12-13-2022

Side-Channel Attacks and Countermeasures for the MK-3 Authenticated Encryption Scheme

Peter Fabinski
pnf9945@rit.edu

Side-Channel Attacks and Countermeasures for the MK-3 Authenticated Encryption Scheme

Peter Fabinski

Side-Channel Attacks and Countermeasures for the MK-3 Authenticated Encryption Scheme

Peter Fabinski

December 13, 2022

A Thesis Submitted
in Partial Fulfillment
of the Requirements for the Degree of
Master of Science
in
Computer Engineering

RIT | Kate Gleason College of Engineering

Department of Computer Engineering

Side-Channel Attacks and Countermeasures for the MK-3 Authenticated Encryption Scheme

Peter Fabinski

Committee Approval:

Dr. Marcin Łukowiak *Advisor* Date
Department of Computer Engineering

Dr. Cory Merkel Date
Department of Computer Engineering

Dr. Stanisław Radziszowski Date
Department of Computer Science

Dr. Michael Kurdziel Date
L3Harris Technologies

Acknowledgments

First and foremost, I would like to thank my advisor Marcin Łukowiak for his advice and guidance, both on this thesis and outside of it. I still remember being told to draw my diagrams before starting to code, and how from that point on, every design worked the very first time. This advice has served me well ever since.

I also appreciate the help I've received from the rest of the team, providing valuable feedback on my work and asking the questions I needed to hear. Mike Kurdziel and Steve Farris have been valuable resources on the practical side of things, and it has been great knowing that my research will be of use. Thanks to L3Harris for their financial support of the project.

My research would not have been possible without the hard work of Dan Stafford, whose detailed documentation was a great help, and Matthew Millar, who taught me to use the ChipWhisperer. I'm also very grateful to Mark Indovina, who not only helped me with the infrastructure on this project, but also whose class I'm sure got me the job I have today. Thanks to Maximiliano Cristiá for his generous help with Setlog as well.

Finally, I am incredibly grateful to my friends and family for all their help and support. I am especially thankful for Bob Fabinski's hours helping me think through problems and Qingqing Yang's time spent working along with me to stay on task.

This has been a great experience I will never forget.

To my wonderful family. Thank you for your endless support and encouragement.

Abstract

In the field of cryptography, the focus is often placed on security in a mathematical or information-theoretic sense; for example, cipher security is typically evaluated by the difficulty of deducing the plaintext from the ciphertext without knowledge of the key. However, once these cryptographic schemes are implemented in electronic devices, another class of attack presents itself. Side-channel attacks take advantage of the side effects of performing a computation, such as power consumption or electromagnetic emissions, to extract information outside of normal means. In particular, these side-channels can reveal parts of the internal state of a computation. This is important because intermediate values occurring during computation are typically considered implementation details, invisible to a potential attacker. If this information is revealed, then the assumptions of a non-side-channel-aware security analysis based only on inputs and outputs will no longer hold, potentially enabling an attack.

This work tests the effectiveness of power-based side-channel attacks against MK-3, a customizable authenticated encryption scheme developed in a collaboration between RIT and L3Harris Technologies. Using an FPGA platform, Correlation Power Analysis (CPA) is performed on several different implementations of the algorithm to evaluate their resistance to power side-channel attacks. This method does not allow the key to be recovered directly; instead, an equivalent 512-bit intermediate state value is targeted. By applying two sequential stages of analysis, a total of between 216 and 322 bits are recovered, dependent on customization parameters. If a 128-bit key is used, then this technique has no benefit to an attacker over brute-forcing the key itself; however, in the case of a 256-bit key, CPA may provide up to a 66-bit advantage. In order to completely defend MK-3 against this type of attack, several potential countermeasures are discussed at the implementation, design, and overall system levels.

Contents

List of Figures

List of Tables

Acronyms

AE Authenticated Encryption

AES Advanced Encryption Standard

ASCII American Standard Code for Information Interchange

ASIC Application-Specific Integrated Circuit

CMOS Complementary Metal-Oxide Semiconductor

CPA Correlation Power Analysis

CPU Central Processing Unit

DDL Dynamic and Differential Logic

DPA Differential Power Analysis

DRP Dual-Rail Precharge

DVFS Dynamic Voltage and Frequency Scaling

DWDDL Divided Wave Dynamic Differential Logic

EM Electromagnetic

FCR Field Customization Register

FPGA Field-Programmable Gate Array

HAC Holistic Assessment Criterion

IV Initialization Vector

MAC Message Authentication Code

NIST National Institute of Standards and Technology

RIT Rochester Institute of Technology

RSA Rivest-Shamir-Adleman

RTL Register-Transfer Level

SABL Sense Amplifier Based Logic

S-box Substitution Box

SDDL Simple Dynamic Differential Logic

SHA-3 Secure Hash Algorithm 3

SPA Simple Power Analysis

TDPL Three-phase Dual-rail Precharge Logic

TI Threshold Implementation

TVLA Test Vector Leakage Assessment

VHDL VHSIC Hardware Description Language

VHSIC Very High Speed Integrated Circuit

WDDL Wave Dynamic Differential Logic

XOR Exclusive OR

Chapter 1

Introduction

1.1 Motivation

One of the most effective ways to exploit any kind of system is to approach it from a viewpoint other than that of the original designer. This strategy often exposes weaknesses which were previously overlooked and have therefore not been protected against. Side-channel attacks exploit this type of vulnerability, taking advantage of information unintentionally transmitted through channels outside of the intended interface for the system. This information can often reveal critical secrets, allowing even a system which is perfectly secure in its original context to be attacked with relative ease.

Cryptography is one of the largest areas to which side-channel attacks have been applied; specifically, hardware implementations of cryptographic algorithms often exhibit side-channel vulnerabilities. Many encryption schemes have been successfully exploited using side-channel attacks, inspiring a significant amount of research both on new methods for performing these attacks as well as on how they can be prevented. Protection from side-channel attacks is achieved by applying algorithmic and hardware-based modifications, called countermeasures, to the encryption algorithms and their implementations [2, 3].

In general, there are many different side-channels which can be used to attack a

system, such as the timing of various operations [2], electromagnetic emissions [2,4,5], the results of inducing errors [6], or even sound. However, when studying electronic encryption, power analysis is by far the most commonly-used type of side-channel. In this form of side-channel attack, a device is made to perform a series of cryptographic operations, and the power usage of the device during this process is recorded. Because of the electrical properties of digital logic, the power usage of the device is related to the data being used for the computations, including secret information such as an encryption key. Therefore, by deducing the nature of this relationship, an attacker can analyze the collected power data and obtain the secret information. If this is an encryption key, for example, then the attacker becomes able to decrypt other messages encrypted using the same key and cipher configuration. Additionally, in some cases, messages can be captured, modified, re-encrypted, and then passed on to the intended recipient without their knowledge.

There are two important functions which can be provided by an encryption scheme. The first is confidentiality; this is the typical notion of encryption, which allows only those who are in possession of the key to decrypt and read the encrypted message. However, if an attacker is able to determine the key, they can then modify or produce new messages without the destination being able to tell the difference. This issue is resolved by the second function of authentication. Authentication, in contrast with confidentiality, does not prevent the message from being read. Instead, it ensures that a messages originates from the purported sender and that it has not been replaced or modified along the way.

Many encryption schemes exist which provide both confidentiality and authentication; these are called Authenticated Encryption (AE) schemes. One such AE scheme, which will be the focus of this work, is the MK-3 scheme, proposed in [1, 7] by Rochester Institute of Technology (RIT) and L3Harris Technologies. This scheme has a number of advantages; it provides both encryption and authentication in a sin-

gle pass, is designed with a focus on hardware implementation, and provides a unique customization architecture allowing many non-interoperable instances of the same design to be used in different organizations or applications. However, compared with other encryption schemes such as the Advanced Encryption Standard (AES), MK-3 has had little research performed on its side-channel attack resistance. In addition, MK-3 makes use of the sponge construction, a design for cryptographic algorithms. Because this construction is relatively new, there is less research on its resistance to side-channel attacks in general compared to older constructions. The only analysis so far of MK-3 regarding side-channel attacks was done by Daniel Stafford in [8], in which simulation was used to attempt power analysis attacks and a survey was performed of potential countermeasures which could be applied.

1.2 Contributions

This work investigates the MK-3 algorithm's performance against hardware-based side-channel attacks, as well as potential methods by which it can be protected. The first goal was to determine the side-channel resistance of the MK-3 algorithm with no countermeasures applied. This was done by implementing the MK-3 algorithm in hardware on a field-programmable gate array (FPGA); in comparison with simulation, hardware testing provides an environment much closer to that which would be encountered in real-world applications. Using this implementation, Correlation Power Analysis (CPA) side-channel attacks were performed against the algorithm by collecting power usage data and attempting to perform the attacks directly. The evaluation criterion for these attacks is the amount of power data required to successfully perform the attack; a more effective attack requires less data to extract the desired information, while a less effective attack requires more data or can only provide partial information. In addition to directly performing power attacks, the theoretical limitations of the developed attacks were determined, taking into account

the customization parameters supported by MK-3.

The second goal was to determine potential countermeasures and design techniques which can be applied to the MK-3 algorithm in order to increase its resistance to side-channel attacks. A survey of current implementation-level, algorithm-level, and system-level countermeasures was performed, considering their use in both application-specific integrated circuit (ASIC)-based and FPGA-based designs. Based on this survey, as well as analysis of the FPGA implementation, several design recommendations were developed for protecting MK-3 against power analysis attacks. This includes aspects of physical product design, system design, customization, and hardware implementation. A framework for performing Test Vector Leakage Assessment (TVLA) was also developed to aid in the evaluation of any hardware countermeasures implemented in the future.

The rest of this document is organized as follows. Chapter 2 begins with a detailed description of the MK-3 encryption algorithm and its customization in Section 2.1, followed by an overview of side-channels, power analysis attacks, and leakage assessment in Sections 2.2 through 2.4. Section 2.5 then provides the details of many different power analysis countermeasures at various levels of design. Next, Chapter 3 describes the methods of implementation, data collection, and data analysis used to evaluate the algorithm and perform hardware power attacks. Chapter 4 provides the results of these methods, including the theoretical limitations of the developed attacks. Finally, Chapter 5 concludes the document, providing security recommendations for field applications of MK-3 and ideas for future work.

Chapter 2

Background

2.1 The MK-3 Algorithm

The MK-3 encryption algorithm was developed in a collaboration between L3Harris Technologies and RIT, and was first published in Matthew Kelly's master's thesis [1]. As described in Section 1.1, MK-3 is an authenticated encryption algorithm, meaning that it provides both confidentiality and authenticity.

2.1.1 Sponge Construction

The MK-3 algorithm is based on a version of the sponge construction, which was popularized by Keccak, the winner of the National Institute of Standards and Technology (NIST)'s SHA-3 competition [9, 10]. This construction, as used in MK-3, is based on two main components. One is the 512-bit state, which is divided into the rate (128 bits) and the capacity (384 bits). These 512 bits are also referred to using 32 16-bit partitions called words [1, 11]. The other main component is the permutation function f, which is a bijective function applied to the 512 bits of the state.

In the standard sponge construction, there are two separate phases of functionality called absorbing and squeezing, as shown in Figure 2.1 [10]. First is the absorbing phase, in which inputs such as key or message blocks are absorbed into the state. Each absorbed block consists of 128 bits. In each absorbing operation, the input block is

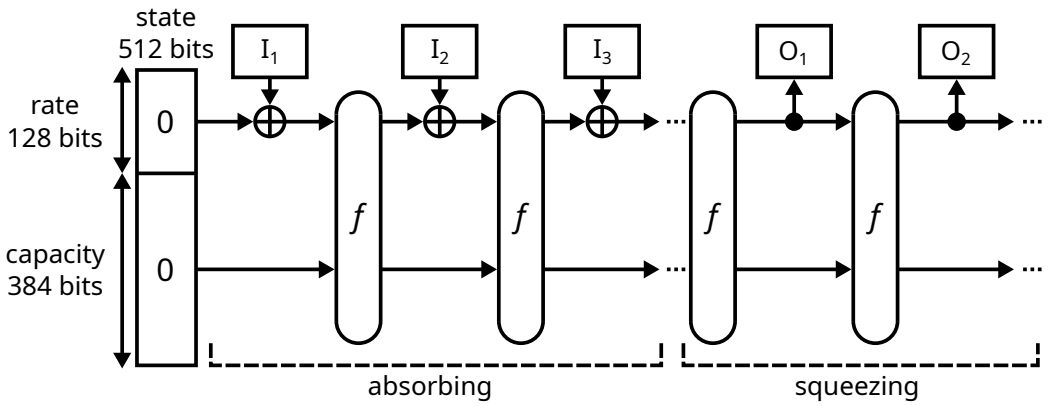

Figure 2.1: Example of the sponge construction.

first combined with the rate portion of the state using an exclusive-or (XOR); the entire state is then permuted by the function f, mixing in the new information and making the state dependent on all the inputs that have been absorbed so far. This continues until all input blocks have been absorbed, at which point the squeezing phase begins. In this phase, the output block from each squeezing operation is the rate portion of the current state. After each output block is produced, the permutation function f is applied to bring out new information into the rate and continue transforming the state as a whole. By continuing to perform squeezing operations, an output of any desired length can be produced.

This construction has applications such as hash functions, message authentication codes (MACs), and stream ciphers; however, because the absorbing and squeezing phases are separated, multiple passes are required to provide authenticated encryption. Instead, the MK-3 algorithm uses a modified version of the sponge construction called the duplex sponge [1, 10]. The main difference between the duplex sponge construction and the standard sponge construction is the addition of a duplexing operation, which allows both absorption and squeezing to be performed in one round. Figure 2.2 shows a diagram of the duplex sponge construction as used in MK-3.

As seen in this diagram, each of the absorbing, squeezing, and duplexing operations are used in several different stages. First, a key K and initialization vector IV

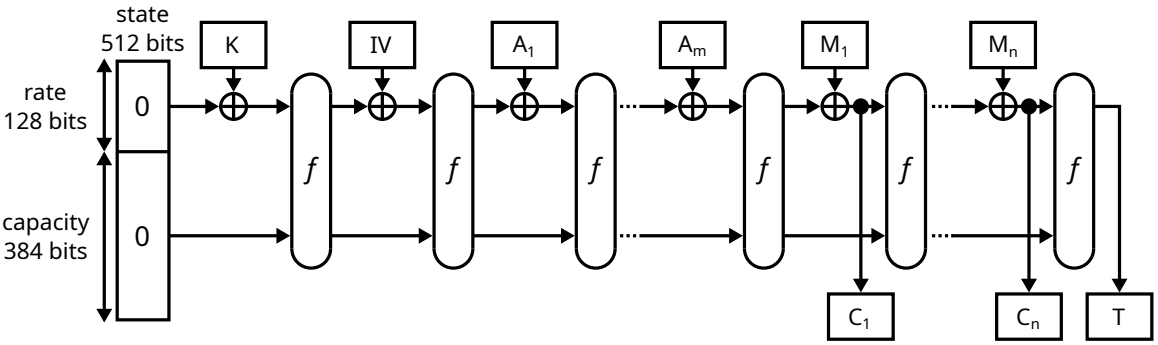

Figure 2.2: Duplex sponge construction used in the MK-3 algorithm (128-bit key).

are absorbed, followed by m blocks of additional authenticated data $A_1 - A_m$ such as headers. In the case of a 256-bit key, two 128-bit key blocks are absorbed sequentially before the IV. The additional authenticated data is not provided confidentiality, and must be known at the time of decryption. The purpose of absorbing this data here is to incorporate it into the state, ensuring that the final authentication tag will not match if it has been modified in transit. Next, the message is absorbed, in 128-bit blocks as before; however, in the duplex sponge construction, ciphertext blocks are also produced each round by an XOR of the plaintext and rate. This is the duplexing operation, which combines absorbing and squeezing into one action [1, 10]. Lastly, a final squeezing operation with no additional input produces the tag. The value of this tag is dependent on all previously-absorbed data, including the key, and thus provides authentication for the encrypted message.

2.1.2 Architecture of MK-3

As described in Section 2.1.1, the high-level design of MK-3 is based on the sponge construction. Therefore, the main contribution of MK-3 itself is the permutation function f, composed of several iterations of a round function g. Within a single application of the overall function f, g is applied repeatedly in a number of rounds dependent on the key length; for a 128-bit key, 10 rounds are used, and for a 256-bit key, 16 rounds are used. The round function g has four stages, as shown in Figure 2.3.

