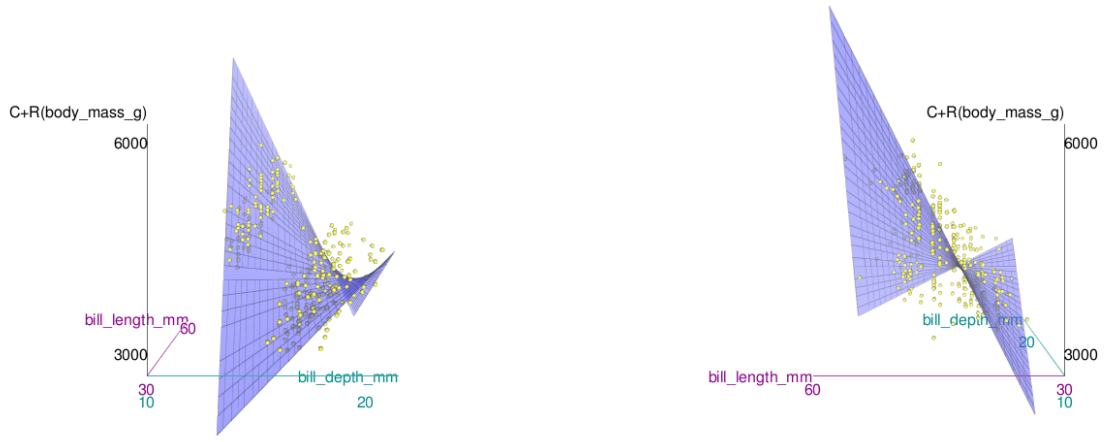


Figure 5.6: 2D Component and Residual Plot Example.

Finally, interaction terms have C+R plots also, however their results needed to be represented in three dimensions to demonstrate their effect. To visualize this, a three-dimensional C+R plot was constructed to visualize the effect of interaction terms in regression prediction models. The effect of each variable on the model as an interaction is represented as a plane rather than a line. The x and y axes were the two interacting variables, and the z axis represented the predicted variables. Two angles of the three-dimensional plots, one focused

on the x-axis and one on the y-axis, had snapshots taken for each of the interaction terms for each predictive model to visualize the three-dimensional object. An example of the effect of an interaction term of two linear terms for the Penguin dataset is shown in Figure 5.7.



(a) x-axis view. (b) y-axis view.

Figure 5.7: 3D Component and Residual Plot Example.

5.4.2 Applying the Methods to S-box Analysis

An "ideal" S-box was hypothesized using the results for each of the ten metrics. This hypothetical S-box had its properties constructed to ensure that its "Number of Traces" metric should be predicted to be high and its "Detected Leakage" metric would be low. An S-box constructed in this way would be the most side-channel-resistant S-box for AES.

Trying to construct such an S-box is a computationally intractable problem. However, given a specific S-box that fulfills some of the properties indicates that S-box would have a higher level of resistance to side-channel attacks when used in AES.

Chapter 6

Results and Conclusions

The first acquired results were each S-box's property values, and the first ten S-boxes (alphabetically) are provided in Table 6.1. The full table of values for each of the sixty-six S-boxes are listed in Appendix 6.4 (Table A.1). The only significant outliers in results are S-boxes that did not have inverses, namely the S-boxes CMEA, Iraqi, and Picaro. Because the S-box property boomerang uniformity requires an inverse S-box to exist, the property value for these S-boxes had boomerang uniformity values of -1. Note that his is visible in Table 6.1, where CMEA's boomerang uniformity has a value of -1.

S-box Name	Nonlinearity	Linear Probability	Differential Probability	Boomerang Uniformity	Differential Branch	Linear Branch	Linearity	BIC	SAC
AES	112	0.06250	0.01562	6	2	2	32	0.13412	0.50488
ARIA_s2	112	0.06250	0.01562	6	2	2	32	0.13412	0.50293
Anubis	94	0.13281	0.03125	18	2	2	68	0.25490	0.50024
BelT	102	0.10156	0.03125	20	2	2	52	0.16841	0.49951
CLEFIA_S0	100	0.10938	0.03906	32	3	3	56	0.33333	0.53906
CLEFIA_S1	112	0.06250	0.01562	6	2	2	32	0.13170	0.49561
CMEA	96	0.12500	0.04688	-1	1	2	64	0.29659	0.51611
CSA	94	0.13281	0.04688	18	2	2	68	0.29261	0.49146
CSS	0	0.50000	0.50000	256	2	2	256	1.00000	0.28125
CS_cipher	96	0.12500	0.06250	256	2	2	64	1.00000	0.43506

Table 6.1: First Ten S-boxes' property values.

The total runtime for computing a metric's result for all sixty-six S-boxes is shown in Table 6.2. Note that an iteration depends on the metric that is being measured. For "Number of Traces", the iteration is the time it takes after programming the device to make a guess on the secret key. For "Leakage Detection", the iteration is the time it takes to compute

one of the leakage ratios out of the total thirty runs. The "Number of Traces" metric has its total time approximated as an upper-bound runtime to have $\log_2$ (high bound) to account for each iteration of the bisection method. The high bound value is determined from Table 5.3. That is, the total runtime for "Number of Traces" metric results ($time_{numtraces}$) and for the "Leakage Detection" metric results ($time_{leakage}$) are provided in Equation 6.1.

$$time_{numtraces} = \sum_{i=0}^{\log_2 (\text{high bound})} it\_time/(2^i) * 66 * \texttt{TOTAL\_RUNS}$$
$$time_{leakage} = it\_time * 66 * \texttt{TOTAL\_RUNS} \tag{6.1}$$

The value `TOTAL_RUNS` can be evaluated depending on the attack method from Table 5.3. The number sixty-six is used because each of the sixty-six S-boxes is tested on each metric.

Metric Result	Mode/ Attack	Device	Iteration Time (seconds)	Total Time (hours)
"Number of Traces" metric	DPA	CW-Lite	130.47	47.83
	DPA	CW-Nano	132.00	48.39
	CPA	CW-Lite	8.56	9.40
	CPA	CW-Nano	10.28	11.29
"Detected Leakage" metric	ECB	CW-Lite	30.77	16.92
	ECB	CW-Nano	54.46	29.95
	CBC	CW-Lite	30.79	16.93
	CBC	CW-Nano	55.05	30.28
	CTR	CW-Lite	30.38	16.71
	CTR	CW-Nano	55.02	30.26

Table 6.2: Runtime values for each metric.

6.1 Metric Results and Density of Results

The total results for each metric for the first ten S-boxes alphabetically is shown in Table 6.3. The full results are shown in Appendix 6.4 (Table A.2). The full results are kept to an appendix because the individual results' values are not important as the overall results.

One noticeable result was that using the CW-Nano with AES in the CTR mode did not have any leakage detected under the "Leakage Detection" metric. In fact, Table 6.3

	Metric Results									
	"Number of Traces" metric				"Detected Leakage" metric					
	DPA		CPA		ECB		CBC		CTR	
S-box Name	CW-Lite	CW-Nano	CW-Lite	CW-Nano	CW-Lite	CW-Nano	CW-Lite	CW-Nano	CW-Lite	CW-Nano
AES	696	1714	23	48	0.55349	0.49239	0.10077	0.00815	0.04563	0.0
ARIA_s2	620	> 2000	23	50	0.51231	0.52465	0.10041	0.00789	0.04456	0.0
Anubis	> 1000	> 2000	28	41	0.54643	0.52184	0.09283	0.00771	0.04245	0.0
BelT	> 1000	> 2000	25	50	0.53611	0.48809	0.09919	0.00787	0.04585	0.0
CLEFIA_S0	585	1684	24	46	0.54451	0.50700	0.10069	0.00784	0.04200	0.0
CLEFIA_S1	> 1000	1914	24	49	0.53212	0.52824	0.10268	0.00797	0.04504	0.0
CMEA	> 1000	> 2000	26	51	0.51932	0.48633	0.06465	0.00796	0.04647	0.0
CSA	> 1000	1989	25	47	0.57307	0.47908	0.01131	0.00829	0.04145	0.0
CSS	> 1000	> 2000	25	47	0.50945	0.49605	0.03713	0.00831	0.04212	0.0
CS_cipher	> 1000	> 2000	28	37	0.59176	0.48869	0.06499	0.00765	0.04204	0.0

Table 6.3: First Ten S-boxes' results for each metric result.

demonstrates that entire column has all zeroes. Because no noticeable result was found, this metric result was omitted from future analysis. Additionally, the "Number of Traces" metrics calculated using DPA had some results that failed to break more than 90% of the time within the initial upper bound, and these results are represented by a "> 1000" or "> 2000" traces value when using the CW-Lite and CW-Nano; respectively. From Table 6.3, some of the S-boxes are shown to have no result demonstrated for DPA on both devices. AES in ECB mode also leaks significantly more information than CBC or CTR modes. In particular, the ECB mode has a leakage ratio about five times larger than either mode for both ChipWhisperer devices.

Table 6.3 also shows general relationships between the metric results of each ChipWhisperer device. In particular, when using CPA for "Number of Traces" results, the CW-Lite used about half as many traces as the CW-Nano to recover the secret key of AES using a provided S-box. Additionally, the CW-Lite is able to detect about 4% more leakage than the CW-Nano in each AES mode. These relationships support the idea that while the power-based side-channel analysis device does affect the measurement of side-channel resistance of AES, it is only a linear difference.

The lack of consistent key breaks using DPA can be explained by ghost peaks. These ghost peaks are seen in Figure 4.2 from Section 4.2 where incorrect guesses for the subkey look similar to the correct guess for the subkey. These peaks would make an incorrect guess for a single subkey of the secret key, and this incorrect guess would result in a failed break by the "Number of Traces" metric. As a result, 39 S-boxes did not have DPA "Number

of Traces" metric results with the CW-Nano, and 29 S-boxes did not have a metric value with the CW-Lite device. Despite these failures, a similar distribution can be seen between both results in Figure 6.1. Both plots have similar shape, although the CW-Lite's x axis is scaled to be half of that of the CW-Nano. Figure 6.1 demonstrates that while DPA can determine the minimum number of traces needed to break the key more than 90% of the time, the weaknesses of DPA show that many S-boxes cannot be accurately measured using this side-channel attack.

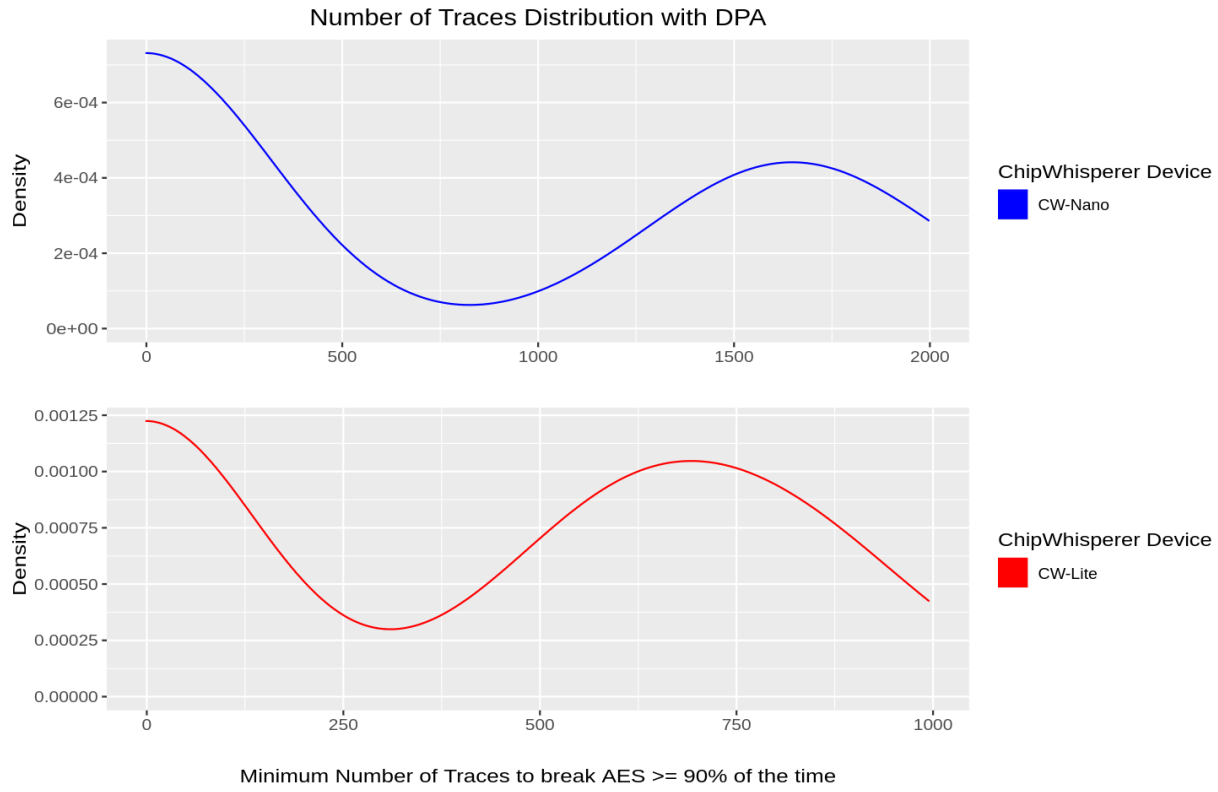


Figure 6.1: "Number of Traces" metric density plots with DPA.

