5-2022

Side-Channel Analysis on Post-Quantum Cryptography Algorithms

Tristen Teague

Side-Channel Analysis on Post-Quantum Cryptography Algorithms

Side-Channel Analysis on Post-Quantum Cryptography Algorithms

An Undergraduate Honors College Thesis

in the

Department of Computer Science and Computer Engineering
College of Engineering
University of Arkansas
Fayetteville, Ar
May, 2022

by

Tristen Teague

Abstract

The advancements of quantum computers brings us closer to the threat of our current asymmetric cryptography algorithms being broken by Shor's Algorithm. NIST proposed a standardization effort in creating a new class of asymmetric cryptography named Post-Quantum Cryptography (PQC). These new algorithms will be resistant against both classical computers and sufficiently powerful quantum computers. Although the new algorithms seem mathematically secure, they can possibly be broken by a class of attacks known as side-channels attacks (SCA). Side-channel attacks involve exploiting the hardware that the algorithm runs on to figure out secret values that could break the security of the system. The third round of the PQC standardization put some emphasis on the algorithm's ability to mitigate side-channel attacks. In this work, two candidate KEM algorithms Kyber and Saber are analyzed through a multi-platform setup. Both unprotected and protected implementations on Cortex-M4 microcontrollers through masking are analyzed using the test vector leakage assessment with an oscilloscope and a ChipWhisperer tool

THESIS DUPLICATION RELEASE

I hereby authorize the University of Arkansas Libraries to duplicate this thesis when needed for research and/or scholarship.

Agreed_____
Tristen Teague

Refused_____
Tristen Teague

ACKNOWLEDGEMENTS

I'd like to first thank the University of Arkansas Honors College for supporting this research through the Honors College Research Grant.

I would like to thank my advisor Dr. Alexander Nelson for mentoring and guiding me through this work. I would also like to thank Dr. Miaoqing Huang, Dr. David Andrews, Tendayi Kamucheka, and Michael Fahr for giving me helpful insight and ideas during this project.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 Introduction

In the Information Age, the Internet has assumed a role as the dominant form of information exchange. Large swathes of the global economy depend on reliable, efficient, and secure information exchange. The last criterion is dependent on cryptography – the branch of computing and mathematics that allows for exchange of information between trusted parties that is indecipherable by an outsider. Modern cryptography is a set of algorithms that establish secure communication in a variety of use cases. Asymmetric (or public-key) cryptography allows for the establishment of secure communications channels without a prior arrangement of a password between two parties. Similarly, digital signature algorithms allow for a single party to "sign" a message that can then be verified by other parties as having to originate from the signing party. These two aspects of cryptography are very important to our modern computing infrastructure, but they are demonstrated to be theoretically broken by a sufficiently large quantum computer.

Cryptography is a field that encompasses a mathematical problem to transform the secret data into something that an outsider cannot decipher without further analysis. One simple early example of cryptography would be the Caesar cipher. The cipher would take in the message and shift each letter up or down the alphabet a certain number of times. A first glance at the message would make it seem unintelligible without further analysis. With today's computing power, the Caesar cipher could easily be broken through sheer guessing also known as a brute-force attacks or an exhaustive search.

Current cryptography algorithms are much more mathematically complex and often involve a problem that would take a large amount of time to brute-force through classical computers. Some examples of commonly currently used cryptography algorithms are Advanced Encryption Standard (AES) and RSA (Rivest-Shamir-Adleman). AES is a symmetric block cipher that involves using one single key for both encryption and decryption of the message. RSA falls under public key cryptography where one key (public key) is used for encryption and is public and another key (secret key) is private and is used for decryption. RSA relies on a mathematical problem where two large prime numbers are multiplied together. It's relatively easy to compute the product of the two values but extremely difficult to factor the product apart into the two numbers. Thus, the security of RSA relies on the difficulty of factoring the product apart.

Quantum computers may unfortunately make much of our current cryptography algorithms become under threat. Shor's algorithm [1] showed that if

we were able to create a sufficiently powerful quantum computer, one could utilize its quantum mechanics to solve two problems in polynomial time. Those two problems are the factoring problem and the discrete logarithm problem. Many popular current asymmetric cryptography algorithm, such as the previously mentioned algorithm RSA, rely on these problems, meaning that their security is in danger. NIST proposed creating a new class of cryptography algorithms named Post-Quantum Cryptography (PQC) which are resistant against the threats of quantum computers [2].

Although the mathematical problems found behind the PQC algorithms seem mathematically secure, a class of attacks known as side-channel attacks may prove to be a threat in the security. Side-channel attacks involve the hardware that the cryptography algorithm is running on rather than attacking the algorithm itself. Algorithms such as RSA still see side-channel attacks being performed and new countermeasures are still being made against them even though the algorithm was made around the 1990s.

It's best that we start side-channel analysis on the newer PQC algorithms early before the algorithms are standardized considering older algorithms are still encountering problems with side-channels. In this paper, side-channel analysis is performed on two finalist key encapsulation mechanisms (KEM) CRYSTALS-Kyber and Saber on the Arm Cortex-M4 platform measured by either the ChipWhisperer platform or an oscilloscope to see the comparison and how well they do with PQC algorithms. The Test Vector Leakage Assessment (TVLA) is utilized to evaluate if possible leakage occurs in side-channel measurements. One thing to note is that this paper does not contain a key recovery attack but rather attempts to show that there's possible leakage that may be able to be exploited.