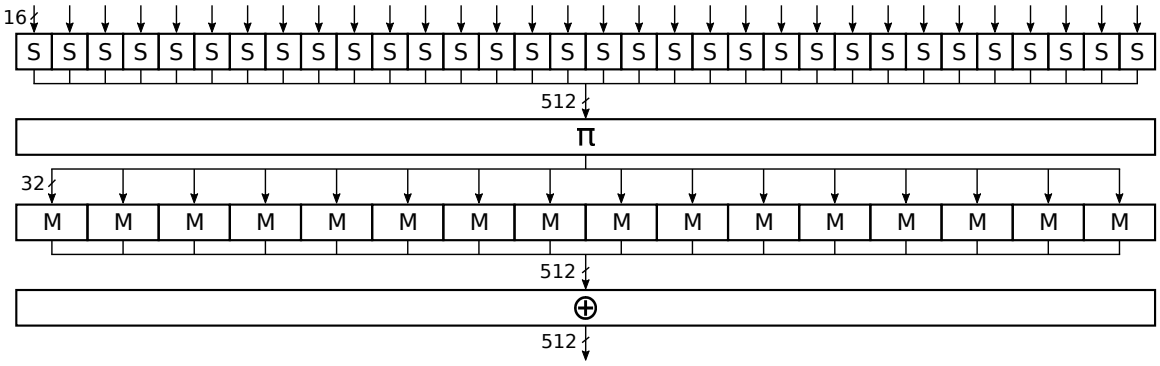

Figure 2.3: Diagram of the MK-3 round function g. [1]

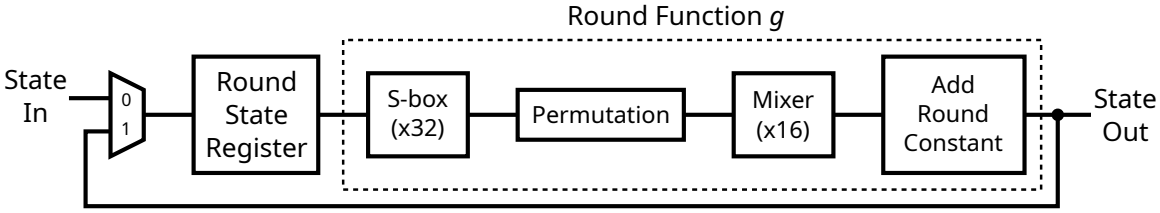

Figure 2.4: Block diagram of the MK-3 permutation function f.

Pseudocode is presented in Algorithm 1 [12] to give a concise definition of the overall permutation function f, along with the diagram shown in Figure 2.4. Sections 2.1.2.1 through 2.1.2.4 then provide detailed explanations of its functionality.

2.1.2.1 Substitution Stage

First is the substitution stage, which provides nonlinearity to the round function. This stage is composed of 32 identical 16-bit bijective substitution boxes (S-boxes); instead of a random mapping from inputs to outputs, these S-boxes use a design based on the computation of a multiplicative inverse in a Galois field followed by an affine transformation [13, 14]. The use of a known function instead of a random mapping allows S-box results to be computed on the fly, instead of implementing a resource-intensive lookup table. Thanks to this method of S-box construction, the MK-3 algorithm is able to use S-boxes with 16-bit inputs and outputs, which are believed at the time of writing to be the largest used in an encryption algorithm to date [1, 13]. In the standard MK-3 implementation, each S-box computes the multiplicative inverse of its input over the Galois field $GF(2^{16})$ modulo the irreducible

Algorithm 1 MK-3 Permutation Function f [12]

$N_r \leftarrow 10$ (or 16) ▷ number of rounds
$N_b \leftarrow 512$ ▷ number of bits in state
$N_w \leftarrow 32$ ▷ number of 16-bit words in state
function $f(S)$
    for $r \leftarrow 1, N_r$ **do**
        $g(S)$ ▷ permute state S in place
    end for
end function
function $g(S)$
    for $i \leftarrow 0, N_w - 1$ **do** ▷ substitution step
        $S.word[i] \leftarrow \text{Sbox}(S.word[i])$
    end for
    $S' \leftarrow S.copy()$
    for $b \leftarrow 0, N_b - 1$ **do** ▷ permutation step
        $b' \leftarrow (31b + 15) \mod 512$
        $S_b \leftarrow S'_{b'}$
    end for
    for $j \leftarrow 0, N_w/2 - 1$ **do** ▷ mixing step
        $i \leftarrow 2j$
        $(S.word[i], S.word[i+1])$
            $\leftarrow mix(S.word[i], S.word[i+1])$
    end for
    $S \leftarrow S \oplus RC[r]$ ▷ add round constant step
end function

polynomial $p(x) = x^{16} + x^5 + x^3 + x + 1$, followed by an affine transformation on the result, as shown in Equation 2.1.

$$\text{Sbox}(x) = \begin{bmatrix} 0&0&1&0&0&0&0&1&0&0&1&1&1&1&1&0 \\ 1&1&0&0&0&0&0&1&0&1&1&0&1&0&1&0 \\ 1&1&0&0&1&0&1&1&0&1&0&1&0&0&1&1 \\ 1&1&1&0&0&0&1&0&0&1&1&0&0&0&0&0 \\ 1&1&0&0&0&1&1&0&0&1&1&1&1&0&1&1 \\ 0&1&0&0&0&0&1&1&0&1&1&1&1&1&0&1 \\ 0&0&1&0&1&0&1&0&1&1&0&0&1&1&0&0 \\ 1&0&1&1&1&0&1&1&0&0&0&1&0&1&1&1 \\ 0&1&0&0&0&0&0&0&1&0&0&1&1&1&0&1 \\ 1&0&1&1&0&0&0&1&0&0&1&0&1&0&0&0 \\ 1&0&1&0&0&1&1&1&0&0&1&1&0&1&0&0 \\ 1&0&1&1&1&0&1&1&1&1&0&1&1&0&0&1 \\ 1&0&1&0&0&1&0&1&1&0&0&1&0&0&0&1 \\ 0&1&0&0&0&1&1&1&1&0&0&0&0&0&0&1 \\ 1&0&0&0&1&1&0&1&0&1&1&1&1&0&0&0 \\ 1&1&0&1&0&1&1&0&1&0&0&1&1&0&0&0 \end{bmatrix} \begin{bmatrix} x_{15} \\ x_{14} \\ x_{13} \\ x_{12} \\ x_{11} \\ x_{10} \\ x_9 \\ x_8 \\ x_7 \\ x_6 \\ x_5 \\ x_4 \\ x_3 \\ x_2 \\ x_1 \\ x_0 \end{bmatrix}^{-1} \oplus \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \end{bmatrix} \tag{2.1}$$

2.1.2.2 Permutation Stage

The next stage of g is the permutation stage. As described in [1], this stage provides diffusion to the round function across the entire 512-bit state. This is accomplished using a 512-bit bitwise permutation, defined by the function $\pi(x) = 31x + 15 \mod 512$. Specifically, the bit at position x in the output of the permutation stage is connected to the bit at position $\pi(x)$ in the input of the stage.

2.1.2.3 Mixing Stage

After the permutation stage is the mixing stage, which serves primarily to create diffusion on a smaller scale than the previous step. In this stage, the 512-bit state is distributed into 16 identical mixers, each of which has a 32-bit input and output. Each mixer operates by dividing the 32-bit input into 16-bit words A and B, and then performing the operation shown in Equation 2.2 to produce the output words A' and B' [1].

$$\begin{bmatrix} A' \\ B' \end{bmatrix} = \begin{bmatrix} 1 & x \\ x & x+1 \end{bmatrix} \begin{bmatrix} A \\ B \end{bmatrix} \tag{2.2}$$

This operation is performed in the Galois field $GF(2^{16})$, modulo the irreducible polynomial $q(x) = x^{16} + x^5 + x^3 + x^2 + 1$. The 32-bit output of the mixer is then the concatenation of A' and B'.

In $GF(2^{16})$, addition can be implemented using a single XOR operation; multiplication by x corresponds to a left shift followed by a modulo operation, which can also be implemented using a small number of XOR gates based on $q(x)$ as shown in Figure 2.5 [1]. Using these implementations of the mixer operations, an efficient hardware design for the mixer can be created. Figure 2.6 shows a diagram of this design, where $*x$ represents the multiplication-by-x operation and $\oplus$ represents a 16-bit XOR.

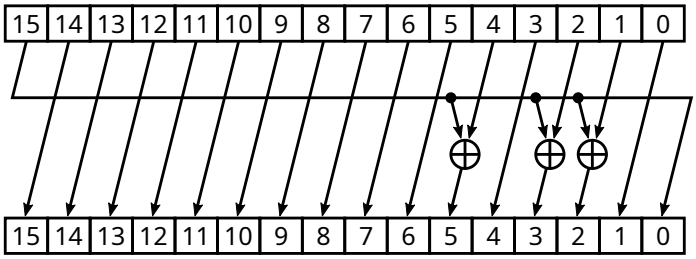

Figure 2.5: Multiplication by x in $GF(2^{16})$ modulo $q(x)$.

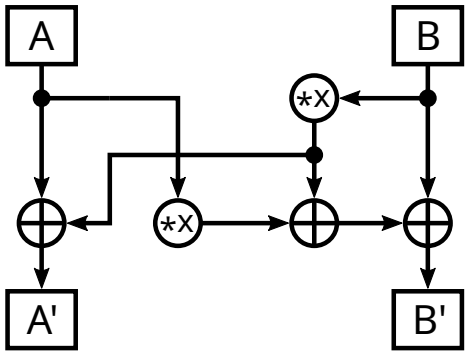

Figure 2.6: Diagram of mixer function for MK-3.

2.1.2.4 Add Round Constant Stage

The final stage of the round function g is the add round constant stage. This stage simply performs a bitwise XOR of all 512 bits from the mixing stage with a 512-bit round constant, which is produced by Keccak-512(ASCII(i)) where i is the current round number (starting at 1 for the first round) [9]. Note that due to a padding change during standardization, the SHA3-512 algorithm referenced in [1] and [7] does not produce the same results as Keccak-512 [9, 15].

2.1.3 MK-3 Customization

There are many ways to customize the MK-3 algorithm, each producing numerous versions which are not interoperable with one another [1, 7, 11]. The available customization modes can be divided into two categories: factory customization and field customization. Factory customizations are modifications to the permutation function which can be made by the manufacturer of the device implementing the encryption, while field customizations are modifications which can be made after the device has been produced and put to use.

Each of the four stages of the function can be modified by factory customizations, while the mixing stage can also be modified by field customizations.

2.1.3.1 Substitution Stage

In the substitution stage, the S-boxes can be customized by modifying both the irreducible polynomial $p(x)$ used to compute the multiplicative inverse, as well as the invertible matrix A and vector b which compose the affine transformation. In Kelly's and Wood's theses [1,13], no specific values are assumed for these parameters on their own when evaluating the security of the S-box design. Instead, the security analysis is based on several characteristics of the S-box itself, as described in [11]:

- Has no fixed points (no input results in itself)
- Has no opposite fixed points (no input results in its bitwise inverse)
- Has a maximum differential probability of 2^{-14}
- Has a maximum linear bias of 2^{-8}

Therefore, when developing a customization, any selection of $p(x)$, A, and b can be used, so long as the resulting S-box meets or exceeds these specifications.

2.1.3.2 Permutation Stage

Similarly, the security of the permutation stage is based on the following properties of the bitwise permutation π, specified in [1]:

- The output of each S-box is evenly distributed across all mixers
- No input position is mapped to the same position in the output (π is a derangement; π has no fixed points)
- When applied repeatedly, the original input pattern does not occur within the number of rounds of the permutation function f (π is of high order)
- No individual bit returns to its original position in fewer applications than it takes for the overall function to repeat (π has no low-order bits)

As with the substitution stage, any bitwise permutation can be used which satisfies these properties. Kelly's thesis [1] provides a listing of 384 candidate affine functions which fulfill all the requirements.

2.1.3.3 Mixing Stage

The mixing stage can be customized both at the factory and in the field. In both cases, the customization modifies the irreducible polynomial $q(x)$ used in each mixer. In total, there exist 4080 degree-16 irreducible polynomials; as described in [11], the security analysis of MK-3 performed in [1] does not assume any particular polynomial. Therefore, any polynomial can be selected without affecting the security of the algorithm.

At the factory, the first part of mixer customization is performed by selecting 256 of the 4080 possible irreducible polynomials and placing their coefficients into a lookup table. The mixers are then configured to use the results of this lookup table instead of a fixed polynomial.

Field customization of the mixers is then implemented using this lookup table. Specifically, a 128-bit Field Customization Register (FCR) is created, which provides separate 8-bit lookup table inputs for each mixer. This allows the irreducible polynomial for each mixer to be individually chosen from the 256 options selected at the factory, without making any modifications to the hardware of the device. Because every lookup table input maps to a valid irreducible polynomial, any of the possible values of the FCR will result in a valid mixing operation.

2.1.3.4 Add Round Constant Stage

Factory customization in the add round constant stage is fairly simple. The only requirements are that the constants are unique in each round, and that they exhibit no discernable structure [1, 11]. As long as these requirements are satisfied, any round constants may be used. One method recommended by [11] is the use of a cryptographic hash function to generate constants based on a counter incremented each round.

2.2 Side-Channel Attacks

A side-channel is any form of information produced by an operation other than its intended result; for example, the use of shared resources, the time between requesting and receiving a result, or the amount of energy used to produce a result. Frequently, side-channels can reveal information not available through typical means, such as the internal state of a device during computation.

In the context of encryption, intermediate values between the input and the output are rarely considered for security, unless for the purpose of preventing side-channel attacks. Therefore, if a side-channel exposes intermediate values in an unprotected implementation of an encryption algorithm, an attack could potentially be performed regardless of the algorithm's security in a traditional cryptographic sense.

2.2.1 Power Side-Channels

There are a variety of side-channels which can be used to extract information from a device. The most common and well-studied is power consumption, a side-channel based on the fact that the changing of electrical signals in a device consumes energy, and that the amount of energy is dependent on the particular changes taking place. This energy consumption can be observed from outside the device, revealing otherwise-inaccessible information about the data being operated on, as well as the nature and timing of the operations [2, 16, 17, 18]. Power analysis will be the focus of this work, and Section 2.3 describes several different attack methods which use power side-channel information.

2.2.2 Electromagnetic Side-Channels

As with power usage, the electromagnetic (EM) emissions produced by computations have a dependency on internal values and can be used to perform similar attacks. EM measurements and attacks are very non-invasive, and can potentially be used with no modifications at all to the target device [2, 4, 16]. However, in exchange, electromagnetic emissions may be more difficult to measure than power usage, often subject to significant interference and typically requiring close proximity with specialized measurement equipment.

2.2.3 Timing Side-Channels

Another common side-channel is timing, which is typically introduced by data-dependent execution. Specifically, when the execution path of a software program or a hardware circuit is determined based on a secret value, the difference in execution time for the different paths can potentially be detected from outside the system, revealing information about the secret value. There are a variety of ways to detect this time difference. The simplest is to measure the overall time period between providing

an input and receiving the corresponding output, which only requires observing the intended output channel. Paul Kocher's seminal paper [19] introduces this type of attack on systems using modular exponentiation and reduction.

Additionally, a simple example of timing attacks on a software level is an attack on a password verification routine which checks each character of the provided password sequentially against the corresponding character of the correct password. As soon as a character does not match, the routine knows that the given password is wrong, and reports a failure. Because the characters are checked one by one, an attacker can measure the time taken for the system to report an incorrect password to determine how many characters were checked, and therefore how many of the beginning characters are correct. Based on this information, an attack is trivial, as the required number of attempts changes from an exponential to a linear relationship (for example, 26^{10} becomes $26 * 10$ for a 10-character alphabetic password).

Timing is also very commonly used in conjunction with other side-channels. For instance, the timing of various internal steps in an algorithm could be identified based on the associated changes in power usage; this is the basis of Simple Power Analysis, discussed in Section 2.3.1. Many shared-resource side-channels also rely on timing to produce their results, as described in Section 2.2.4.

2.2.4 Shared-Resource Side-Channels

Shared-resource side-channels take advantage of systems where several different processes are performed using the same shared resource. An excellent example of this type of resource is the cache in a modern CPU, which holds the data from recently-accessed memory locations to allow faster operation. Notably, the organization of the cache is based only on partial memory addresses; this means that the access time for a given address will depend on whether or not other conflicting addresses have recently been accessed. The recent Meltdown attack [20], as well as one possible implementa-

tion of the more general Spectre attack [21], take advantage of speculative execution by first reading a protected memory location, then accessing a specific address dependent on the read value, before the relatively-costly memory permission checks are performed. Even though the speculatively executed read instruction is later cancelled when the access checks are completed, it has still modified which addresses are present in the cache. The resulting timing differences accessing those addresses later on can then be detected by another process; this is the side-channel, allowing information to pass between the normally-isolated speculative and non-speculative domains.

Another example of a common shared resource is a branch predictor, which could be manipulated to reveal information about the control flow of an unrelated process. An attack using this principle would be similar to the cache-based attack, forming a side-channel to learn the results of a speculative branch.

2.3 Power Analysis Methods

Depending on the system being targeted, there are many different ways of using information obtained through a power side-channel to perform a full-fledged power analysis attack. The following sections highlight several common methods.

2.3.1 Simple Power Analysis

Simple Power Analysis (SPA) is a timing-based power analysis attack. In SPA, a device's power consumption is measured during a single operation, producing a list of power values over time called a power trace. The timing characteristics of this power trace are then analyzed to identify regions with different amounts of power consumption, indicating that different sections of the computation are occurring [2, 16, 18]. This can be thought of as an expansion of the pure timing attacks in [19], as it provides more fine-grained information on the internal state as calculations occur, rather than a single overall metric.

Typically, SPA is applied to processes where the execution path is directly dependent on the desired secret information; the classic example is the RSA algorithm, during decryption of a message with an unknown private key exponent d. In the square-and-multiply process, each bit of d directly determines whether only a squaring operation or both a square and multiply operation take place. When a particular bit of d is 1, the corresponding square-and-multiply will produce a longer period of high power usage when compared to the shorter period for only the squaring operation. Because of this difference, the bits of d can be directly read from a power trace by identifying long and short periods of increased power usage [16].