On the other hand, Figure 6.2 demonstrates the density plots for the "Number of Traces" metric with CPA on both CW-Nano and the CW-Lite. No failures to break ocurred using this metric, and this is highlighted by the lack of each density plot having a spike of elements with values of -1. the CW-Lite also had a narrower range of values, only having the number of traces ranging from 21 to 30, and about 25% of S-boxes used in AES would have their "Number of Traces" value be 25. The CW-Nano's results are a bit more distributed, but

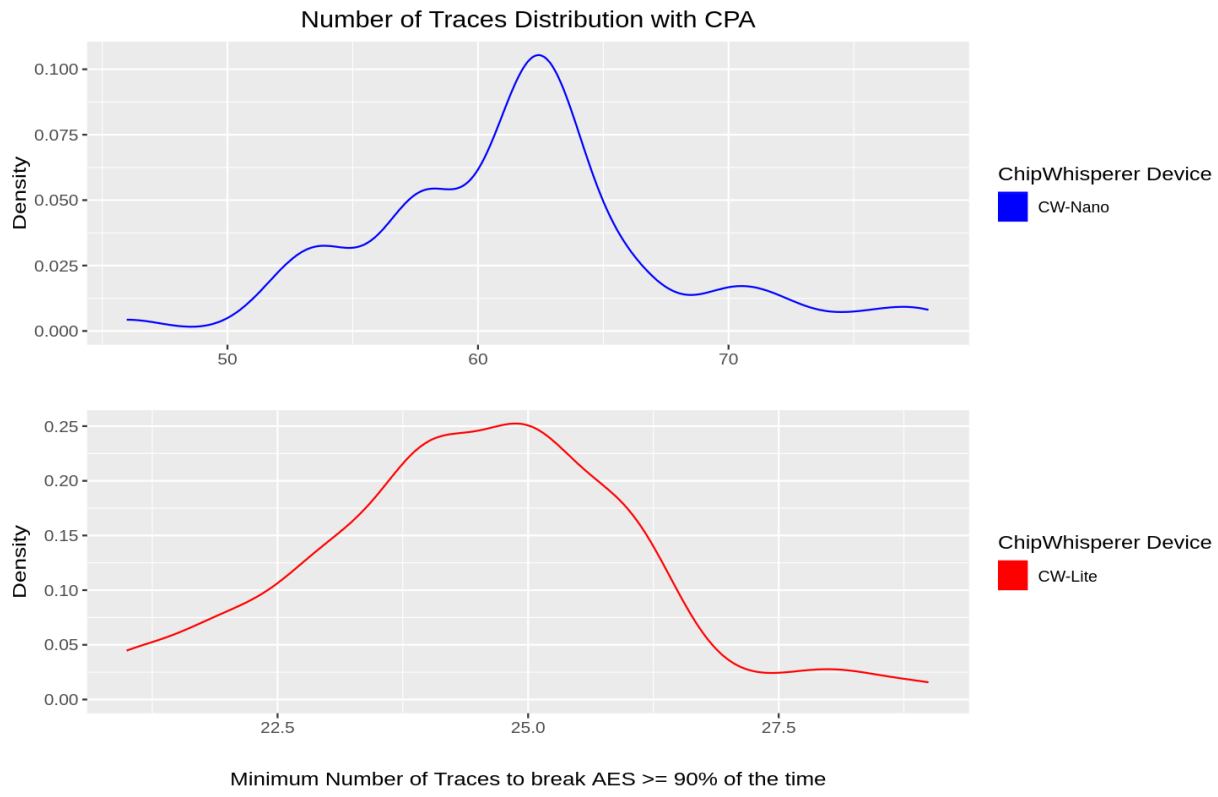


Figure 6.2: "Number of Traces" metric density plots with CPA.

most of the "Number of Traces" values on this device are around 61 to 63 traces.

Although both DPA and CPA are popular methods of side-channel attack, CPA has demonstrated it has more consistent results when it comes to measuring the minimum number of traces needed because of the lack of ghost peaks on incorrect subkey guesses. Because of the consistency of CPA's results, the "Number of Traces" metric will be primarily considered from CPA, but DPA will still be accounted for.

The density plots for each metric result using the "Detected Leakage" metric are shown in Figure 6.3. The results here demonstrate the similarities of the distributions across each ChipWhisperer Device, aside from the CW-Nano when detecting leakage of AES using CTR mode. AES in ECB mode for both devices seem to have similar distributions, supporting the similar results for each device. The CBC mode for both devices seemed to have similar density distributions, but the CW-Lite had large right-skew due to outlier results. Despite the CTR mode not having any detected leakage with the CW-Nano, the CW-Lite's density

plot has a similar shape (and right-skewness) as the plot above. In particular, there are similar density values between the CBC and CTR modes of AES.

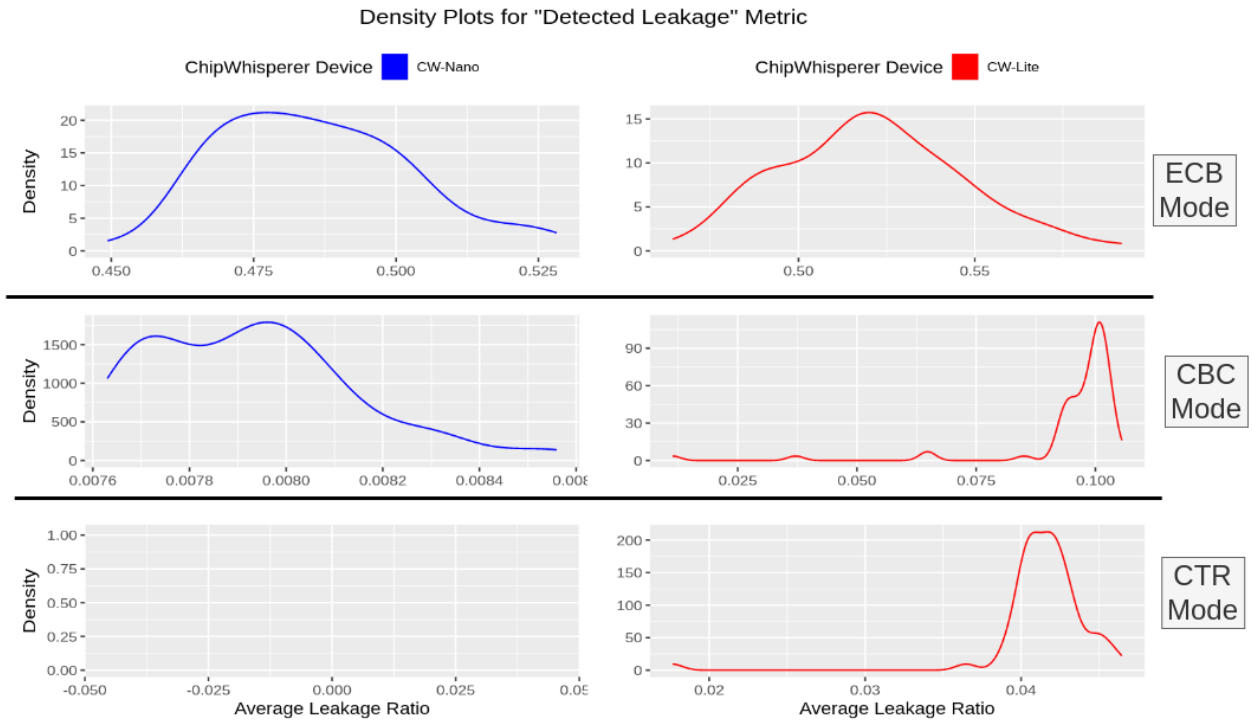


Figure 6.3: "Detected Leakage" metric density plots.

6.2 Regression Analysis

Each metric result had a regression formula fit to predict each result based on the property values of a given S-box. However, the metric results for "Detected Leakage" on the AES mode CTR for both the CW-Nano and CW-Lite were not able to construct linear models with significant predictors. The lack of regression models for these metric results can be explained by the poor quality of results for the "Detected Leakage" metric under the CTR mode. The CW-Nano results for this metric result were all zero, so fitting a regression model would not work. The CW-Lite had all of its values within .05 of zero, so there was not a lot of varied data.

In an effort to concisely label each term, Table 6.4 demonstrates what each abbreviated

version means for a property in the regression equation. For example, `nl` is an abbreviation for nonlinearity.

Abbreviation	Property Name
`bic`	Bit Independence Criterion
`bu`	Boomerang Uniformity
`db`	Differential Branch Number
`dp`	Differential Probability
`lb`	Linear Branch Number
`lp`	Linear Probability
`l`	Linearity
`nl`	Nonlinearity
`sac`	Strict Avalanche Criterion

Table 6.4: Abbreviation of S-box properties for regression formulae.

Additionally, the constructed regression equations will not have certain terms because of their linear relationship to other terms. This lack of inclusion is denoted as a singularity, and not including this means that another term in the formula can represent it. In particular, the linearity S-box property is linearly related to the nonlinearity S-box property. This means that linearity can be explained by nonlinearity in any regression formula, so linearity is removed as a possible term. Some other interaction terms were also determined as linearly related, and they are listed below.

```
Removed terms:
db*lb, db*sac, lb*sac, lb*bic, lp*db, lp*lb, lb*bu, dp*lb, nl*lb
```

In an effort to contain space in the main body of this thesis, only two of the regression formulae are presented in Equation 6.2. The $nt_{DPA,Lite}$ is the regression equation for predicting the "Number of Traces" using DPA on the CW-Lite, and $dt_{CBC,Nano}$ is the regression equation for predicting the detected leakage of AES using CBC mode on the CW-Nano. All of the regression formulae for each metric result are given in Appendix B. The large difference in predictor terms demonstrated how varied these regression equations could be in the number of terms each contains.

$$
\begin{aligned}
nt_{DPA,Lite} = &-4207.5 + 172543.649 * \texttt{dp} \\
&- 0.137 * \texttt{nl*bu} \\
&+ 18.943 * \texttt{nl*db} \\
&- 928675.838 * \texttt{dp*lp} \\
&+ 112486.252 * \texttt{lp*bic} \\
&+ 39413.24 * \texttt{lp*sac} \\
&+ 283.252 * \texttt{dp*bu} \\
&- 40066.13 * \texttt{dp*db} \\
&- 29733.291 * \texttt{bic*sac}
\end{aligned}
\tag{6.2}
$$

$$
dt_{CBC,Nano} = 0.008 + 0.000001538 * \texttt{nl*db}
$$

In particular, looking at Equation 6.2, we can see that predicting the detected leakage of AES using CBC mode decreases as nonlinearity and the differential branch number increase. This would mean that larger differential branch number and nonlinearity in an S-box lead to less leakage detected. The "Number of Traces" metric for DPA with the CW-Lite increases with nonlinearity and differential branch number increase. Both of these results demonstrate that a larger nonlinearity and differential branch number in an S-box both increase side-channel resistance by increasing the number of traces needed to recover a secret key and reducing detected power leakage.

6.2.1 Visual Supplements for Regression Equations

Although all of the regression formulae are computed to have each term be significant in predicting the respective metric result, the p-value doesn't visually demonstrate the impact of how well the results fit. To account for this, the Actual Values vs. Predicted Values for all eight regression formulae are presented in Figure 6.4. This demonstrates how the overall model fits well according to each regression equation. As we can see, the line of fit for each of these values demonstrate that the prediction by the regression equation performs fairly well. These visualizations supplement the knowledge that the regression equation constructed

for each metric was statistically-backed in predicting the metric result based on an S-box's properties.