2 Background

2.1 Post-Quantum Cryptography

If a sufficiently strong quantum computer were to be built, cryptography may lose some of its security. Two quantum algorithms that may cause a threat are named Shor's algorithm and Grover's algorithm [3]. Grover's algorithm helps speed up the brute-force attack on symmetric cryptography algorithms such as AES in square root time. It is suggested that we double the key size on symmetric algorithms to negate the speedup obtained from Grover's algorithm [4]. Shor's algorithm performs problems relating to factorization and discrete logarithms in polynomial time. Unfortunately, doubling the key sizes on algorithms under threat by Shor's algorithm will not be sufficient enough. Therefore, a new class of algorithms named Post-Quantum Cryptography (PQC) will take over the standard of asymmetric cryptography in the PQC Standardization process proposed by NIST. These new algorithms will be quantum-resistant against sufficiently large quantum computers and will replace the standard for both digital signatures and key encapsulation mechanisms (KEM).

2.1.1 Round 3 Candidates

As of round 3 of the NIST PQC Standardization, there are four finalists for KEM schemes and three finalists for signature schemes. For the alternative schemes, there are five KEM schemes and three signature schemes [5]. The finalists schemes are ones most likely to be standardized and the alternates are potential candidates for standardization. Candidates are either lattice-based, hash-based, code-based, or multivariate based.

The three finalists signature schemes are CRYSTALS-Dilithium [6], FALCON [7], and Rainbow [8]. Dilithium is a Fiat-Shamir structured lattice based on Module-Learning with Errors (M-LWE), FALCON is based on the short integer solution (SIS) problem over a NTRU lattice, and Rainbow is a multivariate scheme based on the Oil-Vinegar problem. The alternative signature schemes are GeMSS [9], Picnic [10], and SPHINCS+ [11]. GeMSS is a multivariate-based scheme that uses the Hidden Field Equations, Picnic is a zero knowledge proof scheme that is based on symmetric primitives (block cipher and secure hash function), and SPHINCS+ is a stateless hash-based scheme.

The four finalists for the KEM schemes are Classic McEliece [12], CRYSTALS-KYBER [13], NTRU [14], and SABER [15]. Classic McEliece is a Goppa code-based algorithm that has been around for many years but can stand up against

quantum computers utilizing larger parameters to obtain stronger bits of security. The rest of the three finalists are all structured lattice-based schemes. Kyber uses the Module-Learning with Errors (M-LWE) problem, NTRU uses the shortest vector problem (SVP), and SABER uses the Module-Learning with Rounding (M-LWR) problem. The alternatives for the KEM schemes are BIKE [16], FrodoKEM [17], HQC [18][19], NTRUprime [20], and SIKE [21]. BIKE and HQC are both code-based schemes, FrodoKEM is a structured lattice-based scheme, NTRUprime is a unstructured lattice-based scheme, and SIKE is a isogeny-based elliptic curve scheme. In this work, side-channel analysis is conducted only on two lattice finalists Kyber and Saber. The reason for this being that the algorithms are closely related, meaning that very few changes were needed to implement on the ChipWhisperer platform.

2.1.2 CRYSTALS-Kyber

Kyber is a lattice-based PQC algorithm based on the hardness of the M-LWE problem. Kyber offers a chosen-plaintext attack (CPA) secure public-key encryption (PKE) scheme and a chosen-ciphertext attack (CCA) secure KEM. Kyber consists of three different security parameters named Kyber512, Kyber768, and Kyber1024, where its post-quantum bits of security is 107, 166, and 232 respectively [22]. Round 3 parameters involved are n, k, q, n_1, n_2, d_u, and d_v, where n is 256 and q is 3329 for every security parameter. Every other parameter is based on the variant of Kyber. In this paper, side-channel analysis in only conducted on the Kyber512 variant due to low resources available on the embedded system used to run the algorithm. The rest of the parameters in Kyber512 are $k = 2, n_1 = 3, n_2 = 2, d_u = 10$, and $d_v = 4$. The specific implementation used in this paper originates from [23] and was used in the PQM4 [24] library which consists of PQC algorithms specifically made for ARM Cortex-M4 microcontrollers. Kyber's PKE consists of three main functions named key generation, encryption, and decryption which can be seen in Appendix A.

2.1.3 SABER

Saber is a lattice-based PQC algorithm algorithm based on the hardness of the M-LWR problem. Saber has a CCA secure PKE along with a CCA secure KEM. The three variants of Saber are LightSaber, Saber, and FireSaber and offers 107, 172, and 236 bits of post-quantum security [25]. The parameters involved with Saber are n, l, q, p, T, and μ. For every variant, $q = 2^{13}$ and $n = 256$. The variant measured in this paper is Saber which has parameters $l = 2, T = 2^4$, and $\mu = 8$. The implementation measured in this paper is the multi-moduli NTTs for Saber [26] which was specifically made for both the Cortex-M4 and Cortex-M3

platform and is also found in the PQM4 library. Saber's PKE also consists of three main functions which are key generation, encryption, and decryption. All three functions are shown in Appendix B.

2.2 Side-Channels

Many of these algorithms seem to be mathematically secure but are not fully secure against side-channel attacks. Common side-channel attacks are simple power analysis, differential power analysis, timing attacks, correlation power analysis, and fault injections. Side-channel attacks are still conducted on much older algorithms such as AES and RSA, meaning that research in the side-channel field will be on-going for much time. Since it's still the early phase of PQC, much analysis is needed in the side-channel field. Many side-channel vulnerabilities have already been discovered relating to PQC algorithms [27][28][29][30]. It's best that we attempt to implement countermeasures against both current and future attacks before these algorithms become standardized.