2.3.2 Differential Power Analysis

Differential Power Analysis (DPA) is a more sophisticated method of power analysis than SPA, based on statistical analysis of power traces. It allows the target values to be determined by enumerating partial guesses of these values, and then determining which of the partial guesses is correct [2, 17]. The key difference between DPA and simple brute-forcing is that although both require enumerating possibilities for the target value, brute-forcing requires the entire value to be guessed at once, while DPA allows small portions of the value to be guessed independently. For example, instead of testing all 2^{128} possibilities for a 128-bit encryption key, DPA could allow a byte-by-byte approach, requiring only $16 * 2^8 = 2^{12}$ total guesses.

To perform DPA, power traces are collected from many encryptions, using the same unknown key but different plaintexts each time. Either the plaintexts or the resulting ciphertexts are considered to be known. The power traces are then divided into two subsets based on the result of a selection function; given a plaintext or ciphertext and a guess for part of the target value, the selection function computes a property which is expected to be true at some point in the encryption process [17, 18, 22]. This property is usually the value of a given bit of an intermediate value

in the computation. All traces in each subset are then averaged together, and the difference of these two averages is computed to produce a DPA trace for the given guess.

If the guessed value is incorrect, then the bit computed by the selection function will be different from that bit in the target device in approximately half of the traces in each set; therefore, the difference between the averages of the sets will be approximately zero. On the other hand, if the guessed value is correct, then the power traces will be correctly divided based on the value of the target bit. In this case, the effect that this bit has on power consumption will not be evenly distributed between the sets, causing their averages to differ. This difference causes the DPA trace for the correct guess to stand out from the others, revealing the corresponding part of the key [2, 17, 18]. Finally, this process can be repeated, guessing different sections of the key with appropriate selection functions, to eventually determine the full value.

2.3.3 Correlation Power Analysis

Similarly to DPA, Correlation Power Analysis (CPA) allows guesses to be verified separately for small parts of the target value. However, instead of dividing traces using a selection function, CPA develops a model for the expected power consumption of the target device given a particular guess g and known input value I_i [16, 23]. Using this model, the expected power usage at some point in the target operation is computed for a given guessed value across all collected inputs; then, the Pearson correlation coefficient between the modeled power and true power vectors is computed at each time sample of the traces, as shown in Equation 2.3.

$$\text{corr}(X, Y) = \rho_{X,Y} = \frac{\text{cov}(X, Y)}{\sigma_X \sigma_Y} = \frac{\mathbb{E}[(X - \mu_X)(Y - \mu_Y)]}{\sigma_X \sigma_Y} \tag{2.3}$$

As with DPA, the computations are then repeated for all possible values of the guessed portion. Figure 2.7 shows a visual representation of this process.

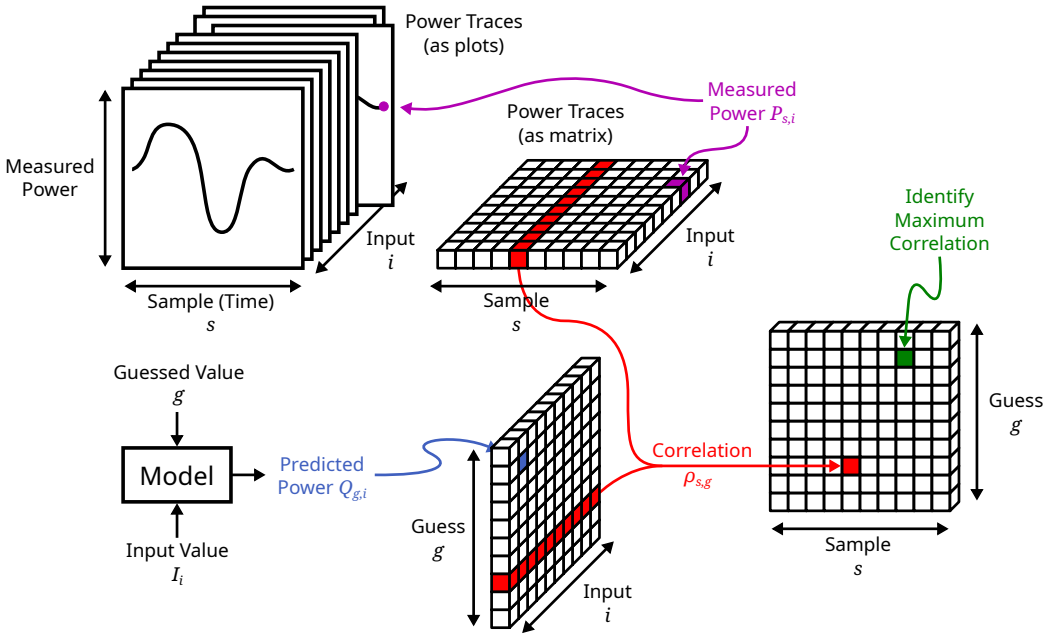

Figure 2.7: Diagram of correlation power analysis computations.

Once all correlation computations have been performed, the result is a matrix of correlations, one for each guessed value at each of the original time samples. If a particular guessed value is incorrect, then there will be minimal correlation between the actual and modeled power consumption across the entire time of the trace. However, if a guess is correct, there will be a visible increase in absolute correlation at the point in time where the modeled values appear in the physical computation. With enough traces, the correct guess will have a strong enough correlation that it is clearly separated from the other guesses. Specifically, the guess containing the highest peak correlation at any point in time is considered the correct guess for the current part of the target value [23]. Figure 2.8 illustrates how this selection process can be performed visually, by plotting the correlation of each guess over time.

Algorithm 2 describes the overall CPA process in pseudocode, including the final guess-selection step. The specifics of power modeling for MK-3 are described in Section 3.3.

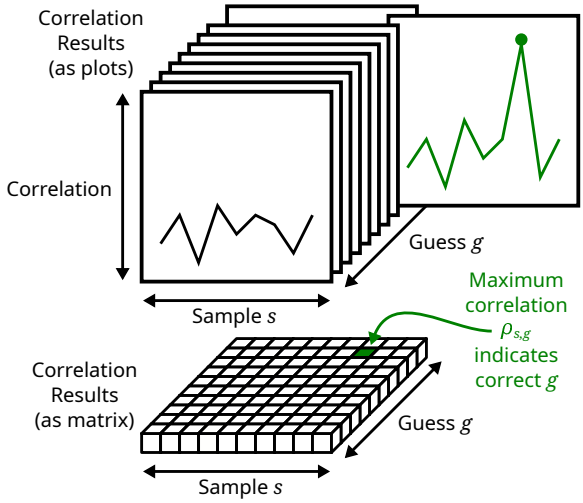

Figure 2.8: Visual guess selection process for CPA.

Algorithm 2 Correlation Power Analysis

$P_{s,i} \leftarrow$ measured power at sample s in trace i
$I_i \leftarrow$ known input used to produce trace i
$G \leftarrow$ set of input bit positions to guess
function CPA(P, I, G)
    for $i \leftarrow 1, |I|$ **do** ▷ model power usage
        for $g \leftarrow 0, 2^{|G|} - 1$ **do**
            $Q_{g,i} \leftarrow$ ModelPower(I_i, g)
        end for
    end for
    for $s \leftarrow 1, |P|$ **do** ▷ compute correlations
        for $g \leftarrow 0, 2^{|G|} - 1$ **do**
            $\rho_{s,g} \leftarrow \text{corr}(P_s, Q_g)$
        end for
    end for
    return $\text{argmax}_g(\max_s(\rho))$ ▷ identify guess with maximum correlation
end function

In order to determine the entirety of the targeted value, one can repeat the analysis process, making guesses on a different part of the target until it has been completely covered. This analysis can be performed using the same power traces for each region being guessed.

When developing an attack using CPA, the majority of the effort goes into producing the power model. This involves two significant points in the algorithm. First is the target value; this is the end goal of CPA, obtained by checking guesses for small sections until the entire value is determined. Typically, the target is an encryption key. The other significant point is the modeling location, at which the algorithm's power consumption is expected to relate to some internal variable. This relationship can be detected through correlation, revealing the variable's value. The model's function is to compute the expected power for a given value at the modeling location, given some known input value and a guess for a part of the target value.

Generally, the model will be similar to the section of the original algorithm between these two points; however, because only a part of the target value is guessed at a time, the implementation must be able to produce a result even with significant missing information. This is the primary factor making a power model difficult to produce. To obtain useful data, one needs to select sets of bits in the modeled and target locations such that the modeled bits are computable based on the targeted bits, while also ensuring that the number of targeted bits remains small enough that performing the correlation analysis remains computationally feasible. The following paragraphs describe these selection requirements more rigorously.

Let G be a set of guessed bits; these are the bits in the target location for which every possible value is tested and one value is determined to be correct. These possible values are indexed by g, which ranges from 0 to $2^{|G|}-1$. For example, if $G = \{2, 5, 8\}$, then the index $g = 3_{10} = 011_2$ represents bit 2 having the value 1, bit 5 having the value 1, and bit 8 having the value 0. Other bits in the target location are undefined.

We also define the set V as the set of bits which are known and variable between traces; at index i, the bits specified in V have their value determined by the bits of the input I_i, with i ranging from 1 to $|I|$. Each input value in I also has a corresponding power trace $P_{*,i}$, produced by setting the bits specified in V to match I. Please note the distinction between a set of bits and the value represented by those bits.

Finally, we define the sets C and O. C is the set of bits which are known and constant across all operations, and O is the set of all bits in the modeling location. Note that not all bits in O will necessarily be calculated in the final power model.

Given the sets C, V, and G, there are several criteria for a bit $b \in O$ in the modeling location to be *modelable*:

1. b is calculable: Each dependency of b is included in any of C, V, or G
2. b is guess-dependent: b depends on at least one bit in G
3. b is variable across traces: b depends on at least one bit in V

Equations 2.4a through 2.4c state these criteria using set notation.

$$\alpha(b) = \text{deps}(b) \subseteq (C \cup V \cup G) \tag{2.4a}$$

$$\beta(b) = \text{deps}(b) \cap G \neq \{\emptyset\} \tag{2.4b}$$

$$\gamma(b) = \text{deps}(b) \cap V \neq \{\emptyset\} \tag{2.4c}$$

If (1) is not met, the value of b cannot be determined for use in the expected power computation. If (2) is not met, b will have the same value for each guess of the value of the bits in G, contributing no information about which guess is correct and potentially creating a situation where two guesses have identical correlations. Finally, if (3) is not met, b will have the same value across all traces for any given guess, making its contribution to the modeled power constant with respect to i. The correlation calculations are performed on vectors along i, and correlation is unaffected

by constant offsets; therefore, any b not satisfying condition (3) will have no effect on the result and should be excluded.

Using these criteria, for a given C, V, and G, we can define the set of all modelable bits $M \subseteq O$ as shown in Equation 2.5.

$$M = \{b \in O : \alpha(b) \wedge \beta(b) \wedge \gamma(b)\} \tag{2.5}$$

In a single attack formulation, the sets C and V remain constant. The difficult part of power modeling is the selection of the guess sets G. The goal is to maximize the number of modelable bits $|M|$ corresponding to each G, while at the same time keeping the size of each G small enough that performing correlation calculations for all $2^{|G|}$ possible values remains computationally feasible. Additionally, in order to have a complete attack, the guess sets G must eventually cover the entire targeted location when combined.

Once a G is chosen, the attack is performed as described earlier, testing all possible values of the bits in G to determine which is correct. This is then repeated with the remaining sets G until the entire target value is obtained.

For the application of CPA to MK-3, two different attacks are performed, each requiring a different power model and selection process for G. See Section 3.3 for the details of these attacks and their power models.

2.4 Leakage Assessment

Leakage assessment is a category of methods for identifying whether or not an implementation of a cryptographic operation is vulnerable to side-channel attacks without attempting to perform such attacks [2,24,25,26,27]. This usually takes the form of statistical testing on captured power traces. On its own, a single leakage assessment test has limited utility; the raw amount of leakage is highly implementation-dependent,

making comparison with published results difficult. Instead, leakage assessment is most useful for evaluating the effectiveness of implementation-level hardware countermeasures, where the algorithm and environment are kept consistent and the same test is run with several different countermeasures.

2.4.1 Test Vector Leakage Assessment

Test Vector Leakage Assessment (TVLA) is a leakage assessment method based on testing the influence of intermediate values on power consumption. This is done by collecting many power traces which are members of two sets; one set uses a fixed input value, while the other set uses a randomized input value. Then, Welch's t-test is used to test whether the distributions of power consumption at each point in time have the same mean; the amount of confidence that the distributions differ indicates the amount of leakage detected at that point in the execution [2, 24, 26, 27]. The measurements are then repeated several times, using different values for the fixed input, to eliminate any dependence on the particular value which was selected.

2.4.2 Holistic Assessment Criterion

An alternative to TVLA is the Holistic Assessment Criterion (HAC), introduced in [25]. This method has a similar basis to TVLA, attempting to quantify the measurement of whether different inputs produce identifiably different power usage; however, in contrast to TVLA, traces are not only considered on a sample-by-sample basis. Instead, HAC is based on comparing nearest neighbors, i.e., traces which have the least difference along their entire length. Using this information, the exchangeability of traces is evaluated, and the final metric is presented as a value representing this exchangeability with a confidence interval. By collecting and analyzing more traces, the confidence interval can be reduced, obtaining a more precise indication of the amount of leakage [25]. This is an advantage over TVLA, as it provides a way to

identify when enough data has been collected to have a meaningful result.

2.5 Power Analysis Countermeasures

Many countermeasures have been proposed for eliminating side-channel information leakage and mitigating the effectiveness of side-channel attacks. This section describes several common countermeasures at various levels of the design process.

2.5.1 Implementation-Level Countermeasures

Implementation-level countermeasures are modifications made to protect a specific hardware design against side-channel attacks. These often come in the form of hardware architecture changes, which can be costly to implement and may need to be tailored to the desired application. The following sections describe a selection of these implementation-level countermeasures.

2.5.1.1 Amplitude Noise

When faced with the problem of power side-channel leakage, a common intuition is to introduce noise into the measurements by adding a component which consumes random amounts of power, such as a set of controllable ring oscillators [28]. This technique is one of several related to introducing noise, collectively called hiding techniques, which aim to reduce the signal-to-noise ratio of any power measurements targeting secret information. In the case of simple power analysis, the introduction of random additional power consumption can successfully prevent an attack; however, with more sophisticated techniques such as DPA or CPA, random noise is generally ineffective [2, 3, 8, 16, 17, 18, 22]. These methods of analysis are already designed to extract the relatively-insignificant leaked information from noisy data; although introducing random noise can increase the number or quality of power traces required for a successful attack, it is typically incapable of complete prevention [2, 3, 17, 18].

An alternative to completely random power consumption is to equalize power consumption over time, using a configuration such as current sensor controlling a shunt transistor to consume unused power [3,8]. Theoretically, making the power consumption perfectly constant would remove any possibility of power attacks; however, such a device-wide component is effectively impossible to implement in practice. Other methods of reducing the power usage data dependency, such as balancing, decoupling, and masking, are discussed in later sections.

One design technique which combines the benefits of these two methods is dynamic voltage and frequency scaling (DVFS), which automatically modifies the operating voltage and clock speed of a device to optimize its power consumption for the current processing demand. Many systems already incorporate this technology for its energy savings and efficiency gain, and with minor modifications, it could be adapted to randomize power usage for the purpose of side-channel security [3, 16].

2.5.1.2 Temporal Noise

In addition to producing randomness in the amount of power being used, DVFS is also valuable for its temporal characteristics. In order to perform DPA or CPA, the power traces need to be aligned so that each sample corresponds to the same point in the operation [23]. By modifying the clock frequency during operation, DVFS makes this alignment much more difficult, as it is possible for the frequency to change in the middle of a trace. Although this type of change can be identified fairly easily, the frequency difference still introduces error when combining traces and requires an attacker to guess at what exact points the target operations occur [3].

Other methods of introducing temporal noise exist aside from DVFS. For example, a device can use a separate clock for critical operations, not accessible from the outside and out of phase with the main clock; this prevents an attacker from using the external clock for sample alignment, as well as offsetting the protected signal changes from the

main clock edges [3, 8, 17, 18]. Alternatively, random delays can be inserted during processing, causing trace alignment issues as with DVFS [3, 17].

One particularly useful method of adding temporal noise is to interleave random data with the true input data, discarding the irrelevant results after the fact. This has the advantage of creating a similar power signature to the operations being performed on real data, making it difficult to identify which power traces correspond to the fake data [3, 17, 29]. If they are not correctly identified by the attacker, these traces could potentially replace those corresponding with known inputs, interfering with the results of DPA and CPA.

2.5.1.3 Balancing

As mentioned earlier, a theoretical device which always consumes exactly the same amount of power is immune to power analysis attacks. Instead of a single chip-level power compensation device, several different balanced logic styles have been proposed which attempt to equalize power for individual operations at a gate or transistor level.

One approach to balancing power consumption is to design custom logic cells which use the same amount of energy every clock cycle. This can be done using the ideas of Dynamic and Differential Logic (DDL). Dynamic logic constructs gates by replacing the pullup network of a standard Complementary Metal-Oxide Semiconductor (CMOS) design with a precharge transistor or small precharge network, and activating the resulting circuit in two phases. First, the precharge phase activates the pullup network, charging the circuit's inherent capacitance; then, in the evaluate phase, the precharge network is switched off, and the pulldown network then discharges the circuit if the inputs allow it to do so. This requires the circuit inputs to be monotonically rising, because once the output has been discharged, it cannot be recharged until the next precharge phase. Differential logic operates using gates which use and produce the complement of each input and output, as well as the val-

ues themselves. Both of these techniques were originally developed as methods for improving the speed of digital logic, but they also have useful properties for reducing power side-channel leakage [8].