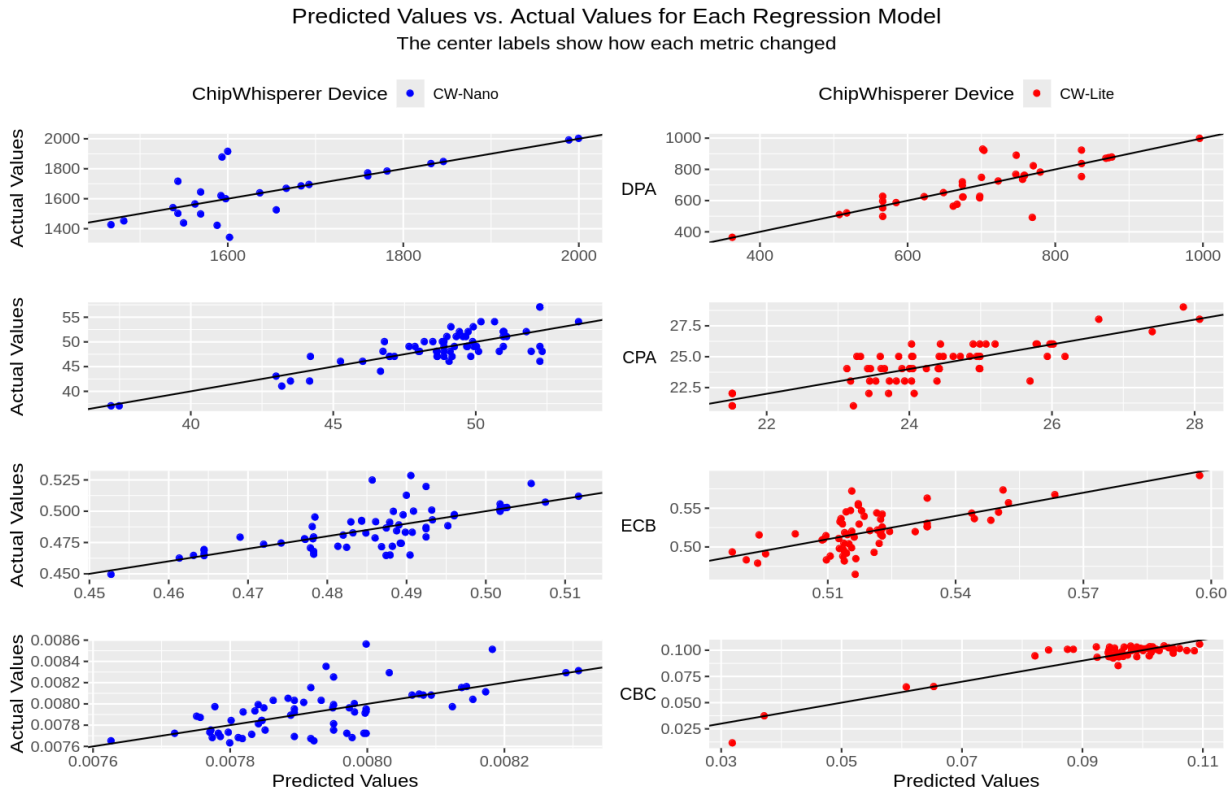
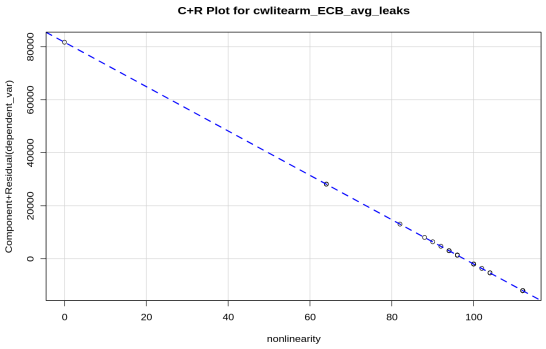


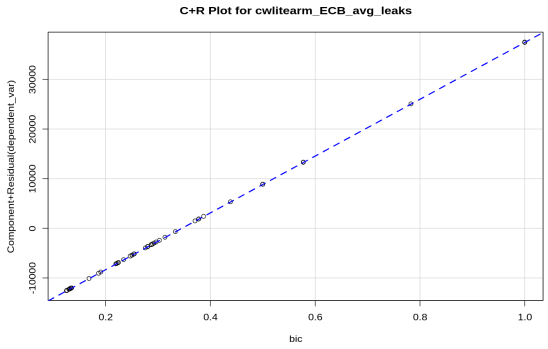
Figure 6.4: Predicted Values vs. Actual Values for each Regression Equation

Each term's significance can be demonstrated by examining each term's C+R plots. A linear term in a regression equation demonstrates a significant effect in prediction of the metric if its respective C+R plot's trend line is not a straight, horizontal line. All of the linear terms' C+R plots in each regression equation were observed to have nonzero slopes. Figure 6.5 demonstrates some of the C+R plots from some of the regression equations. All other C+R plots for other regression formulae and their terms looked similar to these. As a result, effects of each linear term are supplemented as being sufficient by their C+R plot.

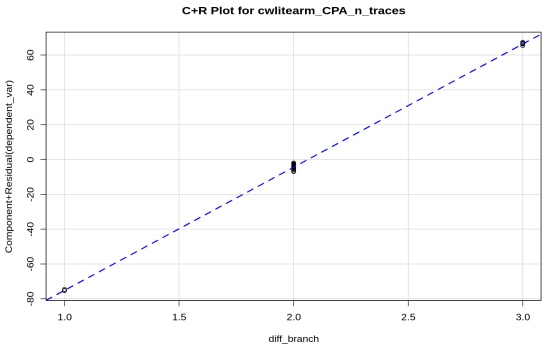
The significance of each interaction term's significance was also observed using 3D C+R plots. These plots helped demonstrate the effect interaction terms had on each other in predicting a metric. A sampling of all the 3D C+R plots seen from the x and y axis angles are given in Figure 6.6. Other plots were constructed to verify the significance of



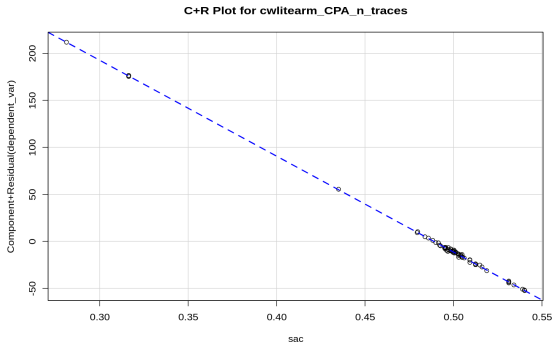
(a) C+R plot for nonlinearity on predicting "Detected Leakage".



(b) C+R plot for BIC on predicting "Detected Leakage".



(c) C+R plot for differential branch number on predicting "Number of Traces".



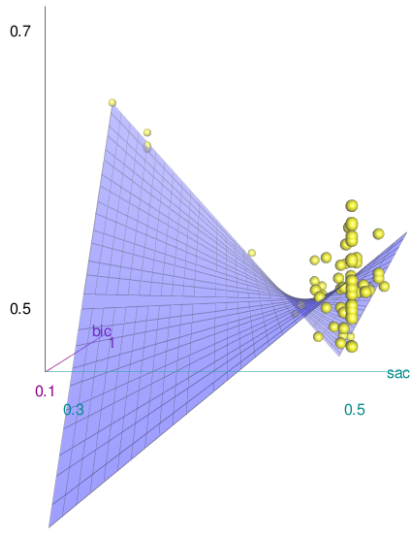
(d) C+R plot for SAC on predicting "Number of Traces".

Figure 6.5: Sampling of 2D C+R plots for linear terms from regression formulae.

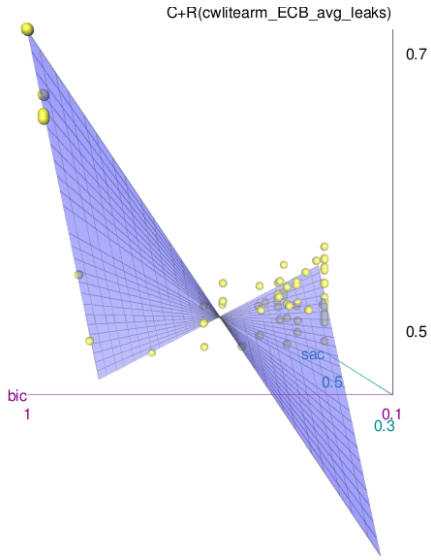
each interaction term for each regression formula, but all the plots looked like there was no horizontal plane. As we can see, these 3D C+R plots show how predicted metric values can be affected as a combination of both sac and bic. As a result, each interaction term had significant visual supplement to suggest the interaction terms significantly affected the prediction of the performance of AES with an S-box against a side-channel attack.

6.3 Creation of a Side-Channel-Resistant S-box

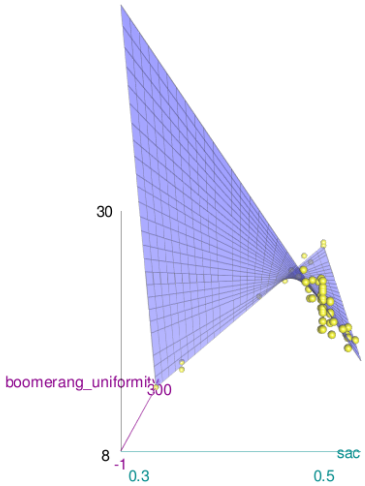
With each regression equation predicting the side-channel metric value for an S-box based on its properties computed, all that remained was to consider what properties were used the most in overall side-channel resistance. The coefficient values for each of these properties



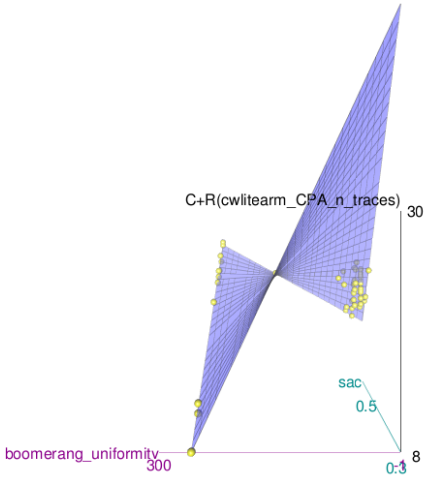
(a) x-axis view of a C+R plot for ECB mode for "Detected Leakage".



(b) y-axis view of a C+R plot for ECB mode for "Detected Leakage".



(c) x-axis view of a C+R plot for CPA for "Number of Traces".



(d) y-axis view of a C+R plot for CPA for "Number of Traces".

Figure 6.6: Sampling of 3D C+R plots interaction terms.

also needed to be considered to determine whether to minimize or maximize their values. In particular, this meant that properties of an S-box that had a negative coefficient in regression equations for "Leakage Detection" would want to be maximized, and if a property was in a regression equation to predict the result in "Number of Traces", it would be maximized if it

had a positive coefficient.