2.2.1 Simple Power Analysis

Simple power analysis (SPA) involves looking at a power waveform measurement and gaining information on that alone. If unprotected, one could collect a power measurement of a cryptography algorithm and distinguish when a certain operation occurs. It may even be possible to figure out secret values from the peaks of the waveform. Some computations can also yield larger power draw such as a multiplication against addition. Branches in conditionals can also be obvious on the waveform which can yield information about a secret value. One early example of this attack was mentioned in [31] which showed that the early cryptography algorithm DES was very susceptible to this attack. If one measured the DES algorithm, they could clearly see which round is occurring in the waveform. One could also see if a jump occurred based off the power difference as a jump shows a higher power draw. One previously mentioned paper [28] conducted SPA on a round 2 PQC candidate algorithm named NewHope. The attack involved measuring NewHope's message encoding and could tell the difference using the power waveform if the message bit contained either a zero or one.

2.2.2 Differential Power Analysis

Differential power analysis (DPA) involves analyzing many power waveform measurements and using statistics to learn information from them. This allows attackers to learn more information previously not available from a simple visual examination. Errors such as noise or measurement errors can be reduced if the

number of observed power waveforms increase. The same paper by Kocher et al. [31] conducted DPA on the early cryptography algorithm DES and was able to learn forty-eight out of fifty-six bits of the key through power analysis alone. DPA exploited the fact that values are correlated and if enough traces were captured, their values would either approach zero or one depending if they're correlated or not. If the key guess on a bit was correct, it would approach one since it's correlated, making a spike in the resulting statistic used. Since errors such as noise or measurement errors are not correlated to the computation, they should approach zero as the number of traces increase, meaning that errors should not affect the differential trace if enough traces are used. Unfortunately, DPA sometimes fails due to a problem named ghost peaks. When guessing the sub-keys, false values in the differential trace can lead to incorrect sub-key guesses, meaning that more traces are possibly needed to circumnavigate this problem. One source of this problem could be that certain bits may be partially correlated with the target bit that DPA is trying to guess [32]. The whole reason why DPA works is because correlated values tend to go towards values one and uncorrelated values go to values zero. If the bit is partially correlated, say one of four bits in the S-box substitution, this it will give a false peak value. The next power analysis method actually avoids this problem and could be seen as an improvement.

2.2.3 Correlation Power Analysis

Correlation power analysis (CPA) shares common characteristics with DPA such as requiring multiple traces and using statistics. One thing to note with CPA is that it may require less traces to complete compared to DPA. DPA often requires a large amount of traces (possibly up the thousands range) to be completed to actually guess the correct sub-key values. CPA may require as little as fifty traces to guess the secret key for AES. CPA also does not have to deal with the ghost peaks problem that DPA suffers with. CPA uses Pearson's correlation coefficient (seen in equation 2.1) with data sets X and Y.

$$\rho_{X,Y} = \frac{\text{covariance}(X, Y)}{\sigma_X \sigma_Y} \tag{2.1}$$

Data set X will consist of the measured traces and data set Y will consist of the leakage model picked. One example of a leakage model used in CPA is hamming weight, which was used in the ChipWhisperer CPA example. Hamming weight involves measuring the number of ones in a string of bits. One example is given below with the string of bits 010111.

$$HammingWeight(010111) = 4$$

In the case of using CPA on AES, one could use this attack on encryption by specifically attacking the S-box substitution. Data set Y consists of calculating the hamming weight of using the plaintext associated with a certain trace, doing an XOR operation with a guessed byte of the sub-key, and using that value into the input of the S-box function. Each guess (0..256) will have a correlation value associated with it and the max value of those guesses will be our guess for the byte of the sub-key. This repeats for all 16 bytes of the AES-128 key and we should have a key guess that may/may not be the actual key. The success rate will depend on the leakage model and the number of traces obtained.

2.2.4 Test Vector Leakage Assessment

One way to test whether or not a cryptography implementation leaks information to side-channel attacks is called the test vector leakage assessment (TVLA) [33]. TVLA uses a two-tailed statistical test named Welch's t-test. The null hypothesis of this test is that the two means of the data sets are the same and have no significant difference. Since this test is also a t-test, we assume unequal variance between the two sets of data. The calculation for a single t-value can be seen in equation 2.2.

$$t = \frac{\mu_A - \mu_B}{\sqrt{\frac{s_A^2}{n_A} + \frac{s_B^2}{n_B}}}, \tag{2.2}$$

The variables found in the test consists of μ, s, and n which are the mean, sample standard deviation, and the sample size respectively. The selected α value for TVLA is 0.00001, giving us a 99.999% confidence level to reject the null hypothesis.

In order to apply this statistical test to side-channel analysis, the two sets of data must be ordered with respect to time. The two sets of data A and B will consist of a total of D power traces named P which each consist of k samples. Each trace is lined-up according to time (which is samples in this case). For each t-value, you will use the k_{th} sample of P for every trace with $n = D$. For example given a data set A below:

$$\begin{aligned} P_0 &= [4, 2, 2, 7, 4, \ldots, k] \\ P_1 &= [2, 1, 2, 8, 4, \ldots, k] \\ &\vdots \\ P_{D-1} &= [3, 1, 2, 7, 4, \ldots, k] \end{aligned}$$

To find the t-value for k = 0, we will look at the first element of each trace

$(4, 2, \ldots, 3)$ P for all traces D. We will calculate μ_A and s_A by taking each 0th element of all traces P and calculate both the mean the sample standard deviation. We will do the same for a data set B and calculate the t-value for the k=0 element. TVLA consists of conducting Welch's t-test k times and if the t-value at k exceeds the critical value (often being ± 4.5), we reject the null hypothesis.