Dynamic and differential logic techniques can be combined to form Dual-Rail Precharged (DRP) logic, a type of logic design taking advantage of both techniques' side-channel prevention properties. The key advantage of DRP is that each gate is ensured to have exactly one logic transition per clock cycle [8,16,22,29,30]. A dynamic circuit guarantees that there will be a maximum of one transition per cycle; either the output will remain high during the evaluation phase, or it will be pulled low and be incapable of returning to a high state until the next cycle. When combined with differential logic, the possibility of no transition is eliminated. Both outputs become high during the precharge phase; during evaluation, if the direct (non-complementary) output remains high, then the complementary output must change to low, and vice versa. The first implementation of DRP is Sense Amplifier Based Logic (SABL), proposed in [30], which is designed to ensure that all internal nodes are eventually discharged, using a cross-coupled inverter as a sense amplifier to generate the output signals.

The main weakness of DRP is that only one of the outputs will make the transition from high to low and back. The goal of equal power consumption independent of the output value is only achieved if the capacitances of both outputs are perfectly balanced, which is a difficult task requiring specialized routing techniques and design tools [16]. In response to this issue, three-phase logic designs were developed such as Three-phase Dual-rail Precharge Logic (TDPL), which extends DRP by adding a discharge phase after evaluation. [3, 16, 31]. This third phase discharges all output nodes before the following precharge phase, ensuring that both outputs make the transition in every cycle. Because all outputs transition every time, the difference in per-cycle power consumption caused by unbalanced output capacitances is eliminated,

allowing the use of traditional routing techniques.

Balanced logic designs such as those described previously can be very effective at reducing power side-channel leakage; however, they all require the use of customized logic cells, which is a significant undertaking in ASIC designs and is impossible on FPGAs. To address this, several logic designs have been proposed which make use of standard cells, instead balancing power consumption by implementing DDL techniques at the gate level [3, 16, 22].

One such logic design is Simple Dynamic Differential Logic (SDDL). At this higher level of implementation, differential gates are constructed by duplicating standard gates and modifying the duplicates in accordance with De Morgan's laws; for example, a differential AND gate is constructed by combining a single-ended AND gate, connected to the non-inverted inputs, with a single-ended OR gate, connected to the inverted inputs [8, 32]. In this configuration, the combined differential AND gate is able to produce both direct (by the AND gate) and complementary (by the OR gate) results given direct and complementary inputs. Other types of differential gates are constructed by the same method.

SDDL augments these gates by adding a single-ended AND gate after both components of the differential unit, if the unit implements negative logic such as NAND or XOR operations; these AND gates allow both outputs to be forced to zero, implementing a "pre-discharge" phase where all nets are discharged before evaluation [3,8,22,32]. For consistency with DRP and dynamic logic terminology, the literature continues to refer to this as a precharge phase. The precharge phase in SDDL has the same purpose as in DRP; by discharging all nets to zero, it is guaranteed that at least one transition from low to high will occur during the evaluation phase. However, unlike DRP, the inclusion of negative logic introduces the possibility of glitches, which can cause more than one transition to occur in a single evaluation phase and thus reveal information via the power side-channel [3, 8, 22, 32, 33].

In order to correct this, [32] introduced Wave Dynamic Differential Logic (WDDL). WDDL operates on the same basic techniques as SDDL, but restricts the implementation to use only positive logic (i.e., AND and OR gates); rather than implementing negative gates, inversion is performed by swapping the differential inputs [3,8,32,33]. This restriction provides several benefits; most importantly, the removal of negative logic eliminates glitches, guaranteeing that exactly one low-high-low transition will occur per cycle [8,29,32,33]. In addition, WDDL significantly reduces the number of precharge gates which need to be inserted. Because only positive logic is permitted, setting the inputs of a combinational chain to zero will cause the first level of gates to output zeros to the next level; this means that precharge gates are only necessary at the beginning of each combinational section of the design. During the precharge phase, the zeros will ripple through the entire combinational chain in a wave, the phenomenon giving WDDL its name [8,32].

Unfortunately, there are also several aspects of WDDL which make implementation difficult. Although it eliminates some of the precharge gates, WDDL still requires more area than SDDL, and significantly more area than an unprotected implementation, especially on FPGAs [3,8,32,33]. This is primarily due to the translation of all logic into only AND and OR gates, a task which also requires modifications to ASIC and FPGA design tools [32,33]. Additionally, replacing inverters with swapping of differential pairs introduces significant complexity when trying to ensure that each side of the differential operations is routed with equal capacitance, which is necessary for maintaining data-independent power consumption.

One method for reducing this difficulty, also proposed in [32], is Divided Wave Dynamic Differential Logic (DWDDL). This modification takes advantage of the fact that in between inversions, there is no cross-connection between the direct and complementary logic chains; i.e., they are completely independent from one another [32]. This allows the routing effort to be reduced by first placing and routing a positive-

logic section of the original design, and then duplicating the routed section into a different location. By exchanging AND gates for OR gates and vice versa, as well as swapping the positive and negative inputs to the duplicate chain, a result identical to WDDL is achieved, but with identical routing between the two chains [32]. This method can be used on any section of the logic not including an inversion (a swap of the differential pair), reducing the workload of differential routing to only the interconnect between these larger sections [3, 8, 22, 32].

Overall, power-balancing countermeasures are a promising method for preventing power side-channel leakage; however, this security comes with costs. With DRP and TDPL, this cost primarily comes as the necessity of developing custom logic cells and integrating them with the rest of the design [16]. In situations where this is not possible, WDDL and DWDDL are good candidates, but have the downside of greatly increasing area requirements and reducing maximum clock speeds [8, 32]. In the case of FPGAs, it may be desirable to use SDDL instead, as its inclusion of negative logic simplifies implementation and allows for better utilization of FPGA features [33].

2.5.1.4 Decoupling

In ASIC designs, there is also the option of using power supply modifications to remove the association between power usage and data. One such method, presented in [34], is to create a set of capacitors within the chip that are used to power cryptographic circuitry in isolation from the main power supply. Before each operation, these capacitors are charged, then disconnected from the power supply; the protected circuit is then operated using only the energy stored in the capacitors. Finally, after each operation is completed, the capacitors are discharged before being recharged again, erasing any information on how much power was used over the entire operation. By decoupling the amount of power delivered into the chip from the actual power consumption of the protected operations, power side-channel leakage is pre-

vented [3, 22, 31]. Overall, this method is quite similar to TDPL, only implemented at a larger scale. As with TDPL, there is still the possibility of some information leakage if an attacker is able to identify the amount of unused energy discharged after the fact; however, as shown in [34], the technique is quite effective in practice.

2.5.1.5 Low-Level Routing

When considering EM information leakage, implementation specifics such as interconnect routing can be much more significant than with power side-channels. As determined in [5] by detailed modeling and simulation, it is the topmost few metal layers which contribute the vast majority of electromagnetic emissions, due to their larger size and proximity to the outside of the chip. In order to prevent the leakage of secret information through these emissions, the proposed technique is to isolate critical cryptographic elements from these higher layers as much as possible by special routing constraints. Although this presents some difficulty in routing complexity, as well as potential speed limitations due to higher resistance of the lower layers, the tests performed on the AES S-box in [5] showed a significant increase in the number of EM traces needed to obtain the key; compared to fewer than 5,000 traces in an unprotected implementation, low-level routing combined with EM noise injection was able to raise the requirement to over 1 million traces.

2.5.2 Algorithm-Level Countermeasures

In addition to implementation-level countermeasures such as those described in the previous section, side-channel leakage countermeasures can be created by modifying the design of the algorithm being implemented or executed.

2.5.2.1 Constant Control Flow

In the context of software, one of the most intuitive countermeasures to defend against side-channel attacks is to remove any dependence of the control flow on the data being processed [2,18,19]. For instance, in a timing attack on RSA done via SPA, the square-and-multiply algorithm could be modified to always perform a multiplication operation in addition to squaring, simply discarding the result of the multiplication if it is not needed based on the current key bit. This eliminates the obvious differences in the power trace, but necessarily comes with the performance reduction resulting from multiplying in every iteration [19]. In order to completely eliminate data-dependent execution, such as the conditional variable assignment described above, branches can also be replaced with arithmetic expressions for assignments, as described in [2].

To extend this concept, memory access patterns can also leak information through side-channels such as timing, from access latency, as well as shared-resource side-channels manipulating the cache, such as the Meltdown attack discussed in Section 2.2.4. Software could also be modified to prevent this leakage by removing data-dependent memory accesses; for instance, if indexing into an array based on part of a key (such as in an S-box), all possible indexes could be accessed, discarding those results which are not needed in the computation [2]. However, this again comes with a significant performance reduction. Additionally, as discussed in [35], successful application of this type of modified program requires careful consideration of compiler configuration, particularly with regard to optimization.

2.5.2.2 Masking

Masking is a very common method for reducing power side-channel leakage, focused on preventing DPA, and can be applied to both hardware and software. The general process of masking is to combine secret input values of potentially leaky operations with random values, called masks, then perform the operations on the masked values

and reconstruct the desired output value from the masked output. The effect of this process is that the leaky operations are not performed directly on secret data; because the inputs are combined with random data, any information leaked through side-channels will not reveal the secrets without additional knowledge of the masks used [2,3,8,16,17,19]. This requires attackers to use higher-order analysis techniques which target both the secret data and the masks at once, significantly increasing complexity and the amount of side-channel information needed.

Although masking can be applied in a variety of contexts, such as modular exponentiation as described in [2, 17, 19], the majority of use cases call for a Boolean implementation in which the masks are combined with secret data using an XOR operation.

For linear operations, such as bitwise permutation and XOR, conversion to a masked implementation simply consists of duplicating the operation for the mask(s) of the masked input(s); the masking is able to propagate through these operations without issue, resulting in a successful reconstruction [17, 36]. Nonlinear operations, such as AND or an S-box, require additional modifications to work with masked inputs and outputs [3, 8, 16, 17, 36].

A simple example of masking a nonlinear operation is to mask an n-bit AND gate, as described in [37]. Given two masked input values X' and Y', with their masks being r_x and r_y respectively, we can compute the masked result $Z' = XY \oplus r_z$ as shown in Equation 2.6.

$$Z' = X'Y' \oplus (r_xY' \oplus (r_yX' \oplus (r_xr_y \oplus r_z))) \tag{2.6}$$

This is logically equivalent to the simpler expression $Z' = ((X' \oplus r_x)(Y' \oplus r_y)) \oplus r_z$, demonstrating that the end result is correct; however, the key difference between this and the form shown in Equation 2.6 is that at no point in the computation of

Equation 2.6 do the secret values X, Y, or Z appear as intermediate values which could potentially be detected via first-order power analysis [37, 38].

Masking individual gates to form a protected system without higher-level modification is technically possible, but comes at a high price; as seen in the previous example, a single unprotected AND gate was expanded to eight separate gates with masking. Because nonlinear operations such as S-boxes are substantially more difficult to adapt, a common technique is to approach masking from an algorithmic standpoint, dividing nonlinear functions into several linear and nonlinear sections which are simple enough to mask separately. However, this division is not an inconsequential task on its own [3, 8, 16, 17].

In addition to the theoretical complexity of designing a masking scheme, there are several issues which make implementation difficult. These include a large increase in area and reduction in performance, as well as the need for fast generation of random data; each nonlinear operation requires a new set of masks, which can result in masked implementations consuming several times the original inputs' size worth of random bits. These new masks need to be made available during the computations, introducing complexity to the design; additionally, to ensure side-channel resistance, the source and transmission of the masks must themselves be adequately protected from analysis [2]. Another consideration is that when implementing masking in software, one must pay close attention to compiler optimizations, as well as the use of hardware registers in the resulting program, to ensure that the masking measures are not nullified; this topic is covered in detail by [35]. Finally, this type of masking does not consider glitches. If masked operations are performed only in the specified order of evaluation, secret values do not appear as unprotected intermediate values; however, glitches can violate this assumption, potentially revealing the secret values even if they are no longer present once all signals have settled [2, 3, 38].

In [38], Nikova et al. introduce Threshold Implementations (TI), which is a mask-

ing method designed to work despite glitches and only require random data at the beginning of computations. The basis of TI is that secret values are XORed with $n-1$ masks in sequence to produce n shares; in order to recover the original value, all n shares are XORed together, and it cannot be recovered using any set of $n-1$ or fewer shares. In this context, the simple masking scheme described earlier would be a 2-share system, in which the original value can be obtained using the masked value and the mask in combination, but not from one individually [38].

Linear operations in TI are implemented in the same way as before, by performing the operation on each share independently; because the operation is linear, the combination of its individual outputs for each share is equivalent to the output of the operation given the combination of the shares (i.e., the output of the operation given the original value) [38].

The key difference between TI and simple masking is the method of implementing nonlinear operations. In TI, a nonlinear operation is divided into a set of n functions with two properties. First, each function must be incomplete; i.e., each function is independent of at least one share of each input variable. Second, the set of functions must be correct, in that the XOR of all function outputs provides the expected result. As described in [38], this will require a minimum of $s+1$ shares for a function of s variables.

Because each function is independent of at least one share, any leakage from an individual function is not capable of revealing the secret value on its own, even in the presence of glitches [2, 3, 38]. This applies to the implementation of linear operations as well; because each operation acts on only one share, it is incapable of revealing the original value. Based on this property, the need to provide new random values during computations is eliminated; processing shares in separated functions, even for nonlinear operations, allows random values to be provided only at the beginning of computations [38]. However, given the need to produce several shares per variable,

the initial random data requirement is also increased.

There are also other concerns with TI. First, as with simple masking, nonlinear operations still present a problem with complexity; in particular, creating the set of incomplete functions for an operation is not a trivial problem [2, 38, 39, 40, 41, 42]. Second, the independence of functions implementing a nonlinear operation provides side-channel protection with glitches given that an attacker can only target a single function at a time [38]; although doing so increases the difficulty of the attack, this assumption does not always hold true. This can be partially remedied by requiring a third property of the set of functions; namely, that the distribution of the output remains uniform given uniformity of the inputs. However, this additional restriction exacerbates the problem of complexity, both in terms of developing a set of functions as well as implementing it, significantly; instead, another option is to restore uniformity by inserting a pipeline stage and re-masking some of the shares [38,39,40]. This technique, in exchange, brings back the requirement of additional random data being provided during computation. ASIC designs also have the option of using a logic style which does not allow glitches, eliminating the issue altogether. Finally, based on the application, TI can have a significant cost in terms of resources, especially if attempting to maintain uniformity directly; [40, 41, 42, 43] show the costs of implementing TI for various cryptography schemes.

2.5.3 System-Level Countermeasures

System-level countermeasures are implemented by modifying the environment or manner in which the target device is operated, rather than the encryption algorithm or the chip-level design of the device itself. Instead of attempting to completely eliminate side-channel information leakage, these countermeasures assume that some degree of leakage is inevitable, and instead seek to make the overall system robust despite this leakage.

2.5.3.1 Physical Security

Typically, power and EM side-channel attacks require close physical access to an operating device for an extended period of time. Power attacks in particular often require physical modifications to the device in order to obtain measurements of high enough quality to perform analysis; for example, a very common modification is the removal of decoupling capacitors to enhance the effects of immediate power consumption on the supply voltage. EM attacks are less invasive; depending on the specifics of the target device, an attack could take place from up to several meters away, or may require placement of a probe very close to the chip in question [4].

One method of preventing side-channel attacks is to design the system such that it is not possible to gain the required access while still having a functioning device. For power attacks, [16] describes several tamper-protection techniques; for example, erasing any encryption keys from memory if the device detects that it has been physically opened, which would effectively prohibit an attacker from gathering data after attempting to make any physical modifications. Many other such tamper-protection schemes exist, most of which are variations on the device rendering itself inoperable if it detects by some means that it has been modified or is in the hands of an attacker. EM attacks are more difficult to prevent with this type of protection; typically, the most one can do is to implement shielding in the physical design, such that long-range attacks are not feasible [4, 16, 18]. If shielding is effective enough, it could potentially force an attacker to open the device and attempt to remove it; this could then trigger tamper-protection as described earlier.

2.5.3.2 Key Rolling / Key Transformation

An alternative to completely preventing access, as described in the previous section, is to modify the system so that only a limited number of traces can be collected with the same key [17, 18, 44]. If this number is small enough, then an attacker will

not be able to gather enough traces to perform a successful attack before the key changes; once the key has been changed, any newly collected traces are not usable in combination with those collected previously [2,8,17,44]. This type of countermeasure can only be effective if the device is protected well enough that techniques using a very small number of traces, like SPA, are not possible; this level of resistance can be accomplished fairly easily by applying one of the other countermeasures described in previous sections, even one as simple as introducing random noise.

It is important to note that re-keying the operations targeted by side-channel analysis does not necessarily imply a new round of key distribution to the devices as a whole; all that matters is that there is no discernable relationship between the inputs used as keys for the target operation, so that the resulting sets of traces are rendered incompatible for analysis [2, 18, 44].