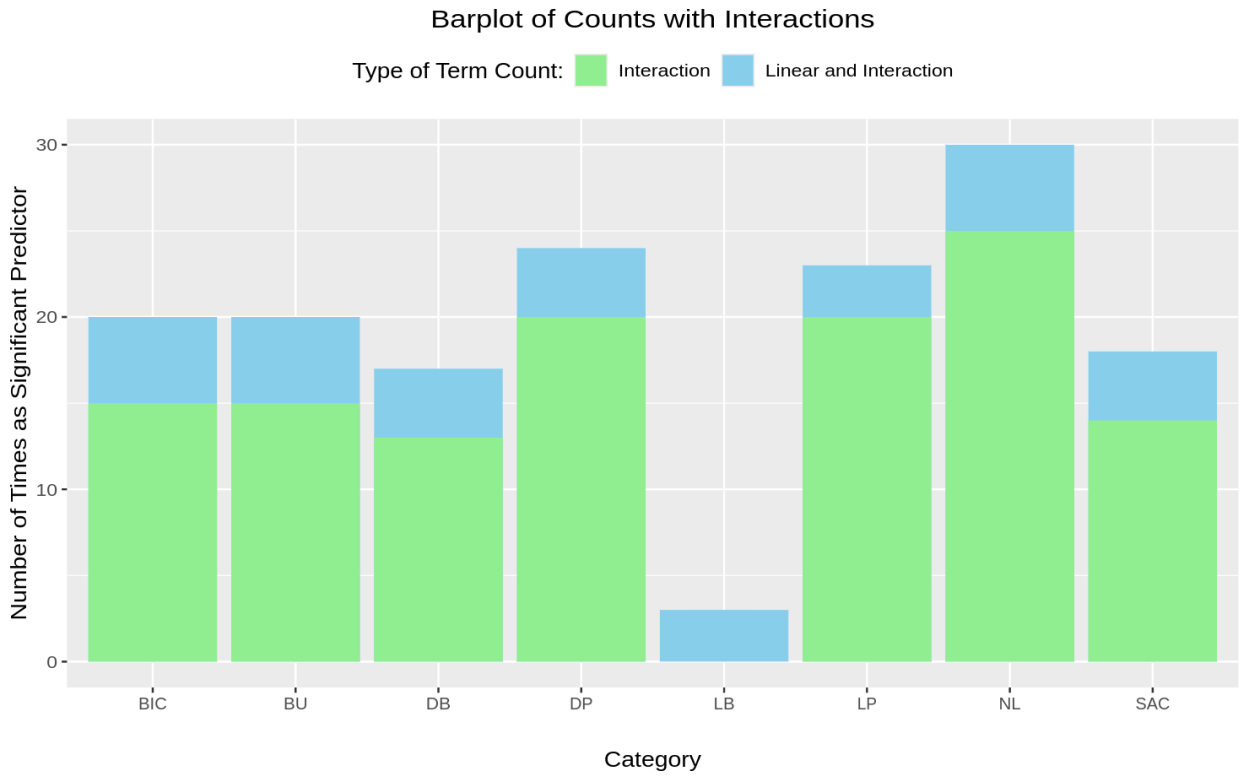


Figure 6.7: Count of how many times each S-box property significantly predicts side-channel resistance. Use Table 6.4 for Abbreviation names.

A visual representation of the frequency of statistical significance in predicting the side-channel resistance metric is presented in Figure 6.7. This plot shows the frequency of how many times each S-box property (except for linearity) was a significant predictor for side-channel analysis. As we can see, nonlinearity is used the most as a significant predictor for the side-channel resistance of an S-box used in AES. This is followed by differential probability and linear probability, but BIC, boomerang uniformity, differential branch number, and SAC also are used fairly frequently as significant predictors. However, linear branch number does not significantly predict the side-channel resistance of an S-box used in AES.

Using these results in combination with the coefficients from the regression formulae, we determine what the ideal properties of an S-box should be to add the most side-channel resistance to AES. In particular, this means maximimizing the nonlinearity and minimizing the linear probability and differential probability of an S-box. These three properties being

the most significant predictors of each metric determine that as we maximize and minimize these properties, there will be less detected leakage and will require more power traces to break

In fact, these results add to the definitions of cryptographic strength for each S-box properties. In particular, while nonlinearity measures the strength of an S-box against linear and differential cryptanalysis, it also is a good indicator of the side-channel resistance of that S-box when used in AES.

Additionally, as seen in the regression formulae, despite using different devices, the regression formulae for a metric are similar in what terms are used. Additionally, the same S-box properties can affect side-channel resistance across the different modes of AES, as we can see that linear probability, differential probability, and nonlinearity are used frequently in each regression formula for each metric result. In particular, these frequent ocurrences are in each regression formula a both linear and interaction terms. As a result, it supports that side-channel resistance is not affected by AES mode or capture device as much as the property values for an S-box.

The application of regression analysis here helps determine what properties of an S-box are significant for the side-channel resistance of AES.

6.4 Future Work

With a framework created to automate through testing the side-channel resistance of a multitude of S-boxes according to two different metrics, additional testing on different metrics could be easily incorporated to add more results for comparison. Additionally, the framework allows for any number of 8-bit S-boxes to be used in AES, and more S-boxes could be added to test for more data to analyze. Analyzing each S-box in Table 4.1 would be a good first step toward finding the most side-channel-resistant S-box given their properties. A more difficult problem would be to reverse-engineer an S-box based on properties that have most statistical significance on “Number of Traces” and “Detected Leakage.”

Other block ciphers that use 8-bit S-boxes would also be interesting targets to analyze

the side-channel resistance. Ciphers such as Camellia [93] or TwoFish [26] as the base cryptographic protocol instead of AES could add to the strength of the concept of a side-channel-resistant S-box. In particular, if similar experiments on different algorithms result in similar S-box properties being ideal for a side-channel-resistant S-box, then side-channel resistance could be seen as more universally-applicable. On the other hand, if different S-box properties affect side-channel resistance on different cryptographic protocols, then this could determine that side-channel resistance is dependent on the cryptographic algorithm itself. This future work would be difficult to implement for measurement with the "Number of Traces" metric because the S-box operation would need to be isolated in each different cryptographic protocol to accurately perform CPA and DPA akin to this experiment.

Future work could also use the built-in `SBOX2` unfinished mode of AES. This mode was created as inspired from [173] where two S-box substitutions are computed in AES rather than using the computationally-expensive `MixColumns` (Section 3.2.4) step of AES. The `SBOX2` mode would use a second S-box, and this could lead to interesting results in determining how the properties of the second S-box would affect the side-channel resistance. The possibilities of interaction terms in the regression equations between both S-box properties would be expected, but the resulting regression formulae would be much larger in number of terms.

Finally, other ChipWhisperer devices (or even other STM32 programmers) as extra data for collection would ease the bias of the gathering device. Other power-based side-channel tools to gather power traces could be utilized, but using such tools would be more difficult to incorporate into the current framework. Using varied power-based side-channel tools to gather power traces would eliminate collection precision bias by considering the effect of S-box properties' effects on the side-channel resistance metrics across a multitude of varied tools.

The framework created in this paper demonstrates that future work can result from what was proposed in this work. The concept of analyzing what parts of an S-box leads to a cryptographic protocol's side-channel resistance demonstrates that using practical data can lead to theoretical results. These theoretical results can then lead to improving AES practically against power-based side-channel attacks on a theoretical and practical level.

Abbreviations, Acronyms, and Symbols

This section provides a partial list of abbreviations, acronyms, and symbols as a guide for readers.

A

AES	Advanced Encryption Standard

B

BIC	Bit Independence Criterion

C

CBC	Cipher Block Chaining
CPA	Correlation Power Analysis
CPU	Central Processing Unit
CTR	Counter
C+R	Component and Residual

D

DES	Data Encryption Standard
DPA	Differential Power Analysis
DUT	Device Under Test

E

ECB Electronic Code Block

F

FIPS Federal Information Processing Standards

G

GF(2) Galois Field of size 2

I

iO Indistinguishability Obfuscation

IV Initialization Vector

N

NIST National Institute of Standards and Technology

NSA National Security Agency

S

SAC Strict Avalanche Criterion

SCA Side-Channel Analysis

SPA Simple Power Analysis

S-box Substitution-box

T

TVLA Test Vector Leakage Assessment

Appendix A

Additional Tables

A.1 Property values for each S-box

Table A.1 contains each S-box's property values as computed from SageMath.

A.2 Metric results for each S-box

Table A.2 contains each S-box's metric result values. The tests involved using DPA and CPA for the "Number of Traces" metric and the AES modes of ECB, CBC, and CTR for the "Detected Leakage" metric. Each of these metric were measured using power traces gathered using the CW-Nano and CW-Lite devices.

S-box Name	Nonlinearity	Linear Probability	Differential Probability	Boomerang Uniformity	Differential Branch	Linear Branch	Linearity	BIC	SAC
AES	112	0.06250	0.01562	6	2	2	32	0.13412	0.50488
ARIA_s2	112	0.06250	0.01562	6	2	2	32	0.13412	0.50293
Anubis	94	0.13281	0.03125	18	2	2	68	0.25490	0.50024
BelT	102	0.10156	0.03125	20	2	2	52	0.16841	0.49951
CLEFIA_S0	100	0.10938	0.03906	32	3	3	56	0.33333	0.53906
CLEFIA_S1	112	0.06250	0.01562	6	2	2	32	0.13170	0.49561
CMEA	96	0.12500	0.04688	-1	1	2	64	0.29659	0.51611
CSA	94	0.13281	0.04688	18	2	2	68	0.29261	0.49146
CSS	0	0.50000	0.50000	256	2	2	256	1.00000	0.28125
CS_cipher	96	0.12500	0.06250	256	2	2	64	1.00000	0.43506
Camellia	112	0.06250	0.01562	6	2	2	32	0.13170	0.49829
Chiasmus	112	0.06250	0.01562	6	2	2	32	0.13412	0.50122
Crypton_0_5	88	0.15625	0.06250	256	2	2	80	0.50000	0.48828
Crypton_1_0_S0	96	0.12500	0.03906	22	3	3	64	0.28058	0.53125
Crypton_1_0_S1	96	0.12500	0.03906	22	3	3	64	0.28058	0.53125
Crypton_1_0_S2	96	0.12500	0.03906	22	3	3	64	0.28058	0.53125
Crypton_1_0_S3	96	0.12500	0.03906	22	3	3	64	0.28058	0.53125
DBlock	112	0.06250	0.01562	6	2	2	32	0.13498	0.49780
E2	100	0.10938	0.03906	18	2	2	56	0.22124	0.50513
Enocoro	96	0.12500	0.03906	32	3	3	64	0.37796	0.53418
FLY	96	0.12500	0.06250	256	3	3	64	0.50000	0.54004
Fantomas	96	0.12500	0.06250	160	2	2	64	1.00000	0.48584
FlexAEAD	112	0.06250	0.01562	6	2	2	32	0.13412	0.50488
ForkSkinny_8	64	0.25000	0.25000	256	2	2	128	1.00000	0.31641
Fox	96	0.12500	0.06250	256	2	2	64	0.37749	0.50928
Iceberg	96	0.12500	0.03125	24	2	2	64	0.37082	0.49219
Iraqi	94	0.13281	0.04688	-1	1	2	68	0.23440	0.50610
Kalyna_pi0	104	0.09375	0.03125	16	2	2	48	0.25417	0.50391
Kalyna_pi1	104	0.09375	0.03125	16	2	2	48	0.22453	0.50928
Kalyna_pi2	104	0.09375	0.03125	18	2	2	48	0.18671	0.50122
Kalyna_pi3	104	0.09375	0.03125	16	2	2	48	0.24706	0.50293
Khazad	96	0.12500	0.03125	20	2	2	64	0.27614	0.47974
Kuznechik	100	0.10938	0.03125	16	2	2	56	0.28755	0.51245
Kuznyechik	100	0.10938	0.03125	16	2	2	56	0.28755	0.51245
Lilliput_AE	96	0.12500	0.03125	256	2	2	64	0.57735	0.47949
MD2	90	0.14844	0.03906	20	2	2	76	0.25049	0.50049
Picaro	94	0.13281	0.01562	-1	1	2	68	0.19089	0.48389
Remus_8	64	0.25000	0.25000	256	2	2	128	1.00000	0.31641
Romulus	64	0.25000	0.25000	256	2	2	128	1.00000	0.31641
SEED_S0	112	0.06250	0.01562	6	2	2	32	0.13498	0.50415
SEED_S1	112	0.06250	0.01562	6	2	2	32	0.12555	0.49854
SKINNY_8	64	0.25000	0.25000	256	2	2	128	1.00000	0.31641
SMS4	112	0.06250	0.01562	6	2	2	32	0.13498	0.49976
SNOW_3G_sq	96	0.12500	0.03125	24	2	2	64	0.38719	0.50269
Safer	82	0.17969	0.50000	256	2	2	92	0.78278	0.54028
Scream	96	0.12500	0.03125	256	2	2	64	0.57735	0.49854
Skipjack	100	0.10938	0.04688	20	2	2	56	0.30283	0.50317
Streebog	100	0.10938	0.03125	16	2	2	56	0.28755	0.51245
Stribog	100	0.10938	0.03125	16	2	2	56	0.28755	0.51245
Turing	94	0.13281	0.04688	20	2	2	68	0.22353	0.50244
Twofish_p0	96	0.12500	0.03906	48	2	2	64	0.28983	0.48999
Twofish_p1	96	0.12500	0.03906	40	2	2	64	0.25064	0.50049
Whirlpool	100	0.10938	0.03125	18	2	2	56	0.31331	0.51489
ZUC_S0	96	0.12500	0.03125	256	2	2	64	0.50000	0.49487
ZUC_S1	112	0.06250	0.01562	6	2	2	32	0.12999	0.50928
Zorro	96	0.12500	0.03906	42	2	2	64	0.43836	0.49268
iScream	96	0.12500	0.06250	256	2	2	64	0.57735	0.51880
newDES	92	0.14062	0.04688	22	2	2	72	0.21972	0.49683
S1	112	0.06250	0.01562	6	2	2	32	0.13170	0.49512
S2	112	0.06250	0.01562	6	2	2	32	0.12610	0.49707
S3	112	0.06250	0.01562	6	2	2	32	0.13170	0.49707
S4	112	0.06250	0.01562	6	2	2	32	0.12610	0.50000
S5	112	0.06250	0.01562	6	2	2	32	0.13170	0.49951
S6	112	0.06250	0.01562	6	2	2	32	0.12610	0.49536
S7	112	0.06250	0.01562	6	2	2	32	0.12610	0.49609
S8	112	0.06250	0.01562	6	2	2	32	0.13170	0.49536

Table A.1: S-box property values.