The two datasets will depend on what variables are being used in the scenario that the t-test is trying to test the difference on. One document that described using TVLA [33] mentions how one should conduct the TVLA on the AES algorithm and some examples. Many times, we will have one dataset having fixed variables and the other dataset having random variables. In the case of this paper, the non-specific fixed vs. random TVLA is conducted. The shared variables between both datasets will be the public key and the secret key. With Kyber, I measured a function in the decryption stage, meaning I use variables that relate to that function. The decryption stage requires the secret key and a generated ciphertext from encryption. The fixed set of dataset will consist of multiple traces of decryption with the same secret key and ciphertext every time. The random set of data involves the same secret key as the fixed set of data but will have differently generated ciphertexts every iteration.

2.2.5 Masking Countermeasure

Masking is one of the countermeasures that is used against side-channel attacks. The side-channel attacks often involve exploiting the power to learn information such as a message or a shared secret. If one were to change the value of the information we want to hide by some value, then if the attacker learns the information, they will not learn the actual value. This idea was developed in [34] in 1999 and has been improved upon ever since. Masking involves taking a sensitive variable and splitting it into shares. For $d+1$ shares, the masking will be a d-order masking implementation. Say we have a variable x and we want to implement first-order masking, we will split x into two different shares such that [35]:

$$x = x_1 \circ x_2,$$

These shares can either be done with arithmetic or boolean masking. Arithmetic masking involves using some sort of arithmetic operation such as addition often in the form $x = x_1 + x_2$. Boolean masking involves splitting the shares by using boolean operations (normally an XOR) such that the shares are in the form $x = x_1 \oplus x_2$. Randomization is used in one of the shares which is a random mask $x_1 = r$ such that computing the other share is found by $x_2 = x \oplus r$ for boolean masking and $x_2 = x - r$ for arithmetic masking. By doing this, an attacker learning information from side-channel analysis cannot find any information pertaining to

x, effectively making the algorithm secure against first-order statistical tests.

3 Experimental Setup

All measurements were completed using a ChipWhisperer-Lite (CW1173) connected to a CW308 UFO target board. The ChipWhisperer platform is an open source hardware security tool specifically designed for side-channel analysis. The ChipWhisperer captures samples using synchronous sampling, which involves synchronizing the sample clock to the targets clock [36]. This allows the ChipWhisperer platform to capture at a much lower sampling-rate (96 MS/s) and still being capable of matching asynchrounous sampling such as an oscilloscope at a much higher sampling-rate (2 GS/s). The two victims boards utilized on the target board involve the STM32F405RGT6 (CW308T-STM32F4) or the STM32F303RCT7 (CW308T-STM32F3). The STM32F4 board was used when a true random number generator (TRNG) was required or when a larger RAM size (192KB) is needed. If the two previous requirements weren't needed, then the STM32F3 was used as the victim board instead. All victim boards run at a clock frequency of 7.37 MHz.

Communication between the capture board (ChipWhisperer-Lite) and the PC involved using the ChipWhisperer API on Python. The ChipWhisperer API allows easy usage to both program and control the capture board. Jupyter Notebooks was utilized to allow easy capturing of traces, running the cryptography algorithm, and analyzing the data obtained from the traces on either the oscilloscope or the ChipWhisperer platform. Communication between the capture board and the target board used ChipWhisperer's serial communication SimpleSerial V2 protocol. On the newer version of the ChipWhisperer, the SimpleSerial V2 protocol is deprecated as of September 14, 2021 due to incorrect CRC usage. The incorrect CRC usage made reading the secret key of the Saber algorithm throw errors, meaning the key cannot be read. The newer SimpleSerial V2.1 was released to fix the CRC issue and allowed the key to be correctly printed but many issues occurred while using the STM32F4 victim board. With all measurements, the ChipWhisperer 5.5.2 version is used with SimpleSerial V2.

Two limiting factors of the ChipWhisperer platform is its buffer size for samples and the limited size of data for sending communication between the capture and target board. The ChipWhisperer-Lite contains a sample buffer size of 24,400 samples, meaning that large functions wanting to be measured (such as the whole decryption step) of a PQC algorithm may not be possible. Downsampling the sample-rate may allow a trace to fit into the sample buffer but it is not tested whether or not it will affect results. This means that the ChipWhisperer platform is good for measuring smaller functions such as a step of NTT multiplication

```
97  uint8_t getBuffer(uint8_t cmd, uint8_t scmd, uint8_t len, uint8_t* in) {
98  //     led_ok(1);
99      int offset = scmd * BYTE_PACKET_SIZE;
100     int size = BYTE_PACKET_SIZE;
101     //Returns pk
102     if(choice == 0) {
103         if(CRYPTO_PUBLICKEYBYTES-offset < BYTE_PACKET_SIZE)
104             size = CRYPTO_PUBLICKEYBYTES-offset;
105         simpleserial_put('r',size,pk + offset);
```