A very simple example of this, mentioned in [18], is to first distribute an initial master key, and then after every n messages, derive a new key by hashing the previous key. Because modifying the key changes the relationship between inputs and power consumption, the traces collected with each derived key are not usable in combination with any others. In this system, the clear next choice of target for an attacker is the hash function itself. However, the hash can also be protected from side-channel attacks using countermeasures from the previous sections, potentially more easily than the main cryptographic operation; additionally, the time required to collect data is longer, as the hash is only computed when each key is produced rather than for every invocation of the target operation.

Although this simple example has technical security against side-channel attacks, it has several unaddressed problems when it comes to implementation, such as the difficulty of synchronizing key changes between different devices and the method by which key changes are enforced. However, a variety of rekeying schemes have been developed and published which resolve these issues; for example, [44] proposes a

scheme based on a skip-list structure, and [2, 17] provide overviews of several others. In combination with implementation-level and algorithm-level countermeasures to provide basic protection, rekeying can effectively eliminate the security impact of any remaining side-channel information leakage.

Chapter 3

Methodology

3.1 MK-3 Implementations

All versions of the permutation function are used in an abbreviated sponge construction, as shown in Figure 3.1. Because IV absorption is the target area, all later steps can be removed with no effect other than a reduction in the time needed to collect power traces. Additionally, because both attacks aim to recover the state after absorbing the key, it makes no difference whether a 128-bit or 256-bit key is used; in either case, the state is 512 bits wide. For simplicity of the test architecture, this work uses a 128-bit key for all attacks, and uses the default customization values in all cases other than the theoretical capacity attack computations.

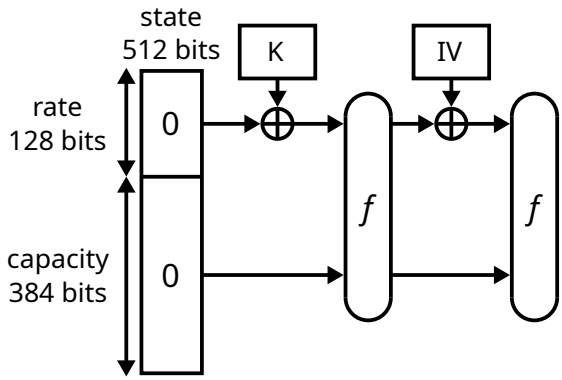

Figure 3.1: High-level sponge configuration used for power analysis.

In total, there are three different environments in which MK-3 was implemented: simulation, hardware, and software.

3.1.1 Simulation

In [45], Werner et al. implemented the MK-3 permutation function using the Very High Speed Integrated Circuit (VHSIC) Hardware Description Language (VHDL), which can be synthesized to produce a hardware design for either ASIC or FPGA applications. The implementation described in [7] was tested on an FPGA, but uses an ASIC-optimized S-box design developed by Wood in [13]. Based on the MK-3 implementation in [45], Stafford developed an alternative FPGA-optimized implementation of the S-box, which is the focus of [46]. The resulting combined design, with the S-box from [46] and the remaining implementation from [45], is the design used by Stafford in [8] for simulation-based power analysis.

In the simulation environment, the main contributions of this work are a reproduction of Stafford's results and updates to the implementation allowing the full number of rounds to be used for analysis and providing access to the state register through a wrapper. Reproduction of the simulations required several process updates due to differences from the infrastructure used to perform the original work, but was eventually successful. The implementation update was fairly simple, only consisting of minor changes to the interface and the timing of the round counter.

Although the simulation approach has the benefit of not requiring any specialized hardware, this comes at the cost of slow data collection and large file sizes, significantly limiting the number of traces which can be acquired. Additionally, the power consumption data produced by simulation requires preprocessing before it can be used in CPA. Section 3.2.1 discusses the data collection process for simulation in more detail.

3.1.2 ChipWhisperer

This work mainly focuses on power attacks using data collected from hardware. To perform these attacks, a system is required which both implements the MK-3 algo-

rithm and takes detailed measurements of its power consumption.

There are several options for creating this system. One option, described in [47], uses two FPGA development boards as a target and a controller, in addition to a USB oscilloscope and desktop computer for power measurement and processing. The main advantage of this system is that it uses generic and readily available products, providing a high degree of customization at a relatively low cost. However, it also requires a significant amount of preparation, such as programming the interaction between the oscilloscope and control board and removing filter capacitors from the target to obtain cleaner power traces. This is in addition to implementing the encryption algorithm and analysis software, which are steps required for attacking any new design.

The data collection system used in this work is called the ChipWhisperer [48]. In contrast with the systems described earlier, this product provides a ready-made framework for power analysis, including a specialized oscilloscope and target board along with software for configuring the system and taking measurements. In the ChipWhisperer system, the main tasks are integrating the desired algorithm into the target's programming and implementing the power analysis process.

For testing MK-3, the same base implementation was used as for simulation [8]. This was integrated into the target FPGA's existing hardware design by adding a wrapper which translates the control signals and manages the high-level absorbing and squeezing process. Additionally, the implementation was modified by adding a trigger signal, indicating when the computations have reached the desired point for the oscilloscope to capture a trace. The specifics of the MK-3 hardware designs used for testing are discussed in Section 3.1.4.

3.1.3 C++

The third and final implementation of MK-3 was a software implementation in C++, primarily used to determine the expected results of power analysis so that its success can be evaluated. The three implementations, in simulation, hardware, and software, were also verified against one another to ensure that all were functioning correctly and consistently with one another.

The C++ version of MK-3 was initially created in [13], which implemented only the S-box. This was then extended to a full implementation of the algorithm in [1]. The full version was used in this work with only minor modifications. Specifically, the software was edited to expose the internal state values targeted by the CPA attack, as well as to accept command-line inputs rather than hard-coded values for use in the analysis scripts.

3.1.4 Variants

This section provides details on the various modifications made to the base MK-3 VHDL implementation and where each modified version was used.

First is the base implementation itself, as created in [8]. Figure 3.2 shows a Register-Transfer Level (RTL) diagram for this design, with all repeated components combined into single elements.

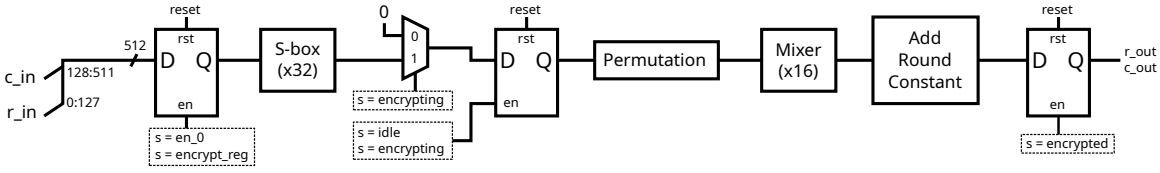

Figure 3.2: Original single-round MK-3 implementation with register after S-box.

This diagram is fairly similar to Figure 2.4, the block diagram from Section 2.1.2 which shows the basic structure of the permutation function f. However, there are a few differences to note. First, because this is an RTL diagram, the round state register is shown in a more detailed style, and control signals and signal widths are included

where appropriate. Second, there are several additional and missing components. One new component is the register at the end of the function, used for keeping the output stable while it is being read by the wrapper logic in the target. This output register is only activated at the end of computations.

The other additional components are the multiplexer and register in between the substitution and permutation stages. These were added to make power analysis attacks easier to perform, as well as to help demonstrate the effect of design changes on such attacks. The new register buffers the S-box outputs, synchronizing the inputs to the next stages, while the multiplexer clears the register, improving the accuracy of a Hamming weight power model. If the register is not reset, then the attack must depend only on the difference in power consumption between setting a 0 or a 1 rather than the difference between a bit changing or remaining the same.

Finally, because this is only a single-round implementation, the return path from the round constant output to the state register input is not included. This connection does exist in the VHDL code, but is never activated because the round counter code is disabled and so has been omitted from the diagram.

In these RTL diagrams, the expressions in dashed boxes indicate that the connected control signal is active when the state machine is in any of the listed states. Figure 3.3 shows the state diagram for this single-round implementation.

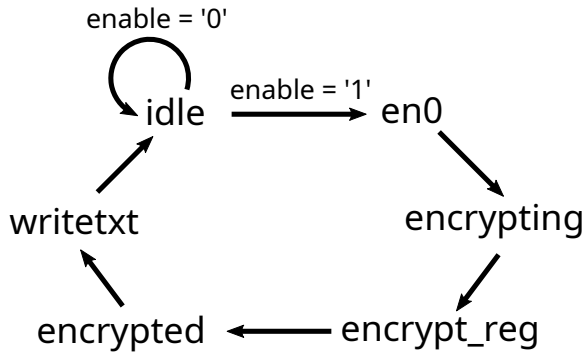

Figure 3.3: State diagram for single-round MK-3 implementation.

As seen in this diagram, the system begins in its idle state. When the enable

signal becomes active, it progresses through the other five states unconditionally until reaching the idle state again, with each state controlling different components as depicted in the RTL diagrams. The state machine component itself is omitted from the single-round diagrams because for these cases, it has no inputs other than the top-level reset, clock, and enable signals.

Figure 3.4 shows the next variant of MK-3. This variant is based on the original design, modified by reenabling and adjusting the round counter to perform ten rounds of encryption instead of one. The return path to the round function input is also added back to the diagram.

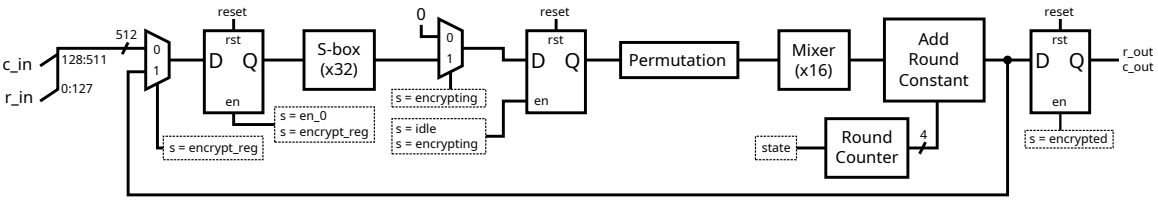

Figure 3.4: 10-round MK-3 implementation with register after S-box.

Enabling the round counter is particularly important to having a realistic implementation because if only one round is performed, then some bits of the state will remain the same for any key, potentially introducing a bias to the analysis. This happens because, within a single round, there is not enough diffusion present to make every output bit depend on the rate. Section 3.3.2 discusses this concept in detail, as it is also an important factor for the capacity attack.

As expected, increasing the number of rounds also increases the amount of time required to collect data. This is not a problem for the hardware implementation because the FPGA is able to run the algorithm in real-time at a high speed. For simulation, however, the increase is a significant issue. As mentioned in Section 3.1.1, simulations for power analysis need to record all signal transitions so that their energy usage can be computed. This results in slow simulation speeds and large file sizes even with only one round in use, so extending to 10 rounds makes it very difficult to collect the required number of traces in any reasonable amount of time.

Figure 3.5 shows the updated state diagram for this variant.

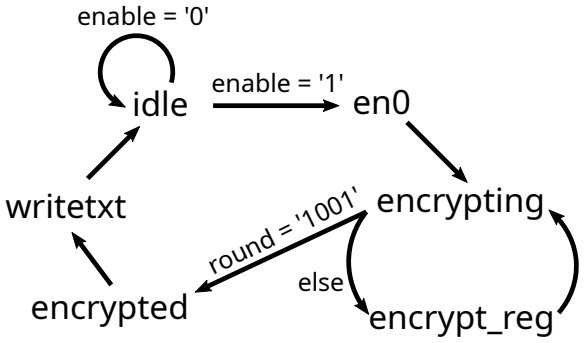

Figure 3.5: State diagram for registered 10-round MK-3 implementations.

As seen in the diagram, the states no longer transition unconditionally all the way back to idle. Instead, based on the current round number, a decision is made either to continue encrypting for another two cycles or to leave the loop and save the state in the output register.

Figure 3.6 shows a further modification to this design which resets the state register before the start of each new round. This variant uses the same state machine, shown in Figure 3.5, as before.

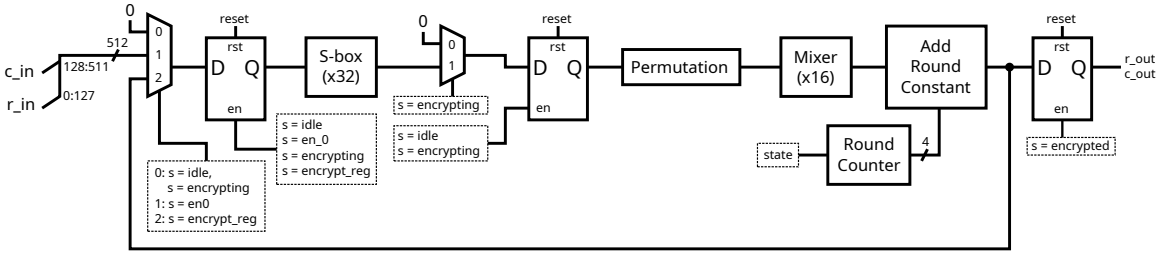

Figure 3.6: 10-round MK-3 implementation with register after S-box, and with state register cleared each round.

As with the S-box output register, this reset makes a Hamming weight power model more effective, as it can rely on the more significant difference in energy consumption between changing and not changing a bit, rather than only the difference between setting a bit to 0 and setting it to 1. This improves the effectiveness of the capacity attack.

Finally, Figure 3.7 shows an MK-3 variant which does not include any S-box output register and does not reset the state register.

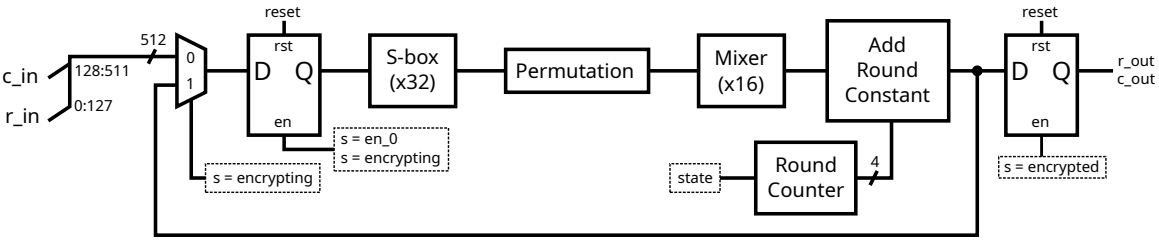

Figure 3.7: 10-round MK-3 implementation with no register after S-box.

This variant is meant to make testing more general by providing a realistic design with no CPA-specific modifications. The S-box output register is not required for implementing the algorithm, and was originally inserted in order to make power attacks easier to perform. However, adding this register is also not an unreasonable step for a designer wishing to increase the design's clock speed through pipelining. Because the S-box output register could potentially be included in a real design, most testing is performed on both this variant and the variant in Figure 3.4, using the other variants only where they are needed for simulation or the proof-of-concept capacity attack.

Because this variant omits the S-box output register, the state machine needs to be modified to match, as shown in Figure 3.8.

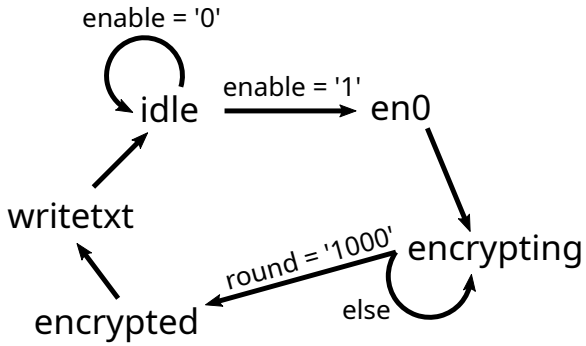

Figure 3.8: State diagram for unregistered 10-round MK-3 implementation.

As seen in this diagram, the only differences between this and the previous state machine are that each round requires only one clock cycle instead of two, and that the round threshold has been modified to account for this timing difference.

3.2 Data Collection

Using the implementations described in Section 3.1, power measurements were taken as required for CPA. This section describes the processes used to collect these measurements from simulations and from hardware.

3.2.1 Simulation

There are several steps required to obtain useful power data from simulation. The first step is to synthesize the VHDL design to a netlist, which implements the design in the form of logic gates. Next, the netlist is simulated, using a testbench file to control the stimulus and configuring the simulator to record the transitions of all signals in the design to a file. Once the simulation has completed, the recorded state transitions are analyzed, resulting in discrete values for power consumption at times determined by the defined delays of the logic gates. These steps were performed using the Synopsys Design Compiler, VCS, and PrimeTime tools respectively.

The resulting power measurements form a single, long trace spanning the entire length of the simulation. A Matlab script developed in [8] is then used to divide and align the power traces for each individual operation. Additionally, the precise point measurements produced by the simulation process cannot be used directly for CPA. There are far too many samples to process individually, and the transitions recorded in each sample are unlikely to correspond precisely with those from other related operations. The Matlab script resolves this problem by consolidating the power traces on a 1-nanosecond interval, where the value used for a given interval is the maximum of the points it contains. After this preprocessing is completed, the power traces are passed to the analysis script, described in Section 3.3.

3.2.2 ChipWhisperer

Section 3.1.2 describes the integration of MK-3 into the target, while this section provides details for the power measurement configuration and procedures.

In order to capture power data using the ChipWhisperer, the design first needs to be synthesized and a bitstream needs to be produced for the FPGA on the target board. This is done using the Xilinx Vivado tools. The resulting bitstream is then programmed into the FPGA through an interface on the ChipWhisperer oscilloscope, controlled by a Python script on a computer.