S-box Name	Metric Results									
	"Number of Traces" metric				"Detected Leakage" metric					
	DPA		CPA		ECB		CBC		CTR	
	CW-Lite	CW-Nano	CW-Lite	CW-Nano	CW-Lite	CW-Nano	CW-Lite	CW-Nano	CW-Lite	CW-Nano
AES	696	1714	23	48	0.55349	0.49239	0.10077	0.00815	0.04563	0.0
ARIA_s2	620	> 2000	23	50	0.51231	0.52465	0.10041	0.00789	0.04456	0.0
Anubis	> 1000	> 2000	28	41	0.54643	0.52184	0.09283	0.00771	0.04245	0.0
BelT	> 1000	> 2000	25	50	0.53611	0.48809	0.09919	0.00787	0.04585	0.0
CLEFIA_S0	585	1684	24	46	0.54451	0.50700	0.10069	0.00784	0.04200	0.0
CLEFIA_S1	> 1000	1914	24	49	0.53212	0.52824	0.10268	0.00797	0.04504	0.0
CMEA	> 1000	> 2000	26	51	0.51932	0.48633	0.06465	0.00796	0.04647	0.0
CSA	> 1000	1989	25	47	0.57307	0.47908	0.01131	0.00829	0.04145	0.0
CSS	> 1000	> 2000	25	47	0.50945	0.49605	0.03713	0.00831	0.04212	0.0
CS_cipher	> 1000	> 2000	28	37	0.59176	0.48869	0.06499	0.00765	0.04204	0.0
Camellia	927	1421	25	52	0.54484	0.48409	0.08504	0.00767	0.04447	0.0
Chiasmus	622	1618	25	51	0.54668	0.47141	0.09513	0.00803	0.04215	0.0
Crypton_0_5	> 1000	1782	26	52	0.56708	0.47715	0.09540	0.00800	0.04224	0.0
Crypton_1_0_S0	752	1643	24	49	0.52997	0.47913	0.09760	0.00795	0.04212	0.0
Crypton_1_0_S1	921	> 2000	25	52	0.56253	0.51943	0.10183	0.00769	0.04391	0.0
Crypton_1_0_S2	> 1000	1496	25	52	0.52569	0.48675	0.10107	0.00792	0.04324	0.0
Crypton_1_0_S3	835	> 2000	26	51	0.52575	0.48573	0.10191	0.00803	0.04196	0.0
DBlock	574	1524	23	49	0.52917	0.47416	0.09716	0.00768	0.04029	0.0
E2	649	> 2000	24	54	0.55657	0.48236	0.10053	0.00769	0.03889	0.0
Enocoro	623	1846	24	48	0.51960	0.49280	0.10149	0.00784	0.04127	0.0
FLY	> 1000	> 2000	26	47	0.52323	0.50257	0.10108	0.00772	0.04008	0.0
Fantomas	> 1000	> 2000	25	49	0.51515	0.47435	0.09931	0.00809	0.04044	0.0
FlexAEAD	718	1500	26	47	0.55515	0.49175	0.10105	0.00767	0.04163	0.0
ForkSkinny_8	> 1000	> 2000	21	46	0.53516	0.50203	0.10101	0.00775	0.04159	0.0
Fox	> 1000	1598	26	50	0.53912	0.49704	0.10152	0.00808	0.04308	0.0
Iceberg	873	> 2000	25	42	0.51971	0.46757	0.09924	0.00775	0.04029	0.0
Iraqi	> 1000	> 2000	26	50	0.52907	0.46435	0.10556	0.00816	0.03964	0.0
Kalyna_pi0	> 1000	> 2000	26	48	0.50851	0.47771	0.10297	0.00796	0.04047	0.0
Kalyna_pi1	759	> 2000	26	48	0.49855	0.47763	0.10208	0.00772	0.03899	0.0
Kalyna_pi2	869	1667	25	54	0.54353	0.48744	0.10356	0.00805	0.04084	0.0
Kalyna_pi3	> 1000	2000	23	53	0.48784	0.46537	0.10096	0.00803	0.04053	0.0
Khazad	> 1000	> 2000	29	37	0.52049	0.47899	0.09628	0.00788	0.04095	0.0
Kuznechik	551	1750	24	48	0.54185	0.46843	0.10105	0.00772	0.03941	0.0
Kuznyechik	496	> 2000	25	47	0.52536	0.46903	0.10033	0.00856	0.01764	0.0
Lilliput_AE	> 1000	> 2000	24	52	0.51652	0.47331	0.10051	0.00811	0.04023	0.0
MD2	723	1832	24	47	0.48296	0.48291	0.10092	0.00808	0.04025	0.0
Picaro	363	> 2000	26	43	0.52141	0.50267	0.10373	0.00829	0.04003	0.0
Remus_8	> 1000	> 2000	22	49	0.51568	0.50032	0.10271	0.00825	0.04077	0.0
Romulus	> 1000	> 2000	21	57	0.51816	0.49969	0.09947	0.00799	0.04028	0.0
SEED_S0	562	1437	25	47	0.48451	0.48231	0.10060	0.00801	0.04101	0.0
SEED_S1	490	1450	25	47	0.57176	0.49963	0.09323	0.00792	0.04069	0.0
SKINNY_8	996	> 2000	22	57	0.51976	0.50539	0.09353	0.00781	0.04025	0.0
SMS4	> 1000	> 2000	23	46	0.49159	0.49088	0.09561	0.00781	0.04311	0.0
SNOW_3G_sq	758	> 2000	25	48	0.49061	0.48057	0.10123	0.00835	0.04079	0.0
Safer	> 1000	> 2000	24	48	0.54315	0.51165	0.09668	0.00808	0.04264	0.0
Scream	> 1000	> 2000	25	53	0.53411	0.47043	0.10055	0.00765	0.04316	0.0
Skipjack	> 1000	> 2000	24	47	0.49307	0.49131	0.09433	0.00796	0.04317	0.0
Streebog	594	> 2000	24	50	0.52440	0.46417	0.09672	0.00795	0.04512	0.0
Stribog	626	1770	24	49	0.51383	0.46561	0.09776	0.00793	0.04344	0.0
Turing	> 1000	> 2000	24	48	0.51408	0.49501	0.10159	0.00815	0.04273	0.0
Twofish_p0	> 1000	1563	25	44	0.50441	0.47095	0.10265	0.00772	0.04071	0.0
Twofish_p1	780	1538	26	54	0.50396	0.47173	0.09936	0.00791	0.03964	0.0
Whirlpool	> 1000	1637	26	48	0.51792	0.44931	0.09996	0.00797	0.04256	0.0
ZUC_S0	> 1000	> 2000	24	48	0.53611	0.50060	0.09925	0.00851	0.04095	0.0
ZUC_S1	766	> 2000	22	50	0.49243	0.47168	0.10277	0.00792	0.04200	0.0
Zorro	518	> 2000	23	49	0.48153	0.49120	0.10287	0.00768	0.04248	0.0
iScream	877	> 2000	27	42	0.47855	0.46225	0.09221	0.00784	0.04177	0.0
newDES	508	1693	22	46	0.48271	0.47832	0.09433	0.00804	0.04167	0.0
S1	615	1341	22	52	0.51036	0.49983	0.09720	0.00773	0.04316	0.0
S2	734	1425	25	49	0.51547	0.47417	0.09335	0.00773	0.04496	0.0
S3	746	1876	23	51	0.49639	0.49691	0.09507	0.00763	0.04253	0.0
S4	821	> 2000	24	51	0.46424	0.46436	0.09377	0.00799	0.04183	0.0
S5	920	> 2000	24	51	0.50357	0.46464	0.09840	0.00793	0.04171	0.0
S6	888	> 2000	23	48	0.48757	0.46469	0.09949	0.00775	0.04136	0.0
S7	> 1000	> 2000	21	49	0.49319	0.51251	0.09325	0.00772	0.04019	0.0
S8	625	> 2000	23	49	0.49741	0.48289	0.09355	0.00768	0.03647	0.0

Table A.2: Metric results for each S-box values.

Appendix B

Regression Formulae

Reference table for acronyms

As shown in Chapter 6, Table B.1 demonstrates what each abbreviation means for each regression term.

Abbreviation	Property Name
`bic`	Bit Independence Criterion
`bu`	Boomerang Uniformity
`db`	Differential Branch Number
`dp`	Differential Probability
`lb`	Linear Branch Number
`lp`	Linear Probability
`l`	Linearity
`nl`	Nonlinearity
`sac`	Strict Avalanche Criterion

Table B.1: Abbreviation of S-box properties for regression formulae.

B.1 "Number of Traces" Metric with DPA on CW-Nano

Equation B.1 is the regression equation to predict "Number of Traces" metric with DPA on CW-Nano.

$$\begin{aligned} nt_{DPA,Nano} = & \ 48862.356 - 244.823 * \texttt{bu} \\ & + 228050.325 * \texttt{bic} \\ & - 1367.735 * \texttt{nl*lp} \\ & - 18967.821 * \texttt{nl*dp} \\ & - 176.115 * \texttt{nl*db} \\ & - 5100151.698 * \texttt{lp*dp} \\ & + 342575.16 * \texttt{dp*db} \\ & + 3635630.419 * \texttt{dp*sac} \\ & + 116.892 * \texttt{bu*db} \\ & - 442996.431 * \texttt{bic*sac} \end{aligned} \tag{B.1}$$

B.2 "Number of Traces" Metric with DPA on CW-Lite

Equation B.2 is the regression equation to predict "Number of Traces" metric with DPA on CW-Lite.

$$\begin{aligned} nt_{DPA,Lite} = & - 4207.5 + 172543.649 * \texttt{dp} \\ & - 0.137 * \texttt{nl*bu} \\ & + 18.943 * \texttt{nl*db} \\ & - 928675.838 * \texttt{dp*lp} \\ & + 112486.252 * \texttt{lp*bic} \\ & + 39413.24 * \texttt{lp*sac} \\ & + 283.252 * \texttt{dp*bu} \\ & - 40066.13 * \texttt{dp*db} \\ & - 29733.291 * \texttt{bic*sac} \end{aligned} \tag{B.2}$$

B.3 "Number of Traces" Metric with CPA on CW-Nano

Equation B.3 is the regression equation to predict "Number of Traces" metric with CPA on CW-Nano.

$$\begin{aligned} nt_{CPA,Nano} = &- 1209.38 + 10.733 * \texttt{nl} \\ &+ 0.411 * \texttt{bu} \\ &- 11.348 * \texttt{db} \\ &+ 633.366 * \texttt{bic} \\ &+ 1445.864 * \texttt{sac} \\ &+ 9.416 * \texttt{nl*lp} \\ &- 8.092 * \texttt{nl*dp} \\ &- 5.466 * \texttt{nl*bic} \\ &- 12.545 * \texttt{nl*sac} \\ &- 2.416 * \texttt{bu*dp} \\ &+ 597.781 * \texttt{db*dp} \\ &- 0.61 * \texttt{bu*sac} \\ &- 55.388 * \texttt{db*bic} \end{aligned} \tag{B.3}$$

B.4 "Number of Traces" Metric with CPA on CW-Lite

Equation B.4 is the regression equation to predict "Number of Traces" metric with CPA on CW-Lite.