Figure 3.1: C Code Snippet of Getting Buffer Code

```
58  uint8_t setBuffer(uint8_t cmd, uint8_t scmd, uint8_t len, uint8_t* in) {
59
60      int offset = scmd * BYTE_PACKET_SIZE;
61
62      if(choice == 0)
63      {
64          for(int i = 0; i < len; i++) {
65              pk[i + offset] = in[i];
66          }
67      }
```

Figure 3.2: C Code Snippet of Setting Buffer Code

or converting the polynomials to a message. For stable communication between the capture and target board, it is suggested to use 128-byte payloads so that information is not lost. SimpleSerial V2 allows up to 192-bytes but many issues occurred whilst attempting to use that size. PQC algorithm keys tend to be much larger than keys used by RSA and AES and often have keys that are larger than 128-bytes, such as Kyber512's secret key being 1632-bytes long. To send a large amount of data from each other, one must use SimpleSerial V2+ to be able to use the subcommand variable. Using the subcommand variable, we can specify which 128-byte part of the buffer that either needs to be read from or written to. Since there's multiple buffers to work with (such as public key, secret key, ciphertext), a separate SimpleSerial command was used to manipulate a global variable which chooses which buffer to work with. A code snippet of getting data from a buffer and setting information to a buffer on the firmware of the victim board in C can be seen in Figure 3.1 and Figure 3.2 and is used as a SimpleSerial command.

The specific oscilloscope used in this paper was the Tektronix oscilloscope MDO34 3-BW-200. For the measurements with the oscilloscope, two passive probes were connected to each side of the shunt resistor on the CW308 UFO

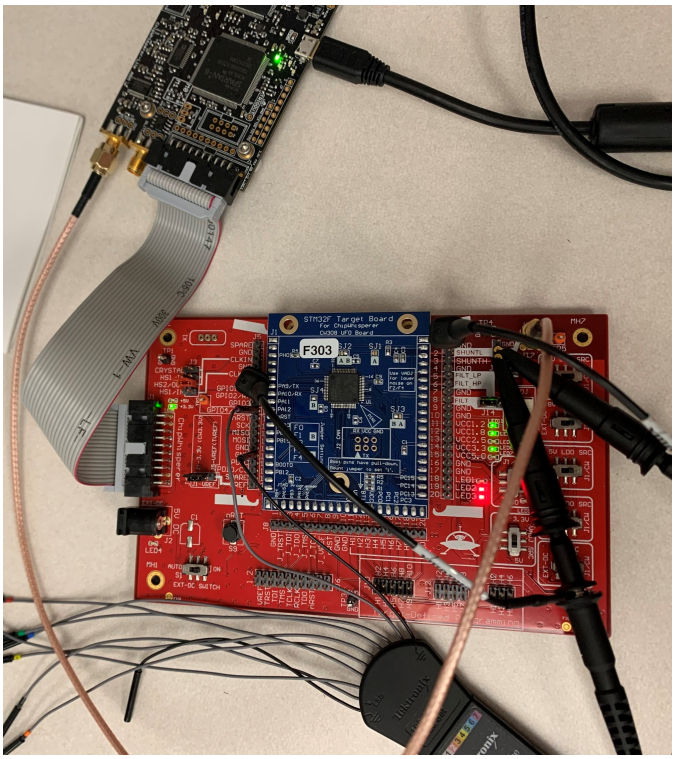

Figure 3.3: ChipWhisperer Probe Setup

board labled SHUNTL and SHUNTH (see Figure 3.3). A math function on the oscilloscope was used to take the difference between both sides of shunt to measure the voltage difference. When measuring Kyber, the math function approach was used and the resulting waveform looked well. For measuring Saber, the resulting math waveform was digitized with a small range of values, so only SHUNTL was measured to have traces similar to Kyber. A logic probe is directly connected to GPIO4 pin which by default acts as the trigger. When wanting to measure a specific function, GPIO4 pin is raised high to start the measurement and raised low to signify that the function is over in the firmware. This logic probe will allow us to start the measurement when the victim board gets to the function wanted. Communication between the PC and the oscilloscope was done by using PyVisa in the Jupyter Notebook that is used to command the capture board. This allows the user to directly command the oscilloscope in Python and also pull the data off the waveform onto the PC that can be analyzed later.

3.1 Kyber Setup

Every measurement with Kyber involved using the STM32F303 victim board which had no support for true-random number generation (TRNG). The pseudo-random number generator (PRNG) used in this paper was the "randombytes" implementation found in the PQM4 library. With the TVLA, the fixed dataset consisted of a fixed secret key and a valid fixed ciphertext ciphertext. With the random dataset, the same fixed secret key was used and a different psudeo-random ciphertext was used for every trace. The TVLA test conducted in this paper consisted of 2000 traces total (1000 fixed traces and 1000 random traces). The function measured in all Kyber implementations is the "poly_tomsg" function which can be found inside the decryption step. The unnmasked implementation of Kyber was taken from the PQM4 library[1] and is specifically made for the Cortex-M4 platform. The masked implementation of Kyber which implements a masked decoder is taken from [37]. The masked implementation is originally based on the reference Kyber which contained C code and also contained C++ files. To deal with memory issues, I converted the masked C++ functions into C, took the reference C code that the masked functions used, and used the PQM4 implementation for the rest of the functions. Therefore, the masked implementation in this paper will be a combination of the masked reference code and the PQM4 implementation. Another note is that the masked implementation specifically runs PKE rather than the KEM process.