Another script is used to start testing and perform the measurements. This script first configures the test parameters, selecting a 5 MHz clock for the target and a sample rate of 90 MSamples/sec for the oscilloscope. In order to fit the longer runtime of a 10-round implementation with an S-box output register, 72 clock cycles are captured, providing 1296 samples per trace. Almost all of these samples will go unused in CPA, as the attacks only focus on the first round of IV absorption, but they are still useful to capture for context when viewing the traces.

After configuring the clock and capture parameters, the stimulus is arranged. The key is always a fixed value, only changed manually for entirely different test runs. The IV is completely random, generated new for each trace collected. By modifying this generation, multiple traces can be collected using identical inputs and averaged to obtain more precise measurements. This is an alternative to obtaining higher-resolution data and helps to eliminate noise. For the rate attack, 8x averaging is used, which reduces the number of different IVs required. This reduction speeds up the analysis process later on. The capacity attack, on the other hand, benefits more from additional IVs than from higher-quality traces, and its analysis speed is not as sensitive to the total number of traces. Therefore, no averaging is used when capturing traces for the capacity. Note that in theory, the same sets of traces can be used for both attacks; averaging simply helps to reduce the computational workload.

When collecting hardware power traces using the ChipWhisperer [48], the results are already in the expected arrangement and do not require preprocessing. Figures 3.9 and 3.10 show plots of 500 power traces produced by the 10-round registered design without a state reset depicted in Figure 3.4, including the full capture length and only the area used by CPA respectively.

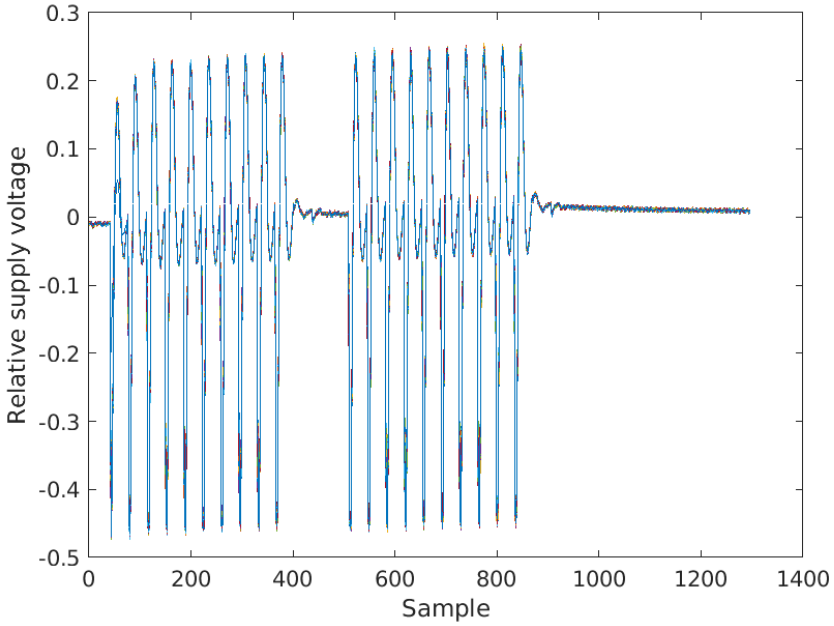

Figure 3.9: Full plot of 500 power traces from 10-round registered design.

In these figures, the X-axis is the sample number, representing time. The time between samples is approximately 11 ns, and there are 18 samples per clock period. The Y-axis is the relative power supply voltage after the shunt resistor; this measurement is AC-coupled, so constant offsets and low-frequency components are removed. There is no calibrated scale for the Y axis. Because these examples are from the registered variant without reset, there is a downward voltage spike caused by the power consumption of the S-box every two clock cycles.

After power traces have been collected, they are passed to the analysis scripts discussed in Section 3.3 to perform the developed attacks.

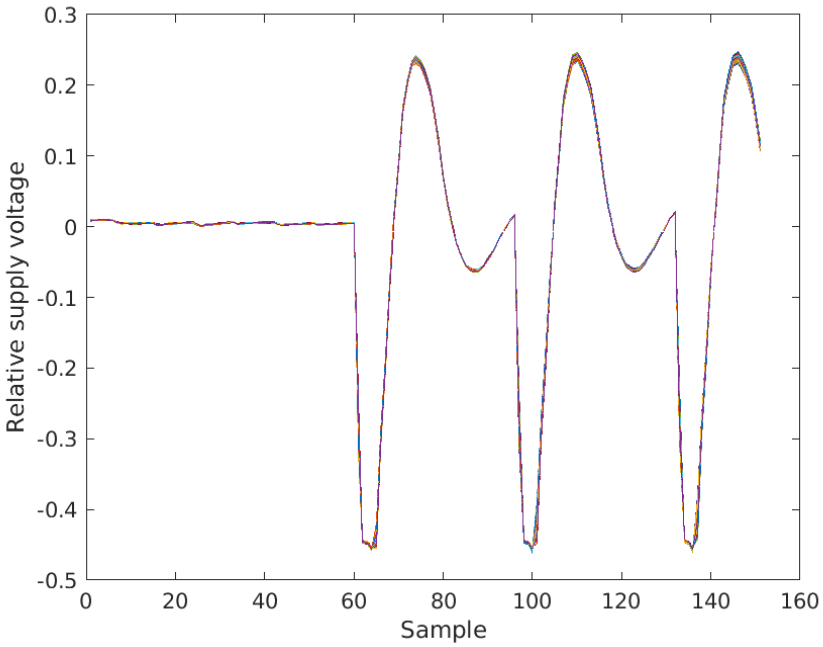

Figure 3.10: Plot of 500 power traces from 10-round registered design, CPA section only.

3.2.2.1 TVLA

As explained in Section 2.4, leakage assessment tests are primarily useful for evaluating implementation-level hardware countermeasures. In this work, no specific hardware countermeasures were implemented. Nevertheless, in order to aid future work, a small system was created for performing TVLA. This includes another Python script, which captures power traces while randomly using either a fixed IV or a randomized IV, as well as a Matlab script, which runs Welch's t-test at each sample of the captured traces as described in Section 2.4.1. This work does not apply this system.

3.3 Analysis

Using the collected power information, CPA attacks were performed, attempting to recover the 512-bit state after the absorption of the key. Due to the structure of the permutation function f, two different attack formulations were required. Both assume that the IVs supplied to the algorithm are known.

3.3.1 Rate Attack

The first attack targets the rate portion of the state, modeling the corresponding bits at the end of the substitution stage. As described in Section 2.3.3, any modeled bit must depend on both the target value being guessed and the input value changing for each trace. The rate is XORed with the IV before the start of the permutation function, so at any point in the first round before the mixing stage, these first eight 16-bit words are the only bits fitting the criteria. The other 24 words, in the capacity, are not affected by the IV and therefore cannot be modeled before the mixing stage. Additionally, this attack does not attempt to directly model the state outside of the permutation function. These values appear in parts of the overall FPGA design other than the core algorithm, such as the wrapper and the ChipWhisperer components, which will be different or missing in other applications of MK-3. Instead, this work focuses only on the implementation of the permutation function itself, avoiding any values whose results would be strongly influenced by the test infrastructure.

For this attack, the power modeling and guess set selection are straightforward. Each of the 16 output bits of a given S-box depend only on the 16 inputs to that S-box, and for the rate S-boxes, each input is a combination of a key bit and an IV bit. Therefore, we can simply perform guesses word-by-word, modeling the outputs of a single S-box at a time. Using the notation defined in Section 2.3.3, this means that each G is one word of the rate input to the substitution stage, and its corresponding M is the matching word of the substitution stage output. For this attack, V, the variable set, is the 128 input bits for the IV; C, the constant set, is empty; and O, the modeling location, is the full substitution stage output, a superset of M. Algorithm 3 describes the power model for this attack in pseudocode, matching the function signature defined in Algorithm 2.

To perform this attack, another Matlab script was used, originally created for the analysis of simulation traces in [8]. This script implements the CPA process in Sec-

Algorithm 3 Rate Power Model

```
1: w ← the index of the S-box being guessed [0, 7]
2: J ← a known IV                                   ▷ value of the bits in V
3: g ← the guess index for this computation
4: function ModelPower(J, g)
5:     IVIn ← J_{16w:16(w+1)−1}
6:     GuessIn ← the 16-bit binary value of g
7:     SboxIn ← IVIn ⊕ GuessIn
8:     SboxOut ← Sbox(SboxIn)
9:     return HammingWeight(SboxOut)
10: end function
```

tion 2.3.3, using the power model which was just described. First, the original results from [8] were reproduced using power measurements from the updated simulation infrastructure. Then, the script was modified to support power traces obtained with the ChipWhisperer. These measurements do not have the same degree of precision as the simulation measurements, as described in Section 3.2, so more traces are required to perform a successful attack on the same MK-3 implementation. However, when using the script on larger numbers of traces, the runtime started to become a problem, with results taking several hours or more to produce. To alleviate this issue, the analysis script was refactored to take advantage of Matlab's matrix processing performance, resulting in a speedup of approximately 34x compared to the original.

Another script was also created based on the first in order to determine how many traces are required to successfully attack a given implementation. This runs the same analysis process repeatedly, placing an artificial limit on the number of traces and iteratively changing this limit to test a range of trace counts. The traces are randomly selected from a larger pool of captured traces. For the rate attack, there are only eight potential words to find, so the success information is quite coarse. Therefore, to better evaluate the attack's performance, the analysis is repeated ten times at each trace count, using a different random selection each time. With this additional data, the relationship between trace count and success rate is much more clear. Section 4.1

describes the results of the rate attack, including these success rate metrics.

3.3.2 Capacity Attack

As described earlier, only the rate can be modeled based on the output of the substitution stage, because it is the only region depending on the IV at this point. In order to attack the capacity, the model needs to be extended into the permutation and mixing stages, where additional dependencies are introduced between the IV and the input capacity. Unfortunately, these dependencies are not as simple as those in the substitution stage.

3.3.2.1 Attack Formulation

The permutation stage ensures that for every mixer, each of the 32 inputs comes from a different S-box output [1], so each mixer always has 8 IV-dependent bits to work with. However, depending on the mixer polynomial used, a given capacity bit may or may not be combined with any IV-dependent bits before reaching the end of the mixing stage. The S-boxes also significantly limit the scope of a model involving the mixers. Because each mixer input comes from a separate S-box, any mixer output which does meet the criteria for modelability will need to depend on an output from both a rate S-box and a capacity S-box. Every S-box output depends on all 16 inputs of that S-box, and each mixer output depends on 2, 3, or 5 mixer inputs. Because at least one dependency must come from the rate and one must come from the capacity, even a single modelable mixer output bit will have 16, 32, or 64 dependencies at the beginning of the round which would need to be guessed. The correlation with only one bit is not very strong, so to get any satisfactory results, more than one mixer output would need to be attacked, increasing the required number of guess bits to 32 or more. This means that the model would need to be applied 2^{32} times per IV, in addition to computing 2^{32} correlations for every sample. These are very significant

computational requirements, and are not feasible in the scope of this work.

Instead, the target point is moved to the output of the substitution stage, so that S-box outputs are the bits being guessed. Although this does not directly provide any bits of the capacity, it does reduce the potential search space of a brute-force attack. Additionally, a Hamming weight power model is most effective when used on a register. Therefore, the modeling point is set at the state register of the second round, rather than directly at the mixer outputs. The round constant stage in between these points does not interfere with the model or dependency calculations, as it only applies an XOR with a constant. This new attack, targeting the capacity S-box outputs and modeling the next round state register, is what this work refers to as the capacity attack. Section 3.3.2.2 focuses on its implementation.

3.3.2.2 Attack Implementation

First, in order to model the power consumption of these bits, we need to be able to compute their values given a guess and an IV. This involves traversing the permutation and mixing stages. The permutation stage is fairly simple, as it only remaps the bit positions of the S-box outputs and can be represented with a simple transformation. The mixing stage, on the other hand, is more complex. Each mixer output bit can depend on between 2 and 5 mixer input bits, and the particular bits depended on are determined by the mixer polynomial as described in Section 2.1.2.3. Fortunately, these mixer dependencies can also be represented fairly easily. A given mixer output is simply computed as the XOR of these several input bits, and the round constant stage adds one additional XOR with a constant value.

To use these dependencies in the CPA script, a Python program was implemented which, given a choice of mixer polynomial, produces a list of sets of bit positions. Each element in the list contains the S-box output bit positions depended on by one round output, obtained by composing the dependency relationships of individual

mixer components, along with mapping back through the permutation stage. The base mixer dependencies are computed based on the definitions in Section 2.1.2.3, parameterized by the selected mixer polynomial. In the set-based description of CPA, this dependency list defines the deps() function, which takes a mixer/round output bit position and returns the set of 2 to 5 S-box output bit positions it depends on.

Now, using these dependency relationships, the CPA script is able to compute power model values based on guessed and/or known S-box output values. However, there still remains the problem of selecting which bits to guess and which output bits are truly modelable based on the selected guess set. This was the primary reason for introducing the detailed set notation for CPA in Section 2.3.3. Using this notation, the sets are defined as follows:

- guess bits, G: some selected set of capacity S-box outputs
- modeling location, O: all next-round state register inputs
- modelable bits, $M \subseteq O$: the truly modelable bits, given G
- known variable bits, V: the bits of the IV input
- known constant bits, C: the rate output from key absorption

Note the important assumption here that the rate output from absorbing the key is known. This value is not exposed directly, and the attack does not work without it. However, the assumption is still reasonable, as this value is exactly the target of the rate attack, which is in turn based only on true known values. It is still important to note that the capacity attack depends on the success of the rate attack, or of some other method of obtaining the rate.

Before continuing on to the process of selecting G and M, pseudocode for the capacity attack power model is shown in Algorithm 4, again matching the function signature defined in Algorithm 2.

Algorithm 4 Capacity Power Model

```
1: G ← the set of bits being guessed
2: M ← the set of bits being modeled
3: R ← the 128-bit key rate output                    ▷ value of the bits in C
4: RC ← the 512-bit round constant for round 1
5: J ← a known IV                                     ▷ value of the bits in V
6: g ← the guess index for this computation
7: function ModelPower(J, g)
8:     Rate ← Sbox(R ⊕ J)
9:     GuessIn ← the |G|-bit binary value of g
10:    Capacity ← 384 undefined bits                  ▷ sparse vector
11:    for i ← 0, |G| − 1 do
12:        Capacity_{G_i} ← GuessIn_i                 ▷ distribute the guess bits
13:    end for
14:    State ← Capacity ‖ Rate
15:    HW ← 0
16:    for m ∈ M do
17:        x ← 0                                      ▷ compute parity of mixer output m
18:        for d ∈ deps(m) do
19:            x ← x ⊕ State_d                        ▷ State_d must be defined
20:        end for
21:        HW ← HW + (x ⊕ RC_m)
22:    end for
23:    return HW
24: end function
```

A key assumption to note in this algorithm is that State_d must be a defined value. In order for this assumption to hold, G and M must be chosen so that, given G, every bit in M is modelable by the three criteria defined in Section 2.3.3. This is not as straightforward as it is in the rate attack.

We can start by creating a maximum set, identifying all bits which could potentially be an element of any G. First, let M' be the set of all modelable bits in the case that G encompasses the entire S-box output capacity. Then, construct the maximum set G' by combining all dependencies of the bits in M', excluding bits which are in C and V and therefore cannot be guessed. This set G' represents all S-box output capacity bits which could possibly be recovered through the capacity attack.

The content of G' depends on the particular irreducible polynomials chosen for each mixer. Note that although every mixer is constructed identically, the permutation stage rearranges their inputs, giving each one a different pattern of known and unknown inputs. This means that each mixer may potentially respond in a different way to a given irreducible polynomial, so the 16 mixer locations must each be treated separately. However, each mixer also makes its own individual contributions to G', not overlapping with any other mixer's contributions; in other words, the configuration of one mixer does not affect the bits contributed by any other mixers.

Based on this information, a Python program was created to calculate the size of G' with various combinations of mixer polynomials. This program provides 16 sets of information, one per mixer location, with each set containing the number of recoverable bits that the mixer contributes to G' for each of the 4080 possible irreducible polynomials. This can then be used to evaluate a given customization by simply looking up the number of bits contributed by each mixer with its configured polynomial and taking the sum of the results. Section 4.2 goes over the results of these upper-bound computations.

Ideally, we would perform the capacity attack using $G = G'$, but this is compu-

tationally infeasible due to the number of guess possibilities. In order to reduce the scope of each application of CPA, we can divide G' into several subsets S, subject to the following constraints:

1. The union of all S is of maximum size (ideally equal to G').
2. For each subset S, $|S| \leq 16$.
3. For each subset S, these exists no subset $T \subseteq S$ for which every $m \in \text{modelables}(T)$ has odd $|\,\text{deps}(m) \cap T|$.

where modelables(T) is the set of all modelable bits where the set of guess bits, called G in the criteria in Section 2.3.3, is equal to T.

Condition 1 must be true in order to make guesses on, and obtain a result for, the greatest number of bits possible. Condition 2 is flexible, but some limit is needed for computation reasons. Condition 3 is explained in the following paragraph.