$$\begin{aligned} nt_{CPA,Lite} = {} & 466.27 - 4.115 * \texttt{nl} \\ & - 0.548 * \texttt{bu} \\ & + 70.73 * \texttt{db} + 7.03 * \texttt{lb} \\ & + 97.263 * \texttt{bic} \\ & - 1019.671 * \texttt{sac} \\ & + 3.656 * \texttt{nl*dp} \\ & - 0.638 * \texttt{nl*db} \\ & + 9.156 * \texttt{nl*sac} \\ & + 708.008 * \texttt{dp*lp} \\ & + 0.849 * \texttt{bu*lp} \\ & - 856.375 * \texttt{bic*lp} \\ & + 0.821 * \texttt{bu*dp} \\ & - 304.554 * \texttt{db*dp} \\ & + 0.078 * \texttt{bu*bic} \\ & + 0.762 * \texttt{bu*sac} \end{aligned} \tag{B.4}$$

B.5 "Detected Leakage" Metric on ECB mode on CW-Nano

Equation B.5 is the regression equation to predict "Detected Leakage" metric with AES on ECB mode on CW-Nano.

$$\begin{aligned} dt_{ECB,Nano} = {} & 5.56 - 0.053 * \texttt{nl} - 274548.223 * \texttt{dp} \\ & + 617.895 * \texttt{bu} - 2.575 * \texttt{db} \\ & + 2144.911 * \texttt{nl*dp} \\ & - 4.827 * \texttt{nl*bu} \\ & + 0.027 * \texttt{nl*db} \\ & + 549097.156 * \texttt{dp*lp} \\ & - 1235.792 * \texttt{bu*lp} \end{aligned} \tag{B.5}$$

B.6 "Detected Leakage" Metric on ECB mode on CW-Lite

Equation B.6 is the regression equation to predict "Detected Leakage" metric with AES on ECB mode on CW-Lite.

$$\begin{aligned} dt_{ECB,Lite} = {} & 107116.884 - 836.841 * \texttt{nl} \\ & - 214241.468 * \texttt{lp} \\ & - 0.985 * \texttt{dp} \\ & + 57238.184 * \texttt{bic} \\ & - 242046.014 * \texttt{sac} \\ & - 447.165 * \texttt{nl*bic} \\ & + 1890.973 * \texttt{nl*sac} \\ & + 6.386 * \texttt{lp*dp} \\ & - 114473.462 * \texttt{lp*bic} \\ & + 484109.505 * \texttt{lp*sac} \\ & - 2.333 * \texttt{bic*sac} \end{aligned} \tag{B.6}$$

B.7 "Detected Leakage" Metric on CBC mode on CW-Nano

Equation B.7 is the regression equation to predict "Detected Leakage" metric with AES on CBC mode on CW-Nano.

$$dt_{CBC,Nano} = 0.008 + 0.000001538 * \texttt{nl*db} \tag{B.7}$$

B.8 "Detected Leakage" Metric on CBC mode on CW-Lite

Equation B.8 is the regression equation to predict "Detected Leakage" metric with AES on CBC mode on CW-Lite.

$$\begin{aligned} dt_{CBC,Lite} = & -236520.125 + 1847.791 * \texttt{nl} \\ & + 473055.326 * \texttt{lp} + 472.973 * \texttt{bu} \\ & - 4.459 * \texttt{db} - 0.041 * \texttt{lb} \\ & - 38355.92 * \texttt{bic} \\ & + 470198.545 * \texttt{sac} \\ & + 0.025 * \texttt{nl*lp} \\ & - 0.028 * \texttt{nl*dp} \\ & - 3.695 * \texttt{nl*bu} \\ & + 0.047 * \texttt{nl*db} \\ & + 299.659 * \texttt{nl*bic} \\ & - 3673.427 * \texttt{nl*sac} \\ & - 945.941 * \texttt{lp*bu} \\ & + 76707.868 * \texttt{lp*bic} \\ & - 940392.618 * \texttt{lp*sac} \\ & + 4.682 * \texttt{bic*dp} \\ & + 0.0004 * \texttt{bu*bic} \end{aligned} \tag{B.8}$$

References

[1] W. C. Barker, "Guideline for Identifying an Information System as a National Security System," *National Institute of Standards and Technology*, Aug. 2003.

[2] C. Paar and J. Pelzl, "The Advanced Encryption Standard (AES)," in *Understanding Cryptography: A Textbook for Students and Practitioners* (C. Paar and J. Pelzl, eds.), pp. 87–121, Berlin, Heidelberg: Springer, 2010.

[3] P. Kocher, J. Jaffe, B. Jun, and P. Rohatgi, "Introduction to differential power analysis," *Journal of Cryptographic Engineering*, vol. 1, pp. 5–27, Apr. 2011. Number: 1.

[4] O. Lo, W. J. Buchanan, and D. Carson, "Power analysis attacks on the AES-128 S-box using differential power analysis (DPA) and correlation power analysis (CPA)," *Journal of Cyber Security Technology*, vol. 1, no. 2, pp. 88–107, 2017. Publisher: Taylor & Francis.

[5] JoeyLupo, "cryptolib," Accessed 2023. Accessed on October 1, 2023.

[6] L. de Moura, S. Kong, J. Avigad, F. van Doorn, and J. von Raumer, "The Lean Theorem Prover (System Description)," in *Automated Deduction - CADE-25* (A. P. Felty and A. Middeldorp, eds.), (Cham), pp. 378–388, Springer International Publishing, 2015.

[7] T. mathlib Community, "The lean mathematical library," in *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2020, (New York, NY, USA), pp. 367–381, Association for Computing Machinery, 2020.

[8] C. Carper, S. Olguin, J. Brown, C. Charlton, and M. Borowczak, "Challenging Assumptions of Normality in AES s-Box Configurations under Side-Channel Analysis," *Journal of Cybersecurity and Privacy*, vol. 3, pp. 844–857, Dec. 2023. Number: 4 Publisher: Multidisciplinary Digital Publishing Institute.

[9] N. Koblitz and A. J. Menezes, "Another Look at "Provable Security"," *Journal of Cryptology*, vol. 20, pp. 3–37, Jan. 2006.

[10] Overleaf, "Overleaf." `https://www.overleaf.com`, 2024. Accessed: Apr. 2024.

[11] Draw.io, "Draw.io." `https://www.draw.io`, 2024. Accessed: Apr. 2024.

[12] M. J. Dworkin, "Advanced Encryption Standard (AES)," Tech. Rep. NIST FIPS 197-upd1, National Institute of Standards and Technology (U.S.), Gaithersburg, MD, May 2023.

[13] T. Kelly, "THE MYTH OF THE SKYTALE," *Cryptologia*, vol. 22, pp. 244–260, July 1998.

[14] T. Limbong and P. D. Silitonga, "Testing the classic caesar cipher cryptography using of matlab," *International Journal of Engineering Research & Technology*, vol. 6, no. 2, pp. 175–178, 2017.

[15] D. Luciano and G. Prichett, "Cryptology: From Caesar Ciphers to Public-key Cryptosystems," *The College Mathematics Journal*, vol. 18, pp. 2–17, Jan. 1987. Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/07468342.1987.11973000.

[16] B. Schneier, "A Self-Study Course in Block-Cipher Cryptanalysis," *Cryptologia*, vol. 24, pp. 18–33, Jan. 2000. Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/0161-110091888754.

[17] S. Sharma and Y. Gupta, "Study on cryptography and techniques," *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, vol. 2, no. 1, pp. 249–252, 2017.

[18] H. F. Gaines, *Cryptanalysis: a study of ciphers and their solution*. New York: Dover Publ, corr. ed ed., 1989.

[19] D. R. Ridley, L. G. Ridley, and C. B. Walker, "English Letter Frequencies as Found in Whissell's Parsimonious Sampling of English Words," *Perceptual and Motor Skills*, vol. 88, pp. 607–614, Apr. 1999.

[20] A. G. Konheim, *Cryptography: a primer*. A Wiley-Interscience publication, New York: Wiley, 1981.

[21] G. J. Simmons, "Symmetric and Asymmetric Encryption," *ACM Computing Surveys*, vol. 11, pp. 305–330, Dec. 1979.

[22] N. I. of Standards and Technology, "Skipjack and KEA Algorithm Specifications," *NIST*, 1998.

[23] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, pp. 120–126, Feb. 1978.

[24] J. Daemen, L. Knudsen, and V. Rijmen, "The block cipher Square," in *Fast Software Encryption* (E. Biham, ed.), Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 149–165, Springer, 1997.

[25] W. Diffie and M. Hellman, "Special Feature Exhaustive Cryptanalysis of the NBS Data Encryption Standard," *Computer*, vol. 10, pp. 74–84, June 1977.

[26] B. Schneier, J. Kelsey, D. Whiting, N. Ferguson, D. Wagner, and C. Hall, "Twofish: A 128-Bit Block Cipher," *AES Submission*, 2008.

[27] A. Deane and A. Kraus, *The Official (ISC)2 CISSP CBK Reference*. Indianapolis: John Wiley and Sons, 6th edition ed., 2021.

[28] V. Grosso, *Low Entropy Masking Schemes, Revisited*, vol. 8419 of *Lecture Notes in Computer Science*. Cham: Springer International Publishing, 2014.

[29] ETSI/Sage, "Specification of the 3GPP Confidentiality and Integrity Algorithms UEA2 & UIA2," *Document 5: Design and Evaluation Report*, 2011.

[30] G. Rose and P. Hawkes, "Turing, a fast stream cipher," 2002. Publication info: Published elsewhere. Accepted to FSE 2003, Lund, SE.

[31] M. Rathidevi, R. Yaminipriya, and S. V. Sudha, "Trends of cryptography stepping from ancient to modern," in *2017 International Conference on Innovations in Green Energy and Healthcare Technologies (IGEHT)*, pp. 1–9, 2017.

[32] P. M. Valkenburg and J. Peter, "Internet communication and its relation to well-being: Identifying some underlying mechanisms," *Media Psychology*, vol. 9, no. 1, pp. 43–58, 2007.

[33] A. Pay, "Transaction and account security | Amazon Pay Help," 2024.

[34] F. Piper, "An introduction to cryptography," *INFORMATION SYSTEMS CONTROL JOURNAL*, vol. 6, pp. 54–61, 2003.

[35] S. Arora and B. Barak, *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[36] C. M. Institute, "The Millennium Prize Problems," 2000.

[37] P. W. Shor, "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer," *SIAM Journal on Computing*, vol. 26, pp. 1484–1509, Oct. 1997. arXiv:quant-ph/9508027.

[38] N. I. o. S. a. Technology, "Data Encryption Standard (DES)," Tech. Rep. Federal Information Processing Standard (FIPS) 46-1 (Withdrawn), U.S. Department of Commerce, Jan. 1988.

[39] M. Matsui, "The First Experimental Cryptanalysis of the Data Encryption Standard," in *Advances in Cryptology — CRYPTO '94* (Y. G. Desmedt, ed.), Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 1–11, Springer, 1994.

[40] M. Matsui, "Linear Cryptanalysis Method for DES Cipher," in *Advances in Cryptology — EUROCRYPT '93*, vol. 765, pp. 386–397, Berlin, Heidelberg: Springer Berlin Heidelberg, 1994. Series Title: Lecture Notes in Computer Science.

[41] J. Håstad, R. Impagliazzo, L. A. Levin, and M. Luby, "A pseudorandom generator from any one-way function," *SIAM Journal on Computing*, vol. 28, no. 4, pp. 1364–1396, 1999.

[42] Q. H. Dang, "Secure Hash Standard," Tech. Rep. NIST FIPS 180-4, National Institute of Standards and Technology, July 2015.

[43] kokke, "kokke/tiny-AES-c," Jan. 2024. original-date: 2012-05-24T15:27:24Z.

[44] M. Ciampa, *CompTIA security+ guide to network security fundamentals*. Cengage Learning, 2015.

[45] S. Almuhammadi and I. Al-Hejri, "A comparative analysis of AES common modes of operation," in *2017 IEEE 30th Canadian Conference on Electrical and Computer Engineering (CCECE)*, pp. 1–4, Apr. 2017.

[46] M. Dworkin, "Recommendation for Block Cipher Modes of Operation: Methods and Techniques," Tech. Rep. NIST Special Publication (SP) 800-38A, National Institute of Standards and Technology, Dec. 2001.

[47] H. M. Heys, "A TUTORIAL ON LINEAR AND DIFFERENTIAL CRYPTANALYSIS," *Cryptologia*, vol. 26, pp. 189–221, July 2002.

[48] J. Daemen and V. Rijmen, *The Design of Rijndael*. Information Security and Cryptography, Berlin, Heidelberg: Springer Berlin Heidelberg, 2nd ed., 2020.

[49] I. T. L. Computer Security Division, "AES Development - Cryptographic Standards and Guidelines | CSRC | CSRC," Dec. 2016.