The process for the TVLA on the unmasked Kyber implementation with N traces is as follows:

1. Run KEM keypair once
2. Run KEM encapsulation once to generate fixed ciphertext
3. For fixed dataset ($N/2$ times):
   (a) Run KEM decapsulation and measure poly_tomsg function from decryption every iteration
   (b) Compare $key_a == key_b$ to check to see if operation is successful
4. For random dataset ($N/2$ times):
   (a) Run KEM encapsulation to get random ciphertext every iteration
   (b) Run KEM decapsulation and measure poly_tomsg function from decryption every iteration

[1]This specific Kyber implementation used in the paper from the PQM4 library around 2021 and has been replaced with a newer version.

(c) Compare $key_a == key_b$ to check to see if operation is successful

The process for the TVLA on the masked Kyber implementation with N traces is as follows:

1. Run PKE keypair once
2. Give fresh mask to secret shares
3. Run PKE encryption once to generate fixed ciphertext
4. For fixed dataset ($N/2$ times):
   (a) Give fresh mask to secret shares every iteration
   (b) Run PKE decryption and measure poly_tomsg function every iteration
   (c) Compare $key_a == key_b$ to check to see if operation is successful
5. For random dataset ($N/2$ times):
   (a) Give fresh mask to secret shares every iteration
   (b) Run PKE encryption to get random ciphertext every iteration
   (c) Run PKE decryption and measure poly_tomsg function from decryption every iteration
   (d) Compare $key_a == key_b$ to check to see if operation is successful

3.2 Saber Setup

Every measurement with Saber involved using the STM32F405 which offers TRNG support. Whenever randomness was required from either "random-bytes" or masking, the TRNG was utilized. The TVLA consisted of a fixed dataset with a fixed secret key and ciphertext and the random dataset involved the same fixed secret key with truly-random ciphertexts for every trace. A total of 2000 traces were collected with 1000 fixed traces and 1000 random traces. Two functions measured on the Saber algorithm was the inverse-NTT step of decryption and the Keccak absorb step found within decapsulation. For comparison between oscilloscope and the ChipWhisperer, only the Keccak absorb step is analyzed and the inverse-NTT is only measured on the ChipWhisperer. The unmasked implementation of Saber was taken from the PQM4 library which originates from [26] which is optimized for the Cortex-M4 platform and the stack. The masked implementation also comes from the previously mentioned paper and is optimized for stack and the Cortex-M4 platform. The implementations aim to improve the polynomial multiplication speed by replacing the original Toom-Cook multiplications

with multi-moduli NTTs. One difference between setting up Kyber and Saber is that the two implementations of Saber utilize the FPU on the Cortex-M4.

The process for the TVLA on the unmasked multi-moduli NTT Saber implementation with N traces is as follows:

1. Run KEM keypair once
2. Run KEM encapsulation once to generate fixed ciphertext
3. For fixed dataset ($N/2$ times):
   (a) Run KEM decapsulation and measure either Keccak absorb step from decapsulation or the inverse-NTT step from decryption every iteration
   (b) Compare $key_a == key_b$ to check to see if operation is successful
4. For random dataset ($N/2$ times):
   (a) Run KEM encapsulation to get random ciphertext every iteration
   (b) Run KEM decapsulation and measure either Keccak absorb step from decapsulation or the inverse-NTT step from decryption every iteration
   (c) Compare $key_a == key_b$ to check to see if operation is successful

The process for the TVLA on the masked multi-moduli NTT Saber implementation with N traces is as follows:

1. Run KEM keypair once
2. Give fresh mask to secret shares
3. Run KEM encapsulation once to generate fixed ciphertext
4. For fixed dataset ($N/2$ times):
   (a) Run KEM decapsulation and measure either Keccak absorb step from decapsulation or the inverse-NTT step from decryption every iteration
   (b) Compare $key_a == key_b$ to check to see if operation is successful
   (c) Give fresh mask to secret shares every iteration
5. For random dataset ($N/2$ times):
   (a) Run KEM encapsulation to get random ciphertext every iteration
   (b) Run KEM decapsulation and measure either Keccak absorb step from decapsulation or the inverse-NTT step from decryption every iteration
   (c) Compare $key_a == key_b$ to check to see if operation is successful
   (d) Give fresh mask to secret shares every iteration

4 Results

The countermeasure of masking against side-channel analysis lowered the number of leakage points when comparing against unprotected implementations on the Cortex-M4 according to the TVLA's conducted. The summary of the results can be found in Table 4.1 and Table 4.2 which show the results of the ChipWhisperer and oscilloscope respectively. The next two sections will go into detail about the exact setup for each measurement system and the amount of samples that were used for the TVLA.

Table 4.1: TVLA Results of ChipWhisperer.

Algorithm	Unmasked Failed Samples of CW(%)	Masked Failed Samples of CW(%)
Kyber	93.2%	0.073%
Saber(inverse-NTT)	27.1%	0.12%[1]
Saber(Keccak)	5.83%	0.034%[2]

Table 4.2: TVLA Results of Oscilloscope.

Algorithm	Unmasked Failed Samples of Oscilloscope(%)	Masked Failed Samples of Oscilloscope(%)
Kyber	31.2%	0.11%
Saber(Keccak)	0.31%	0%

4.1 Kyber ChipWhisperer Measurements

For the unmasked Kyber512 implementation, the ChipWhisperer was clocked at four times the frequency of the victim board sampling at 29.5 MS/s. A total of

[1]Does not measure whole inverse-NTT operation, only the first 24,400 samples.
[2]Averaged the 3 and 4 failed samples of the two TVLA results.