As described earlier, due to the structure of the mixing stage, each mixer output bit can depend on either two, three, or five mixer input bits. For a set of guess bits S, if there exists some subset $T \subseteq S$ for which every mixer output bit $m \in \text{modelables}(T)$ depends on an odd number of bits in T, then every guess of S will have a partner guess where the values of all bits in T have been inverted, which produces a correlation of exactly the same magnitude. This is because inverting all bits in T will invert every output bit, producing a Hamming weight which is linearly related to the original result across all IVs.

If two guesses will always have identical correlations, then we are not able to determine which is the correct value. In order to obtain a unique result, it is necessary to choose S so that no such set T exists, forming another constraint in addition to those specified above. The optimization problem of finding several S following these constraints, which in combination cover all of G', is not solved in this work. It is

possible that there are no S for which no T exist, limiting this type of attack to providing multiple possibilities for the correct value rather than a definitive result.

In order to provide a proof-of-concept attack without having the solution to this optimization problem, the following heuristic approach was used to select a G of reasonable size (≤ 16 bits):

1. Sort G' by total number of dependent end-of-round bits, in decreasing order. This is not a deterministic process, as many bits have the same number of dependents.

2. Take the top k bits as G'_k, and let M (necessarily $\subseteq M'$) be all bits which are modelable by the criteria in Section 2.3.3 given that $G = G'_k$.

3. Construct G by removing from G'_k any bits which are not depended on by any bit in M.

4. Adjust k to maximize $|M|$ in the previous steps, subject to the condition that $|G| \leq 16$ for computation reasons.

With all mixers using the default polynomial `0x002d`, the largest k satisfying the conditions was 28, resulting in $|G| = 15$ and $|M| = 27$. This G and M were used to perform a proof-of-concept capacity attack, the results of which are detailed in Section 4.2.

Chapter 4

Results

4.1 Rate Attack

After implementing the design variants, capture system, and analysis software, the rate attack was applied to the registered and unregistered 10-round MK-3 variants depicted in Figures 3.4 and 3.7. As an example, the registered implementation was tested with the single-shot analysis script, using 5,000 traces captured with the ChipWhisperer. Figure 4.1 shows the correlation plots produced by this analysis.

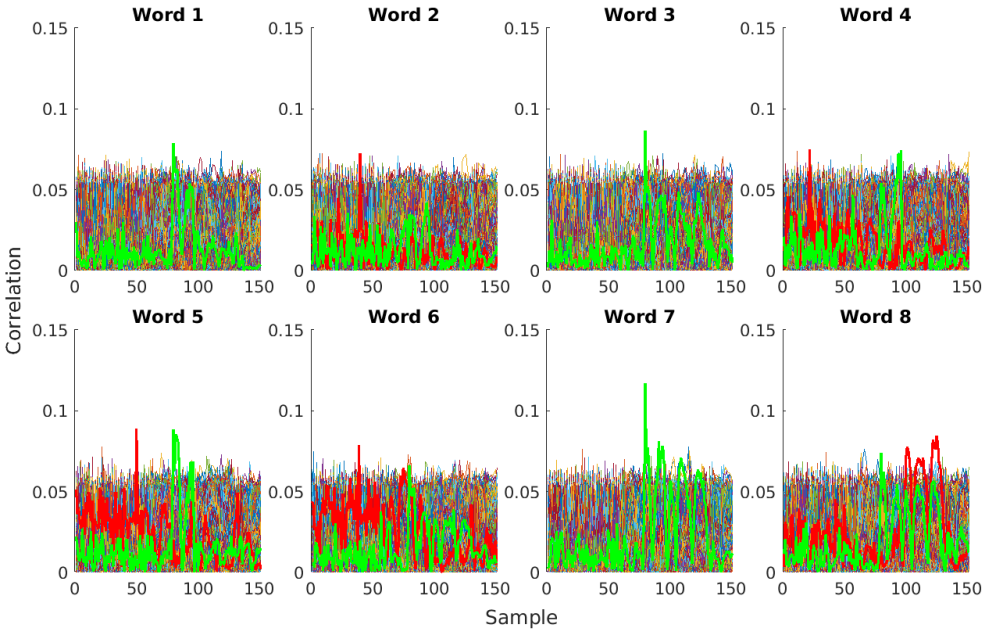

Figure 4.1: Analysis results for 10-round registered variant using 5,000 traces.

In this figure, each of the eight subplots corresponds to one word of the rate

being targeted. On the X-axis is the sample number, representing time. The Y-axis is correlation; specifically, each line on a plot depicts the correlation results over time for a particular guess of the plot's word in the rate. Each word is 16 bits, so there are $2^{16} = 65536$ possible guesses, and therefore 65536 lines on each plot. The correlation for each of these guesses indicates the strength of the relationship between its modeled power consumption and the true power consumption. In most cases, if there are enough IVs used (i.e., enough traces collected), then one of the guesses will have a strong enough relationship for the correlation to rise above the baseline, producing a peak at the point in the algorithm's execution where the power matches. The guess with the highest peak for a word is considered to be the correct value of that word. In the correlation plots, the truly correct value, determined from the real key for evaluation purposes, is highlighted in green. If the highest peak belongs to some guess other than the correct guess, then the plot of the other guess is highlighted in red, and the analysis incorrectly identifies it as the value for that word.

Based on this figure, we can see that words 1, 3, and 7 were correctly identified by CPA, while the other five words were not. Additionally, note the location of the peaks for the correct results. These plots are over the same sample range depicted in Figure 3.10, and as expected, the points of maximum correlation occur at approximately sample 80. Based on the plotted power traces, this point is where the S-box output register is activated, which is indeed the location targeted by the power model.

Next, Figure 4.2 shows another test of the same design, this time using 20,000 IVs/traces rather than 5,000.

As seen in this figure, all eight words are correctly recovered by this version of the attack. Notice that in most cases, the correlation of the correct guess has not increased significantly. Instead, the baseline correlation has decreased, exposing the peaks of the correct guesses which were previously indistinguishable from the others. With more IVs, there are more points for the correlation metric to use, providing

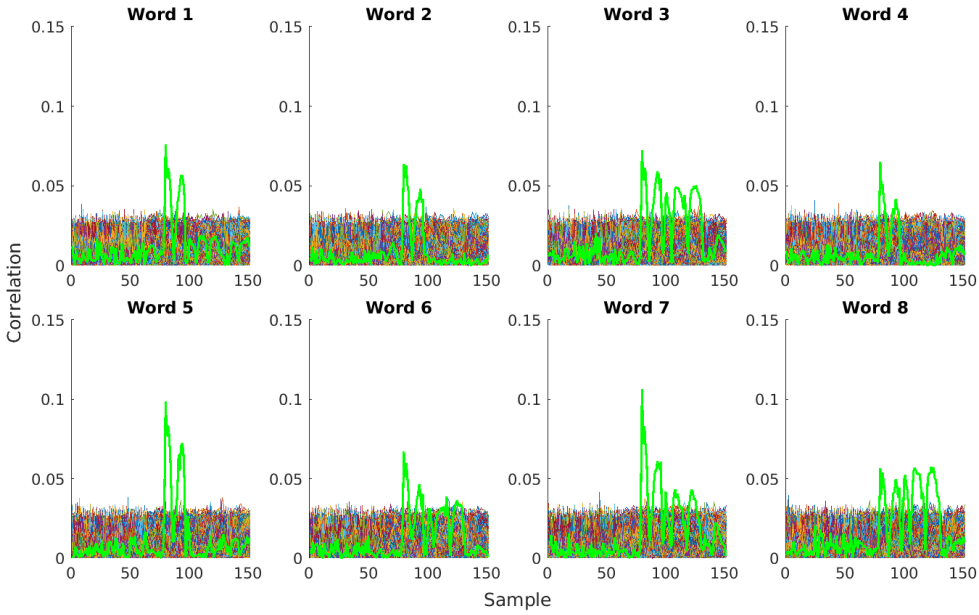

Figure 4.2: Analysis results for 10-round registered variant using 20,000 traces.

additional possibilities for the incorrect guesses to not match the true power and thus reducing their correlation.

In order to characterize the success of CPA against the registered implementation with respect to the number of traces captured, the repeating analysis script was used. For this analysis, two different keys were tested through the same process. Figure 4.3 shows a plot summarizing the results of this testing.

In this plot, solid lines indicate the success rate when considering only the result identified as most likely, while dashed lines indicate the success rate where the correct value being in any of the top ten most likely candidates is considered as success. On the Y-axis, the scale only goes up to 25%. This is because the rate is only 1/4 of the total state bits, and this attack does not target the capacity section of the state. As expected, increasing the number of traces increases the success rate, with the top-ten success rate consistently remaining slightly higher than the perfect success rate. The results between the two keys show no significant differences across the entire test. At approximately 15,000 traces, both methods achieved the maximum success rate for

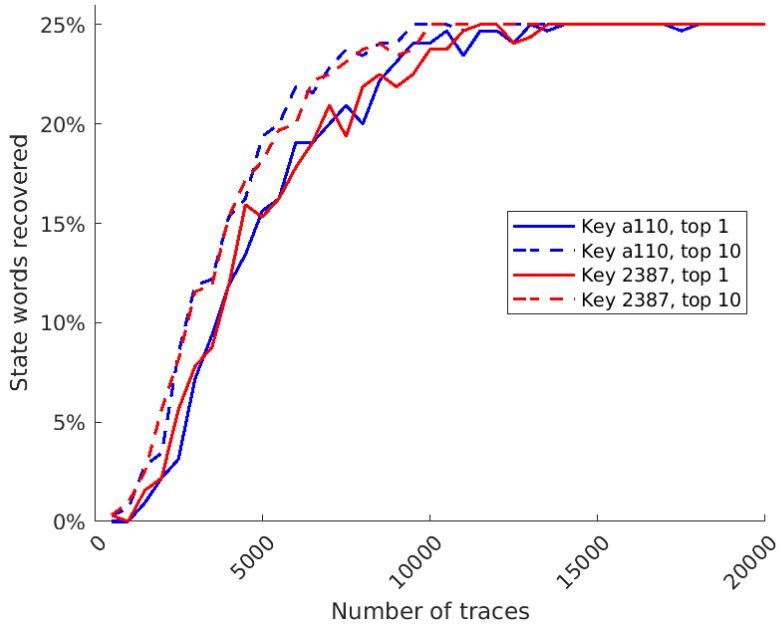

Figure 4.3: Success rate results for 10-round registered variant at various trace counts.

this attack, meaning that in ten out of ten iterations, all eight words were correctly identified.

Next, a similar test was performed on the unregistered implementation. Figure 4.4 shows the success rate plot for this test.

In comparison with the previous test, this version was much less successful, demonstrating that this implementation of the algorithm is more resistant to the rate attack. Even with the maximum trace count increased to 40,000, both keys hit plateaus and did not progress any further with additional traces. Key 2387 hit this plateau with only one out of the eight words correct, while Key a110 plateaued at two words. The reason for this difference between keys is that, with the S-box output register removed, there is no synchronized point at which the modeled values, the S-box outputs, are all changing. Instead, these transitions are spread out over several samples, with their timing depending on the specific values used as inputs.

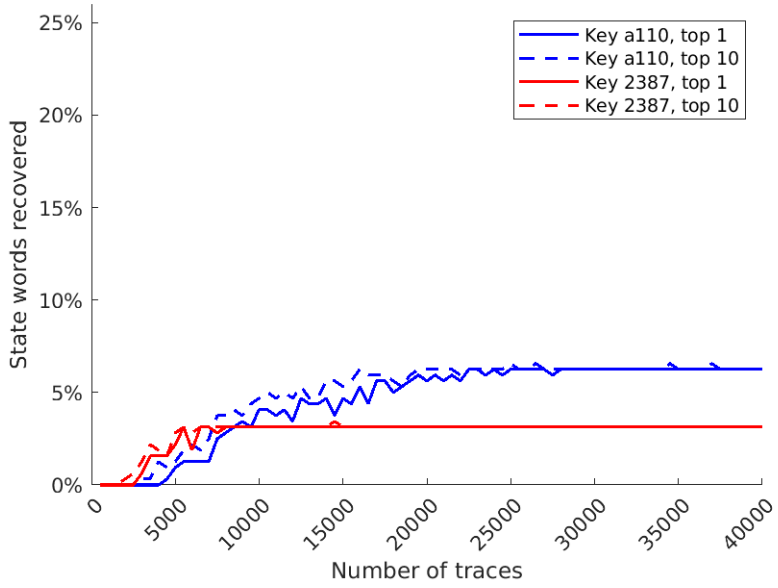

Figure 4.4: Success rate results for 10-round unregistered variant at various trace counts.

4.2 Capacity Attack

This section discusses the results of the capacity attack, covering both the theoretical computations of its limits and the proof-of-concept hardware attack.

First, Figure 4.5 shows the results of the hardware attack, performed against the 10-round registered implementation with the state register reset in between rounds and using 150,000 non-averaged traces.

This plot is very similar to those in Figures 4.1 and 4.2, with the sample on the X-axis and correlation on the Y-axis. Because this attack is not word-based, and the 15 target bits were attacked in a single group, there is only one plot. As before, the correct result is highlighted in green, and if there were an incorrect peak, it would be highlighted in red. The empty space under the correlation spikes is inconsequential; there are still correlation measurements present, but they could not be rendered on the plot due to the number of lines.

As seen from this result, the proof-of-concept capacity attack was able to recover the 15 targeted bits. However, this particular set of bits does not meet Condition 3

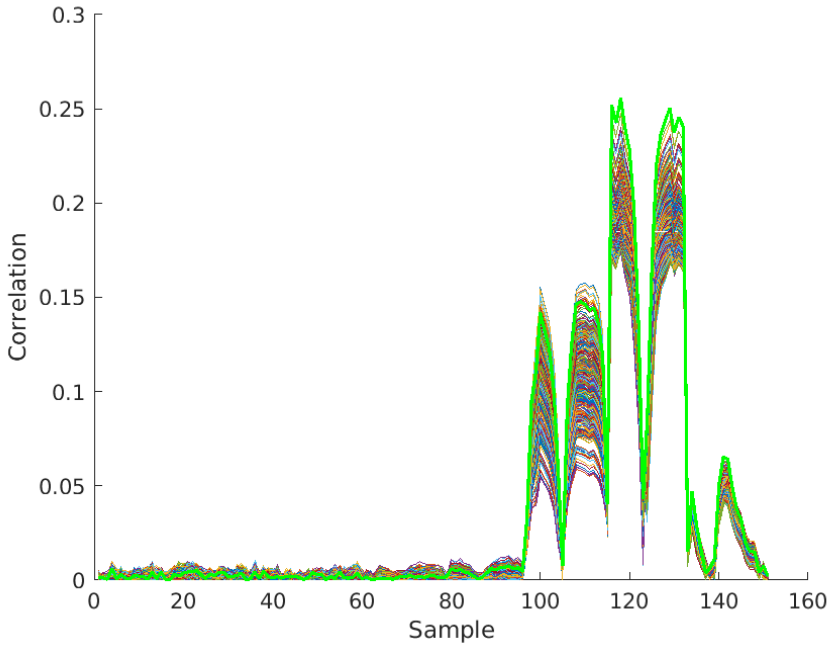

Figure 4.5: Capacity PoC results for 10-round registered variant with reset using 150,000 traces.

from Section 3.3.2.2; therefore, both the correct value and its inverse appeared with identical peak correlations, independent of the number of traces used. There is still a gap between these two values and the next-highest pair, so even with this limitation, an attacker would be able to narrow down the results to two possible values.

After performing the proof-of-concept attack, computations were performed as described in Section 3.3.2.2 to calculate the limits of this type of attack. Using the relevant Python script, data was collected for all 4080 irreducible polynomials in each of the 16 mixers. Table 4.1 provides a summary of these results, listing the polynomials resulting in the lowest and highest numbers of recoverable bits in the case that the same polynomial is applied to all 16 mixers.

As seen from this table, the number of recoverable bits is highly dependent on the number of nonzero coefficients in the polynomial. This is because, as seen from Section 2.1.2.3, each nonzero coefficient introduces one additional dependency to the mixer outputs, potentially bringing together a rate and a capacity bit which previously

Table 4.1: Mixer polynomials with lowest and highest upper bounds on potentially recoverable capacity bits, if all mixers are configured identically.

Polynomial (hex)	Polynomial (formula)	Recoverable Bits
`0x9003`	$x^{16} + x^{15} + x^{12} + x + 1$	102
`0x5003`	$x^{16} + x^{14} + x^{12} + x + 1$	104
`0x8013`	$x^{16} + x^{15} + x^4 + x + 1$	105
`0x6009`	$x^{16} + x^{14} + x^{13} + x^3 + 1$	108
`0x200d`	$x^{16} + x^{13} + x^3 + x^2 + 1$	109
⋮	⋮	⋮
`0xfefb`	$x^{16} + x^{15} + x^{14} + x^{13} + x^{12} + x^{11} + x^{10} + x^9 + x^7 + x^6 + x^5 + x^4 + x^3 + x + 1$	193
`0xfefd`	$x^{16} + x^{15} + x^{14} + x^{13} + x^{12} + x^{11} + x^{10} + x^9 + x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + 1$	193
`0xff5f`	$x^{16} + x^{15} + x^{14} + x^{13} + x^{12} + x^{11} + x^{10} + x^9 + x^8 + x^6 + x^4 + x^3 + x^2 + x + 1$	193
`0xffeb`	$x^{16} + x^{15} + x^{14} + x^{13} + x^{12} + x^{11} + x^{10} + x^9 + x^8 + x^7 + x^6 + x^5 + x^3 + x + 1$	194
`0xffed`	$x^{16} + x^{15} + x^{14} + x^{13} + x^{12} + x^{11} + x^{10} + x^9 + x^8 + x^7 + x^6 + x^5 + x^3 + x^2 + 1$	194

had no interaction. As described earlier, the exact number of recoverable bits also depends on the specifics of how each mixer is connected to the S-box outputs through the permutation stage and how this aligns with the bits from the polynomial. For the default polynomial, $q(x) = x^{16} + x^5 + x^3 + x^2 + 1$ (`0x002d`), the upper bound of recoverable capacity bits is 119. The minimum upper bound occurs with the polynomial `0x9003`, while the maximum upper bound occurs with the two polynomials seen at the end of Table 4.1.