[50] A. Jain, H. Lin, and A. Sahai, "Indistinguishability obfuscation from well-founded assumptions," in *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pp. 60–73, 2021.

[51] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, and K. Yang, "On the (im)possibility of obfuscating programs," *Journal of the ACM*, vol. 59, pp. 1–48, Apr. 2012.

[52] A. Shamir, R. L. Rivest, and L. M. Adleman, "Mental Poker," in *The Mathematical Gardner* (D. A. Klarner, ed.), pp. 37–43, Boston, MA: Springer US, 1981.

[53] S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters, "Candidate Indistinguishability Obfuscation and Functional Encryption for all Circuits," in *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, (Berkeley, CA, USA), pp. 40–49, IEEE, Oct. 2013.

[54] C. E. Shannon, "Communication theory of secrecy systems," *The Bell System Technical Journal*, vol. 28, pp. 656–715, Oct. 1949. Conference Name: The Bell System Technical Journal.

[55] J.D, "Answer to "How do we prove that AES, DES etc. are secure?"," Oct. 2013.

[56] Y. Watanabe, J. Shikata, and H. Imai, "Equivalence between Semantic Security and Indistinguishability against Chosen Ciphertext Attacks," in *Public Key Cryptography — PKC 2003* (Y. G. Desmedt, ed.), Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 71–84, Springer, 2002.

[57] O. Goldreich, "On Post-Modern Cryptography," 2006. Publication info: Published elsewhere. Unknown where it was published.

[58] J. H. Fetzer, "Program verification: the very idea," *Communications of the ACM*, vol. 31, pp. 1048–1063, Sept. 1988.

[59] R. J. Lipton and A. J. Perlis, "Social Processes and Proofs of Theorems and Programs," *Association for Computing Machinery*, vol. 22, no. 5, 1979.

[60] J. Daemen and V. Rijmen, "AES proposal: Rijndael," 1999.

[61] W. Alliance, "Security | Wi-Fi Alliance," 2020.

[62] Google, "Default encryption at rest | Documentation," Sept. 2022.

[63] Facebook, "Messenger Secret Conversations Technical Whitepaper," May 2017.

[64] K. Bousselam, G. Di, M.-L. Flottes, and B. Rouzeyre, "Fault Detection in Crypto-Devices," in *Fault Detection*, InTech, Mar. 2010.

[65] S. Mangard, "A Simple Power-Analysis (SPA) Attack on Implementations of the AES Key Expansion," in *Information Security and Cryptology — ICISC 2002* (P. J. Lee and C. H. Lim, eds.), Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 343–358, Springer, 2003.

[66] R. Ueno, N. Homma, and T. Aoki, "Toward More Efficient DPA-Resistant AES Hardware Architecture Based on Threshold Implementation," in *Constructive Side-Channel Analysis and Secure Design* (S. Guilley, ed.), Lecture Notes in Computer Science, (Cham), pp. 50–64, Springer International Publishing, 2017.

[67] X. Wang, J. Zheng, L. Wu, J. Zhu, and W. Hu, "A Correlation Fault Attack on Rotating S-Box Masking AES," in *2021 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*, (Shanghai, China), pp. 1–6, IEEE, Dec. 2021.

[68] A. Biryukov, "Related Key Attack," in *Encyclopedia of Cryptography and Security* (H. C. A. van Tilborg and S. Jajodia, eds.), pp. 1040–1041, Boston, MA: Springer US, 2011.

[69] A. Biryukov and D. Khovratovich, "Related-Key Cryptanalysis of the Full AES-192 and AES-256," in *Advances in Cryptology – ASIACRYPT 2009* (M. Matsui, ed.), Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 1–18, Springer, 2009.

[70] X. Bonnetain, M. Naya-Plasencia, and A. Schrottenloher, "Quantum Security Analysis of AES," *IACR Transactions on Symmetric Cryptology*, pp. 55–93, June 2019.

[71] E. Biham and A. Shamir, "Differential fault analysis of secret key cryptosystems," in *Advances in Cryptology — CRYPTO '97* (G. Goos, J. Hartmanis, J. Van Leeuwen, and B. S. Kaliski, eds.), vol. 1294, pp. 513–525, Berlin, Heidelberg: Springer Berlin Heidelberg, 1997. Series Title: Lecture Notes in Computer Science.

[72] C. Adams and S. Tavares, "The structured design of cryptographically good s-boxes," *Journal of Cryptology*, vol. 3, pp. 27–41, Jan. 1990.

[73] K. Nyberg, "Differentially uniform mappings for cryptography," in *Advances in Cryptology — EUROCRYPT '93* (T. Helleseth, ed.), Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 55–64, Springer, 1994.

[74] "Finite field," Feb. 2024. Page Version ID: 1208645670.

[75] E. W. Weisstein, "Affine Transformation," 2024. Publisher: Wolfram Research, Inc.

[76] S. Wang, "The difference in five modes in the AES encryption algorithm - Highgo Software Inc.," Aug. 2019.

[77] S. Arrag, A. Hamdoun, A. Tragha, and S. Khamlich, "Design and Implementation A different Architectures of mixcolumn in FPGA," *International Journal of VLSI Design & Communication Systems*, vol. 3, Sept. 2012.

[78] F. V. Wenceslao Jr., "Enhancing the Performance of the Advanced Encryption Standard (AES) Algorithm Using Multiple Substitution Boxes," *International Journal of Communication Networks and Information Security (IJCNIS)*, vol. 10, no. 3, 2018. Number: 3.

[79] B. A. Forouzan, *Cryptography & network security*. McGraw-Hill, Inc., 2007.

[80] D. Coppersmith and S. Winograd, "Matrix multiplication via arithmetic progressions," in *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, STOC '87, (New York, NY, USA), pp. 1–6, Association for Computing Machinery, Jan. 1987.

[81] K. Jithendra and T. Shahana, "Hardware Efficient Parallel Substitution Box for Block Ciphers with Static and Dynamic Properties," *Procedia Computer Science*, vol. 46, pp. 540–547, Dec. 2015.

[82] D. Genkin, A. Shamir, and E. Tromer, "RSA Key Extraction via Low-Bandwidth Acoustic Cryptanalysis," 2013. Publication info: Preprint. MINOR revision.

[83] M. G. Kuhn and R. J. Anderson, "Soft Tempest: Hidden Data Transmission Using Electromagnetic Emanations," in *Information Hiding* (D. Aucsmith, ed.), Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 124–142, Springer, 1998.

[84] R. J. Lipton and J. F. Naughton, "Clocked adversaries for hashing," *Algorithmica*, vol. 9, pp. 239–252, Mar. 1993.

[85] J. Bonneau and I. Mironov, "Cache-Collision Timing Attacks Against AES," in *Cryptographic Hardware and Embedded Systems - CHES 2006* (L. Goubin and M. Matsui, eds.), Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 201–215, Springer, 2006.

[86] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Annual international cryptology conference*, pp. 388–397, Springer, 1999.

[87] J. Hertz, "An Introduction to Power Analysis Side Channel Attacks - Technical Articles," Mar. 2023.

[88] Yan1x0s, "Side Channel Attacks — Part 2 (DPA & CPA applied on AES Attack)," Apr. 2021.

[89] M. Randolph and W. Diehl, "Power side-channel attack analysis: A review of 20 years of study for the layman," *Cryptography*, vol. 4, no. 2, p. 15, 2020. Publisher: Multidisciplinary Digital Publishing Institute.

[90] G. S. Ohm, J. G. F. Kniestädt, f. o. D. Kiesel, and d. D. Burndy Library, *Die galvanische Kette*. Berlin : Bei T.H. Riemann, 1827.

[91] R. Mayer-Sommer, "Smartly Analyzing the Simplicity and the Power of Simple Power Analysis on Smartcards," in *Cryptographic Hardware and Embedded Systems — CHES 2000* (C. K. Koc and C. Paar, eds.), Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 78–92, Springer, 2000.

[92] Y. Lu, M. P. O'Neill, and J. V. McCanny, "Differential Power Analysis resistance of Camellia and countermeasure strategy on FPGAs," in *2009 International Conference on Field-Programmable Technology*, pp. 183–189, Dec. 2009.

[93] K. Aoki, T. Ichikawa, M. Kanda, M. Matsui, S. Moriai, J. Nakajima, and T. Tokita, "Camellia: A 128-Bit Block Cipher Suitable for Multiple Platforms — Design andAnalysis," in *Selected Areas in Cryptography*, vol. 2012, pp. 39–56, Berlin, Heidelberg: Springer Berlin Heidelberg, 2001. Series Title: Lecture Notes in Computer Science.

[94] M. Rivain, E. Prouff, and J. Doget, "Higher-Order Masking and Shuffling for Software Implementations of Block Ciphers," in *Cryptographic Hardware and Embedded Systems - CHES 2009* (C. Clavier and K. Gaj, eds.), Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 171–188, Springer, 2009.

[95] H. Maghrebi, J.-L. Danger, F. Flament, S. Guilley, and L. Sauvage, "Evaluation of countermeasure implementations based on Boolean masking to thwart side-channel attacks," in *2009 3rd International Conference on Signals, Circuits and Systems (SCS)*, (Medenine, Tunisia), pp. 1–6, IEEE, Nov. 2009.

[96] R. Soares, V. Lima, R. Lellis, P. Finkenauer Jr., and V. Camargo, "Hardware Countermeasures against Power Analysis Attacks: a Survey from Past to Present," *Journal of Integrated Circuits and Systems*, vol. 16, pp. 1–12, Aug. 2021.

[97] C. Clavier, J.-S. Coron, and N. Dabbous, "Differential power analysis in the presence of hardware countermeasures," in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 252–263, Springer, 2000.

[98] J. W. L. Glaisher, *The Quarterly Journal of Pure and Applied Mathematics*. Longmans, Green, 1899. Google-Books-ID: j7sKAAAAIAAJ.

[99] I. Reed, "A class of multiple-error-correcting codes and the decoding scheme," *Transactions of the IRE Professional Group on Information Theory*, vol. 4, pp. 38–49, Sept. 1954.

[100] E. Brier, C. Clavier, and F. Olivier, "Correlation Power Analysis with a Leakage Model," in *Cryptographic Hardware and Embedded Systems - CHES 2004*, vol. 3156, pp. 16–29, Berlin, Heidelberg: Springer Berlin Heidelberg, 2004.

[101] K. Pearson, "Note on Regression and Inheritance in the Case of Two Parents," *Proceedings of the Royal Society of London Series I*, vol. 58, pp. 240–242, Jan. 1895. ADS Bibcode: 1895RSPS...58..240P.

[102] W. Kirch, ed., *Pearson's Correlation Coefficient*, pp. 1090–1091. Dordrecht: Springer Netherlands, 2008.

[103] X. Xia, B. Chen, and W. Zhong, "Correlation Power Analysis of Lightweight Block Cipher Algorithm LiCi," *Journal of Physics: Conference Series*, vol. 1972, p. 012055, July 2021.

[104] J.-W. Lee, S.-C. Chung, H.-C. Chang, and C.-Y. Lee, "An Efficient Countermeasure against Correlation Power-Analysis Attacks with Randomized Montgomery Operations for DF-ECC Processor," in *Cryptographic Hardware and Embedded Systems – CHES 2012* (D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, E. Prouff, and P. Schaumont, eds.), vol. 7428, pp. 548–564, Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. Series Title: Lecture Notes in Computer Science.

[105] P. Hodgers, N. Hanley, and M. O'Neill, "Pre-processing power traces to defeat random clocking countermeasures," in *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, (Lisbon, Portugal), pp. 85–88, IEEE, May 2015.

[106] G. Goodwill, B. Jun, J. Jaffe, and P. Rohatgi, "A testing methodology for side-channel resistance validation," *Cryptography Research Inc.*, p. 15, 2011.

[107] B. L. Welch, "The Generalization of 'Student's' Problem When Several Different Population Variances are Involved," *Biometrika*, vol. 34, no. 1-2, pp. 28–35, 1947.

[108] K. Pearson, "X. *On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling*," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 50, pp. 157–175, July 1900.

[109] A. Moradi, B. Richter, T. Schneider, and F.-X. Standaert, "Leakage Detection with the x2-Test," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 209–237, Feb. 2018.

[110] F. Wilcoxon, "Individual Comparisons by Ranking Methods," *Biometrics Bulletin*, vol. 1, p. 80, Dec. 1945.