19,104[3] samples were collected and therefore fit into the buffer without any downsampling. Out of the 19,104 samples, 17,806 total samples failed the t-test. All sample points that pass the t-test are drawn in orange. The resulting TVLA can be found in Figure 4.1.

For the masked decoder Kyber512 implementation, the ChipWhisperer was clocked at a sampling rate of 7.37 MS/s with a total of 397,545 samples. Since the total amount of samples is way over the buffer size (∼24,400), all traces had to be downsampled by 17, making all traces have 23,385 samples total. Out of the 23,385 samples, a total of 97 samples failed the t-test. All sample points that pass the t-test are drawn in orange. The TVLA of the masked decoder of Kyber512 on the ChipWhisperer can be found in Figure 4.2.

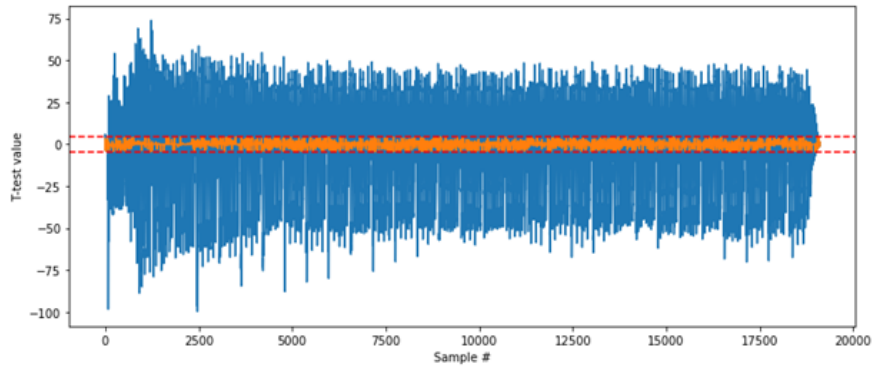

Figure 4.1: TVLA on Unmasked Kyber512 "poly_tomsg" from ChipWhisperer.

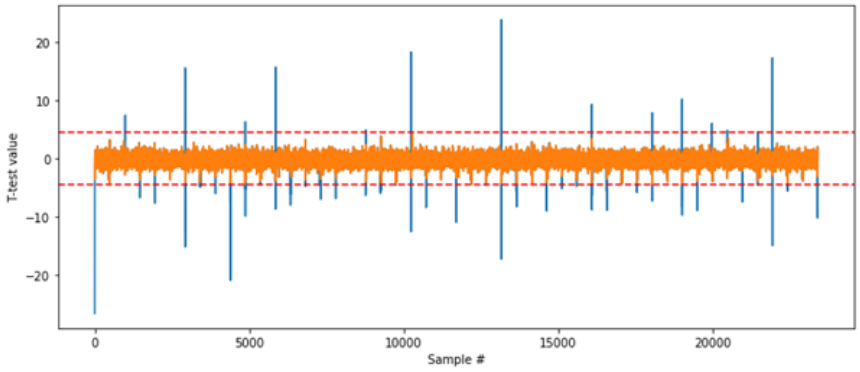

Figure 4.2: TVLA on Masked Kyber512 "poly_tomsg" from ChipWhisperer

[3]Sample size of the unmasked trace varied depending on the ciphertext used. The max trace size out of the 2000 collected traces was 19,104 traces long.

4.2 Kyber Oscilloscope Measurements

For the unmasked implementation of Kyber512, the oscilloscope measured at a sampling rate of 100 MS/s. A total of 100,000 samples were collected per trace with the triggered section having 63,947 samples. Out of the triggered section, a total of 19,961 samples failed the t-test. The resulting TVLA can be seen in Figure 4.3.

The masked decoder implementation of Kyber512 with the oscilloscope had a sampling rate of 25 MS/s. A total of 5,000,000 samples were collected per trace with the triggered section having a total of 1,345,943 samples. Out of the 1,345,943 triggered samples, a total of 1,419 samples failed the t-test. The TVLA result can be seen in Figure 4.4.

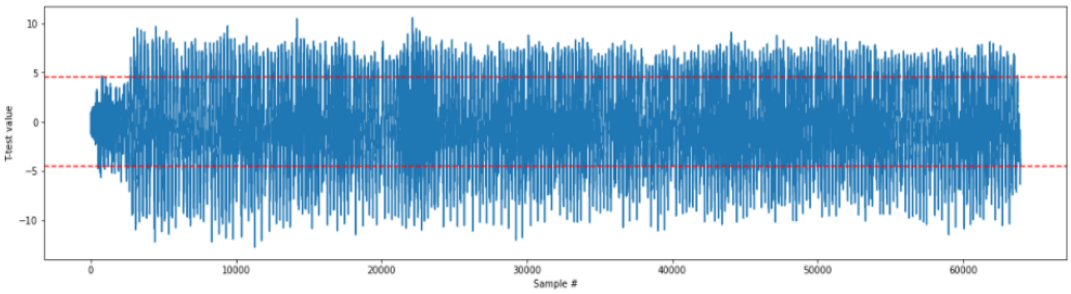

Figure 4.3: TVLA on Unmasked Kyber512 "poly_tomsg" from Oscilloscope.