In addition to the table, a histogram was also produced for the case where all mixers use the same polynomial, shown in Figure 4.6.

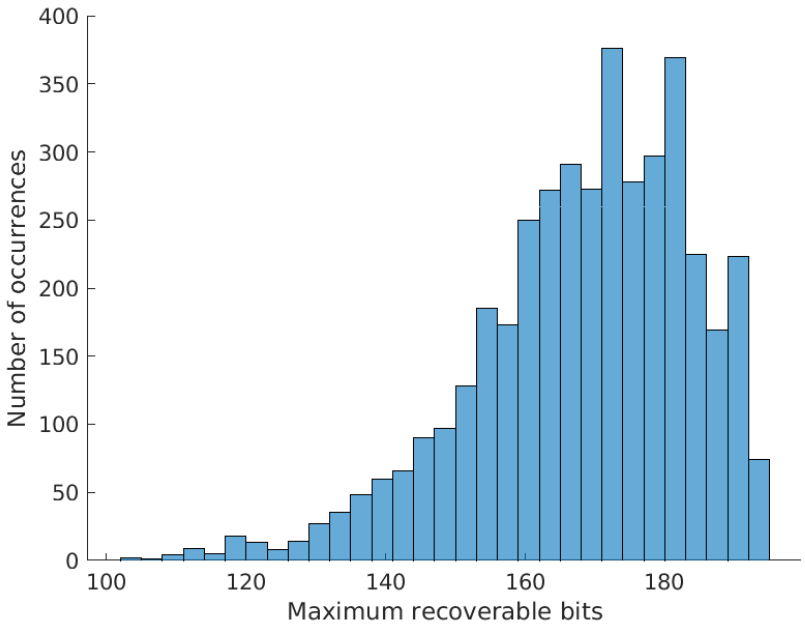

Figure 4.6: Histogram of maximum recoverable capacity bits across all irreducible mixer polynomials.

From this histogram, we can see that most polynomials will result in a relatively large number of recoverable bits, while there are not as many with lower recoverable bit counts. This is due to the fact that most of the irreducible polynomials do not have a particularly low number of coefficients, as would be expected for even a randomly-selected 16-bit polynomial.

It is not possible to enumerate all 4080^{16} combinations of mixers using different

polynomials, and even the full 4080x16 table is too large to include in this document. However, because each mixer depends on a separate set of 32 S-box outputs, mixers can be evaluated independently with each of the polynomials, and then the best or worst results combined to find an overall maximum or minimum. For example, when the mixer configurations are entirely unrestricted, the overall number of total recoverable bits can range from 88 to 194. Table 4.2 shows the polynomials of the arrangement with the smallest number of recoverable bits.

Table 4.2: Mixer configuration producing the lowest possible number of recoverable capacity bits.

Mixer Number	Polynomial (hex)	Recoverable Bits
0	`0x04c1`	5
1	`0x0a81`	5
2	`0x0a03`	5
3	`0x0807`	5
4	`0x040b`	5
5	`0x0807`	5
6	`0x002b`	5
7	`0x002b`	5
8	`0x002b`	5
9	`0x002b`	5
10	`0x002b`	5
11	`0x3801`	6
12	`0x5003`	6
13	`0x200d`	8
14	`0x8013`	7
15	`0x002b`	6

As expected, this configuration uses polynomials with very few nonzero coefficients, in total producing 88 recoverable capacity bits.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

Overall, the power analysis attacks presented in this work do not pose a significant threat to the security of MK-3. In the worst case, the rate attack is completely successful, the mixers are configured in a way that produces the highest possible number of recoverable bits, and the capacity attack is able to successfully recover all of these bits. This results in a total of $128+194 = 322$ bits of the state being revealed, leaving 190 unknown bits which the attacker would need to obtain by another method, likely brute-force attempts. Even in this worst case, the 190 remaining bits are still more than the 128-bit key option of MK-3. If a 256-bit key is in use, then this scenario provides a 66-bit advantage.

Because the number of recoverable bits is dependent on the mixer polynomials specified in customization, careful selection of these polynomials can reduce the total count from a perfectly successful attack down to $128 + 88 = 216$ bits, leaving 296 bits of the state left unknown. This provides no advantage to an attacker even in comparison to brute-forcing a 256-bit key.

As seen in Section 4.1, removing the S-box output register significantly reduces the effectiveness of the rate attack with its current model. In a non-pipelined implementation of MK-3, this register would not exist, and even in such a pipelined

implementation, there would be no reason to reset the register in between each round.

Finally, there are several design constraints and countermeasures, at the hardware, algorithm, and system levels, some of which can be easily applied even to existing designs.

5.1.1 Security Recommendations

Power side-channel attacks typically require physical access to the device in question, so one of the simplest ways to combat these attacks is to design the device to prevent this physical access. For instance, a form of tamper protection could detect if the device's case is opened and erase any stored encryption keys from memory. With the device unable to operate, and the desired information no longer present, these attacks would no longer be possible.

Another possible countermeasure, specifically for CPA, is to implement a key rolling or key derivation scheme in which the key provided to MK-3 is periodically changed. CPA depends on the key to remain the same as many different IVs are absorbed; if the key changes, then any previously collected traces cannot be used with those collected with the new key. An important caveat to this method is that the key change must be enforced from within the device. The objective is to prevent more than a certain number of IVs from being absorbed with the same key, so if key rolling is merely an operational policy rather than a hardware requirement, it does not have any benefit against CPA as an attacker could simply choose to introduce extra IVs. Additionally, if using a form of key derivation in hardware, then this hardware should also be protected from side-channel attacks.

Creating only partially-random IVs, for example by using a counter, is also helpful for preventing CPA. This obscures any key-power relationships based on the seldom-changing bits until enough traces have been collected that all cases eventually occur.

As discussed in Section 4.2, the customization parameters can be tuned for resis-

tance against the capacity attack. With mindful selection of the mixer polynomials, even the theoretical maximum for this attack can be reduced past the point of being useful for any attacker.

There are also several implementation aspects to consider. The use of registers within the round function has the potential to make power side-channel attacks easier to perform, especially if they are often reset. This should be avoided if possible.

A final suggestion is to remember that the attacks presented here focus on the absorption of the IV specifically, not of any other block such as a message. This means that longer messages are desirable, extending the time in between IV absorptions and, if implemented, reducing the required frequency of key rolling.

5.2 Future Work

One large area of potential future work is the implementation and testing of countermeasures. Section 2.5 describes many options; the majority of hardware countermeasures focus on ASIC designs, but the FPGA hardware countermeasures as well as higher-level options could be explored using the current infrastructure.

The analysis process is also not entirely complete, with the capacity attack only developed as a proof-of-concept and the set selection problem still needing a solution before it can be fully implemented. If this problem were solved, it would also provide exact information on the limitations of the capacity attack and the ideal selection of customization parameters, rather than only the upper bounds presented here.

There are also other potential improvements to the analysis process, such as reimplementing the analysis scripts in a different language to improve their performance with high trace counts. Another possibility is to develop a power model optimized for the unregistered MK-3 implementation, focusing more on the internal power consumption of the S-box and mixer components and less on their inputs and outputs.

Bibliography

[1] M. Kelly, "Design and cryptanalysis of a customizable authenticated encryption algorithm," Master's thesis, Rochester Institute of Technology, August 2014.

[2] E. De Mulder, T. Eisenbarth, and P. Schaumont, "Identifying and eliminating side-channel leaks in programmable systems," *IEEE Design & Test*, vol. 35, no. 1, pp. 74–89, 2018.

[3] M. Mayhew and R. Muresan, "An overview of hardware-level statistical power analysis attack countermeasures," *Journal of Cryptographic Engineering*, vol. 7, no. 3, pp. 213–244, Sep 2017.

[4] D. Agrawal, B. Archambeault, J. R. Rao, and P. Rohatgi, "The EM side-channel(s)," in *Cryptographic Hardware and Embedded Systems — CHES 2002*, B. S. Kaliski, Ç. K. Koç, and C. Paar, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 29–45.

[5] D. Das, M. Nath, B. Chatterjee, S. Ghosh, and S. Sen, "STELLAR: A generic EM side-channel attack protection through ground-up root-cause analysis," in *2019 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2019, pp. 11–20.

[6] E. Biham and A. Shamir, "Differential fault analysis of secret key cryptosystems," in *Advances in Cryptology — CRYPTO '97*, B. S. Kaliski, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 513–525.

[7] M. Kelly, A. Kaminsky, M. Kurdziel, M. Łukowiak, and S. Radziszowski, "Customizable sponge-based authenticated encryption using 16-bit S-boxes," in *MILCOM 2015 - 2015 IEEE Military Communications Conference*, 2015, pp. 43–48.

[8] D. F. Stafford, "Correlation power analysis of MK-3 and countermeasures," Project report, Rochester Institute of Technology, July 2017.

[9] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "The Keccak SHA-3 submission, version 3," January 2011.

[10] ——, "Cryptographic sponge functions," January 2011.

[11] P. Bajorski, A. Kaminsky, M. Kurdziel, M. Łukowiak, and S. Radziszowski, "Customization modes for the Harris MK-3 authenticated encryption algorithm," in *MILCOM 2018 - 2018 IEEE Military Communications Conference (MILCOM)*, 2018, pp. 1–5.

[12] P. Bajorski, M. Łukowiak, S. Radziszowski, P. Fabinski, and M. Millar, "Enhanced security analysis of the MK-3 authenticated encryption algorithm," L3Harris Technologies and Rochester Institute of Technology, Tech. Rep., August 2021.

[13] C. A. Wood, "Large substitution boxes with efficient combinational implementations," Master's thesis, Rochester Institute of Technology, Aug 2013.

[14] C. A. Wood, S. P. Radziszowski, and M. Łukowiak, "Constructing large S-boxes with area minimized implementations," in *MILCOM 2015 - 2015 IEEE Military Communications Conference*, 2015, pp. 49–54.

[15] N. I. of Standards and Technology, "SHA-3 standard: Permutation-based hash and extendable-output functions," U.S. Department of Commerce, Washington, D.C., Tech. Rep. Federal Information Processing Standards Publications (FIPS PUBS) 202, 2015.

[16] M. Tehranipoor and C. Wang, *Introduction to Hardware Security and Trust*. New York, NY: Springer New York, 2012.

[17] P. Kocher, J. Jaffe, B. Jun, and P. Rohatgi, "Introduction to differential power analysis," *Journal of Cryptographic Engineering*, vol. 1, no. 1, pp. 5–27, April 2011.

[18] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Advances in Cryptology — CRYPTO' 99*, M. Wiener, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 388–397.

[19] P. C. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems," in *Advances in Cryptology — CRYPTO '96*, N. Koblitz, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 104–113.

[20] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown: Reading kernel memory from user space," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018.

[21] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," in *40th IEEE Symposium on Security and Privacy (S&P'19)*, 2019.

[22] K. J. Meritt, "Differential power analysis study and experimental results," Graduate project paper, Rochester Institute of Technology, May 2012.

[23] E. Brier, C. Clavier, and F. Olivier, "Correlation power analysis with a leakage model," in *Cryptographic Hardware and Embedded Systems - CHES 2004*, M. Joye and J.-J. Quisquater, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 16–29.

[24] G. Goodwill, B. Jun, J. Jaffe, and P. Rohatgi, "A testing methodology for side-channel resistance validation," in *Non-Invasive Attack Testing Workshop*. NIST, September 2011.

[25] A. Althoff, J. Blackstone, and R. Kastner, "Holistic power side-channel leakage assessment: Towards a robust multidimensional metric," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2019, pp. 1–8.

[26] T. Schneider and A. Moradi, "Leakage assessment methodology: A clear roadmap for side-channel evaluations," in *Cryptographic Hardware and Embedded Systems – CHES 2015*, T. Güneysu and H. Handschuh, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 495–513.

[27] K. Papagiannopoulos, O. Glamocanin, M. Azouaoui, D. Ros, F. Regazzoni, and M. Stojilovic, "The side-channel metric cheat sheet," Cryptology ePrint Archive, Report 2022/253, 2022.

[28] Y. Yao, P. Kiaei, R. Singh, S. Tajik, and P. Schaumont, "Programmable RO (PRO): A multipurpose countermeasure against side-channel and fault injection attacks," Cryptology ePrint Archive, Paper 2021/878, 2021.

[29] G. Gordon, "Study and experimental results of AES countermeasures," Graduate project paper, Rochester Institute of Technology, May 2012.

[30] K. Tiri, M. Akmal, and I. Verbauwhede, "A dynamic and differential CMOS logic with signal independent power consumption to withstand differential power analysis on smart cards," in *Proceedings of the 28th European Solid-State Circuits Conference*, 2002, pp. 403–406.

[31] M. Bucci, L. Giancane, R. Luzzi, and A. Trifiletti, "Three-phase dual-rail pre-charge logic," in *Cryptographic Hardware and Embedded Systems - CHES 2006*, L. Goubin and M. Matsui, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 232–241.

[32] K. Tiri and I. Verbauwhede, "A logic level design methodology for a secure DPA resistant ASIC or FPGA implementation," in *Proceedings Design, Automation and Test in Europe Conference and Exhibition*, vol. 1, 2004, pp. 246–251 Vol.1.

[33] R. Velegalati and J.-P. Kaps, "DPA resistance for light-weight implementations of cryptographic algorithms on FPGAs," in *2009 International Conference on Field Programmable Logic and Applications*, 2009, pp. 385–390.

[34] C. Tokunaga and D. Blaauw, "Securing encryption systems with a switched capacitor current equalizer," *IEEE Journal of Solid-State Circuits*, vol. 45, no. 1, pp. 23–31, 2010.

[35] J. Wang, C. Sung, and C. Wang, "Mitigating power side channels during compilation," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2019. New York, NY, USA: Association for Computing Machinery, 2019, p. 590–601.

[36] L. Goubin and J. Patarin, "DES and differential power analysis: The "duplication" method," in *Cryptographic Hardware and Embedded Systems*, Ç. K. Koç and C. Paar, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 158–172.

[37] J. D. Golic, "Techniques for random masking in hardware," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 54, no. 2, pp. 291–300, 2007.

[38] S. Nikova, C. Rechberger, and V. Rijmen, "Threshold implementations against side-channel attacks and glitches," in *Information and Communications Security*, P. Ning, S. Qing, and N. Li, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 529–545.

[39] B. Bilgin, B. Gierlichs, S. Nikova, V. Nikov, and V. Rijmen, "A more efficient AES threshold implementation," in *Progress in Cryptology – AFRICACRYPT 2014*, D. Pointcheval and D. Vergnaud, Eds. Cham: Springer International Publishing, 2014, pp. 267–284.

[40] W. Diehl, A. Abdulgadir, J.-P. Kaps, and K. Gaj, "Side-channel resistant soft core processor for lightweight block ciphers," in *2017 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, 2017, pp. 1–8.

[41] ——, "Comparing the cost of protecting selected lightweight block ciphers against differential power analysis in low-cost FPGAs," in *2017 International Conference on Field Programmable Technology (ICFPT)*, 2017, pp. 128–135.

[42] W. Diehl, A. Abdulgadir, F. Farahmand, J.-P. Kaps, and K. Gaj, "Comparison of cost of protection against differential power analysis of selected authenticated ciphers," in *2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2018, pp. 147–152.

[43] A. Moradi, A. Poschmann, S. Ling, C. Paar, and H. Wang, "Pushing the limits: A very compact and a threshold implementation of AES," in *Advances in Cryptology – EUROCRYPT 2011*, K. G. Paterson, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 69–88.

[44] M. Abdalla, S. Belaïd, and P.-A. Fouque, "Leakage-resilient symmetric encryption via re-keying," in *Cryptographic Hardware and Embedded Systems - CHES 2013*, G. Bertoni and J.-S. Coron, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 471–488.

[45] G. Werner, S. Farris, A. Kaminsky, M. Kurdziel, M. Łukowiak, and S. Radziszowski, "Implementing authenticated encryption algorithm MK-3 on FPGA," in *MILCOM 2016 - 2016 IEEE Military Communications Conference*, 2016, pp. 1225–1230.

[46] D. F. Stafford, "Evaluating performance and efficiency of a 16-bit substitution box on an FPGA," Master's thesis, Rochester Institute of Technology, June 2021.

[47] A. Abdulgadir, W. Diehl, and J.-P. Kaps, "An open-source platform for evaluation of hardware implementations of lightweight authenticated ciphers," in *2019 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, 2019, pp. 1–5.

[48] C. O'Flynn and Z. D. Chen, "ChipWhisperer: An open-source platform for hardware embedded security research," in *Constructive Side-Channel Analysis and Secure Design*, E. Prouff, Ed. Cham: Springer International Publishing, 2014, pp. 243–260.