[111] M. L. McHugh, "The Chi-square test of independence," *Biochemia Medica*, pp. 143–149, 2013.

[112] K. Papagiannopoulos, O. Glamočanin, M. Azouaoui, D. Ros, F. Regazzoni, and M. Stojilović, "The Side-channel Metrics Cheat Sheet," *ACM Computing Surveys*, vol. 55, pp. 216:1–216:38, Feb. 2023.

[113] I.-C. A. Chiang, R. S. Jhangiani, and P. C. Price, "Understanding Null Hypothesis Testing," Oct. 2015. Book Title: Research Methods in Psychology - 2nd Canadian Edition Publisher: BCcampus.

[114] W. Unger, L. Babinkostova, M. Borowczak, and R. Erbes, "Side-channel Leakage Assessment Metrics: A Case Study of GIFT Block Ciphers," in *2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, (Tampa, FL, USA), pp. 236–241, IEEE, July 2021.

[115] A. Jayasena, E. Andrews, and P. Mishra, "TVLA*: Test Vector Leakage Assessment on Hardware Implementations of Asymmetric Cryptography Algorithms," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 31, pp. 1269–1279, Sept. 2023.

[116] L. Lathrop, "Differential Power Analysis Attacks on Different Implementations of AES with the ChipWhisperer Nano," 2020. Publication info: Preprint. MINOR revision.

[117] L. Mather, E. Oswald, J. Bandenburg, and M. Wójcik, "Does My Device Leak Information? An a priori Statistical Power Analysis of Leakage Detection Tests," in *Advances in Cryptology - ASIACRYPT 2013*, vol. 8269, pp. 486–505, Berlin, Heidelberg: Springer Berlin Heidelberg, 2013.

[118] The Sage Developers, *SageMath, the Sage Mathematics Software System (Version 9.5)*, 2022. `https://www.sagemath.org`.

[119] P. S. Barreto and V. Rijmen, "The ANUBIS Block Cipher."

[120] D. Kwon, J. Kim, S. Park, S. H. Sung, Y. Sohn, J. H. Song, Y. Yeom, E.-J. Yoon, S. Lee, J. Lee, S. Chee, D. Han, and J. Hong, "New Block Cipher: ARIA," in *Information Security and Cryptology - ICISC 2003* (J.-I. Lim and D.-H. Lee, eds.), Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 432–445, Springer, 2004.

[121] B. S. University, "Data Encryption and Integrity Algorithms," *Information Technologies*, 2011.

[122] D. Wagner, B. Schneier, and J. Kelsey, "Cryptanalysis of the cellular message encryption algorithm," in *Advances in Cryptology — CRYPTO '97*, vol. 1294, pp. 526–537, Berlin, Heidelberg: Springer Berlin Heidelberg, 1997. Series Title: Lecture Notes in Computer Science.

[123] J. Schejbal, "Reverse engineering of CHIASMUS from GSTOOL," 0100.

[124] T. Shirai, K. Shibutani, T. Akishita, S. Moriai, and T. Iwata, "The 128-Bit Blockcipher CLEFIA (Extended Abstract)," in *Fast Software Encryption* (A. Biryukov, ed.), Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 181–195, Springer, 2007.

[125] C. H. Lim, "Crypton: A new 128-bit block cipher - specification and analysis," 1998.

[126] C. H. Lim, "A Revised Version of CRYPTON: CRYPTON V1.0," in *Fast Software Encryption* (L. Knudsen, ed.), Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 31–45, Springer, 1999.

[127] J. Stern and S. Vaudenay, "CS-Cipher," in *Fast Software Encryption*, vol. 1372, pp. 189–204, Berlin, Heidelberg: Springer Berlin Heidelberg, 1998. Series Title: Lecture Notes in Computer Science.

[128] R.-P. Weinmann and K. Wirt, "Analysis of the DVB Common Scrambling Algorithm," in *Communications and Multimedia Security* (D. Chadwick and B. Preneel, eds.), IFIP — The International Federation for Information Processing, (Boston, MA), pp. 195–207, Springer US, 2005.

[129] M. Becker and A. Desoky, "A study of the DVD content scrambling system (CSS) algorithm," in *Proceedings of the Fourth IEEE International Symposium on Signal Processing and Information Technology, 2004.*, pp. 353–356, Dec. 2004.

[130] W. Wu, L. Zhang, and X. Yu, "The DBlock family of block ciphers," *Science China Information Sciences*, vol. 58, pp. 1–14, Mar. 2015.

[131] M. Kanda, S. Moriai, K. Aoki, H. Ueda, Y. Takashima, K. Ohta, and T. Matsumoto, "E2 - A new 128-bit block cipher," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E83-A, pp. 48–59, Jan. 2000.

[132] D. Watanabe, S. Furuya, H. Yoshida, K. Takaragi, and B. Preneel, "A New Keystream Generator MUGI," in *Fast Software Encryption* (J. Daemen and V. Rijmen, eds.), Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 179–194, Springer, 2002.

[133] V. Grosso, G. Leurent, F.-X. Standaert, and K. Varıcı, "LS-Designs: Bitslice Encryption for Efficient Masked Software Implementations," in *Fast Software Encryption*, vol. 8540, pp. 18–37, Berlin, Heidelberg: Springer Berlin Heidelberg, 2015. Series Title: Lecture Notes in Computer Science.

[134] P. Junod and S. Vaudenay, "FOX : A New Family of Block Ciphers," in *Selected Areas in Cryptography*, vol. 3357, pp. 114–129, Berlin, Heidelberg: Springer Berlin Heidelberg, 2004. Series Title: Lecture Notes in Computer Science.

[135] F.-X. Standaert, G. Piret, G. Rouvroy, J.-J. Quisquater, and J.-D. Legat, "ICEBERG : An Involutional Cipher Efficient for Block Encryption in Reconfigurable Hardware," in *Fast Software Encryption* (B. Roy and W. Meier, eds.), Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 279–298, Springer, 2004.

[136] Anonymous, "Iraqi block cipher," June 2023. Page Version ID: 1158654835.

[137] R. Oliynykov, I. Gorbenko, O. Kazymyrov, V. Ruzhentsev, O. Kuznetsov, Y. Gorbenko, A. Boiko, O. Dyrda, V. Dolgov, and A. Pushkaryov, "A New Standard of Ukraine: The Kupyna Hash Function," 2015. Publication info: Preprint.

[138] P. S.L.M and V. Rijmen, "The khazad legacy-level block cipher," 2001.

[139] V. Dolmatov, "GOST R 34.12-2015: Block Cipher "Kuznyechik"," Request for Comments RFC 7801, Internet Engineering Task Force, Mar. 2016. Num Pages: 14.

[140] A. Adomnicai, T. P. Berger, C. Clavier, J. Francq, Paul, Huynh, V. Lallemand, K. L. Gouguec, M. Minier, L. Reynaud, and G. Thomas, "Lilliput-ae : a new lightweight tweakable block cipher for authenticated encryption with associated data submission to the nist lightweight cryptography standardization process," 2019.

[141] B. Kaliski, "The MD2 Message-Digest Algorithm," Request for Comments RFC 1319, Internet Engineering Task Force, Apr. 1992. Num Pages: 17.

[142] R. Scott, "Wide-Open Encryption Design Offers Flexible Implementations," *Cryptologia*, vol. 9, pp. 75–91, Jan. 1985. Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/0161-118591859799.

[143] G. Piret, T. Roche, and C. Carlet, "PICARO – A Block Cipher Allowing Efficient Higher-Order Side-Channel Resistance," in *Applied Cryptography and Network Security* (F. Bao, P. Samarati, and J. Zhou, eds.), Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 311–328, Springer, 2012.

[144] J. L. Massey, "SAFER K-64: A byte-oriented block-ciphering algorithm," in *Fast Software Encryption* (R. Anderson, ed.), Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 1–17, Springer, 1994.

[145] J. Yoon, S. Lee, d. h. cheon, J. Lee, and H. Lee, "The SEED Encryption Algorithm," Request for Comments RFC 4269, Internet Engineering Task Force, Dec. 2005. Num Pages: 16.

[146] C. Beierle, J. Jean, S. Kölbl, G. Leander, A. Moradi, T. Peyrin, Y. Sasaki, P. Sasdrich, and S. M. Sim, "The SKINNY Family of Block Ciphers and its Low-Latency Variant MANTIS," 2016. Publication info: A major revision of an IACR publication in CRYPTO 2016.

[147] E. Andreeva, V. Lallemand, A. Purnal, R. Reyhanitabar, A. Roy, and D. Vizar, "ForkAE v.1," 2019.

[148] T. Iwata, M. Khairallah, K. Minematsu, and T. Peyrin, "Remus v1.0," *National Institute of Standards and Technology*, 2019.

[149] T. Iwata, M. Khairallah, K. Minematsu, and T. Peyrin, "Romulus v1.0," *National Institute of Standards and Technology*, 2019.

[150] N. Siddiqui, F. Yousaf, F. Murtaza, M. Ehatisham-ul Haq, M. U. Ashraf, A. M. Alghamdi, and A. S. Alfakeeh, "A highly nonlinear substitution-box (S-box) design using action of modular group on a projective line over a finite field," *PLoS ONE*, vol. 15, p. e0241890, Nov. 2020.

[151] F. Liu, W. Ji, L. Hu, J. Ding, S. Lv, A. Pyshkin, and R.-P. Weinmann, "Analysis of the SMS4 Block Cipher," in *Information Security and Privacy* (J. Pieprzyk, H. Ghodosi, and E. Dawson, eds.), Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 158–170, Springer, 2007.

[152] P. Barreto and V. Rijmen, "The Whirlpool hashing function," *First Open NESSIE Workshop*, vol. 24, Jan. 2003.

[153] B. Gérard, V. Grosso, M. Naya-Plasencia, and F.-X. Standaert, "Block Ciphers That Are Easier to Mask: How Far Can We Go?," in *Cryptographic Hardware and Embedded Systems - CHES 2013* (G. Bertoni and J.-S. Coron, eds.), Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 383–399, Springer, 2013.

[154] ETSI/Sage, "Specification of the 3GPP Confidentiality and Integrity Algorithms 128-EEA3 & 128-EIA3," *Document 4: Design and Evaluation Report*, 2011.

[155] C. O'Flynn and Z. D. Chen, "Chipwhisperer: An open-source platform for hardware embedded security research," in *International Workshop on Constructive Side-Channel Analysis and Secure Design*, pp. 243–260, Springer, 2014.

[156] N. T. Inc., "Dpa on firmware implementation of aes," Accessed 2023. Accessed on September 29, 2023.

[157] kokke, "tiny-aes-c: A small portable aes128/192/256 in c," Accessed 2023. Accessed on September 29, 2023.

[158] NewAE Technology, *CW1101: Chipwhisperer-Nano*, 5 2019.

[159] NewAE Technology, *CW1173: Chipwhisperer-Lite*, 2 2018.

[160] NewAE Technology, *CW308: UFO Target Board*, 2 2018.

[161] Atmel, *Atmel AVR SMEGA Microcontrollers*, 2012.

[162] ARM, "STM32F ARM Microcontrollers - MCU Datasheets – Mouser," 2024.

[163] ARM, *STM32F303xB STM32F303xC*, 8 2018.

[164] C. Boura and A. Canteaut, "On the boomerang uniformity of cryptographic sboxes," *IACR Transactions on Symmetric Cryptology*, pp. 290–310, 2018.

[165] M. Abrar, "Assessment of block cipher using various tests," Accessed 2024. Accessed on February 12, 2024.

[166] C. O'Flynn and A. Dewar, "Chipwhisperer - the complete open-source toolchain for side-channel power analysis and glitching attacks," Accessed 2024. Accessed on March 12, 2024.

[167] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[168] T. pandas development team, "pandas-dev/pandas: Pandas," Feb. 2020.

[169] Wes McKinney, "Data Structures for Statistical Computing in Python," in *Proceedings of the 9th Python in Science Conference* (Stéfan van der Walt and Jarrod Millman, eds.), pp. 56 – 61, 2010.

[170] R Core Team, *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2024.

[171] H. Wickham, *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016.

[172] M. Kuhn, *modeldata: Data Sets Useful for Modeling Examples*, 2024. R package version 1.3.0.

[173] J. Felicisimo Villarico Wenceslao, "Enhancing the Performance of the Advanced Encryption Standard (AES) Algorithm Using Multiple Substitution Boxes," *International Journal of Communication Networks and Information Security (IJCNIS)*, vol. 10, no. 3, 2018. Number: 3.