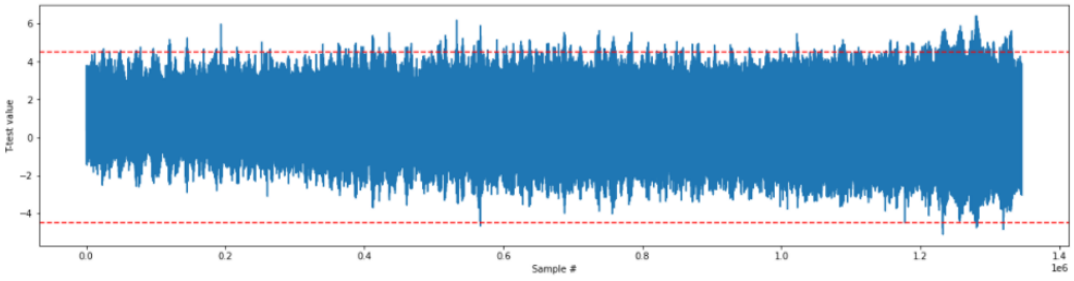

Figure 4.4: TVLA on Masked Kyber512 "poly_tomsg" from Oscilloscope

4.3 Saber ChipWhisperer Measurements

With the unmasked Saber measuring the Keccak Absorb step inside decapsulation, the ChipWhisperer measured at 29.5 MS/s. A total of 4,168 samples were collected and fit into the buffer, so downsampling was not needed. Out of

the 4,168 samples, a total of 243 samples failed. The TVLA results can be seen in Figure 4.5.

For measuring the unmasked inverse-NTT step inside decryption, a sampling rate of 7.37 MS/s. 14,476 samples were collected with no downsampling. A total of 3,928 samples failed the t-test out of the 14,476 samples. The TVLA plot can be seen in Figure 4.6.

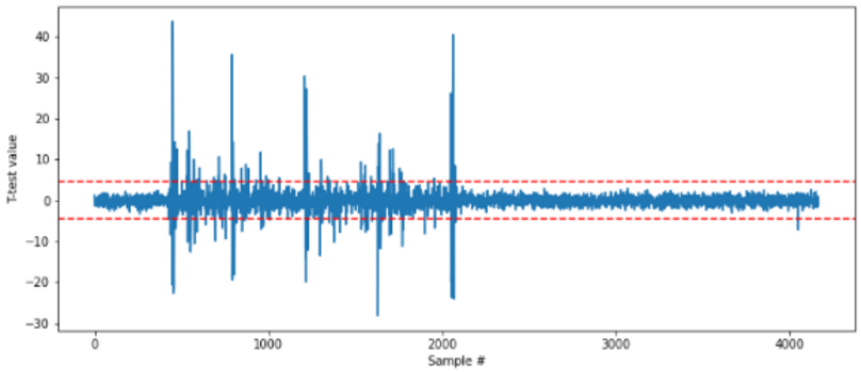

Figure 4.5: TVLA on Unmasked Saber Keccak Absorb step from ChipWhisperer.

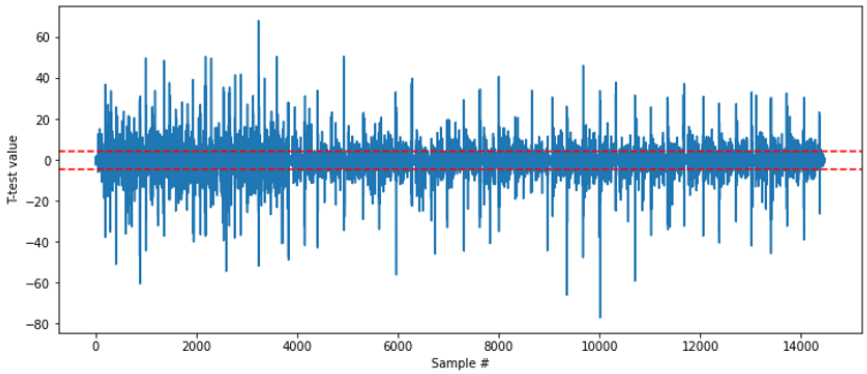

Figure 4.6: TVLA on Unmasked Saber inverse-NTT step from ChipWhisperer.

For the masked Keccak Absorb step, the ChipWhisperer sampled at 7.37 MHz with 10,404 samples total. Two different TVLA's were conducted due to the spike found in the first TVLA result to attempt to remove any false results. On the first TVLA, a total of 4 samples failed. That TVLA result can be see in Figure 4.7. On the second TVLA, a total of 3 samples failed and can be seen in Figure 4.8.

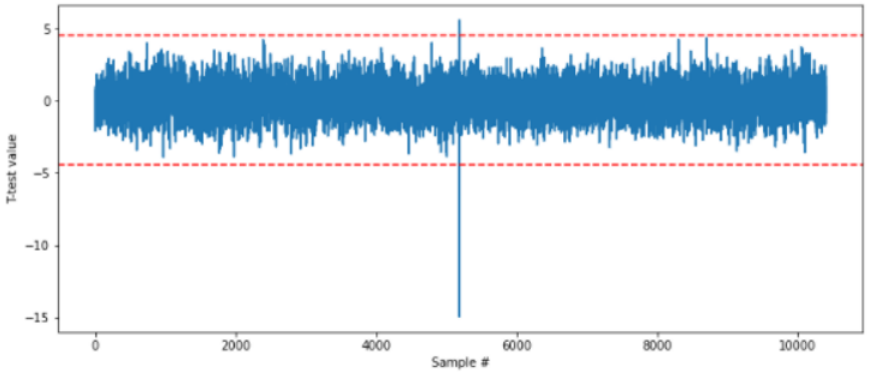

Figure 4.7: First TVLA on Masked Saber Keccak Absorb step from ChipWhisperer.

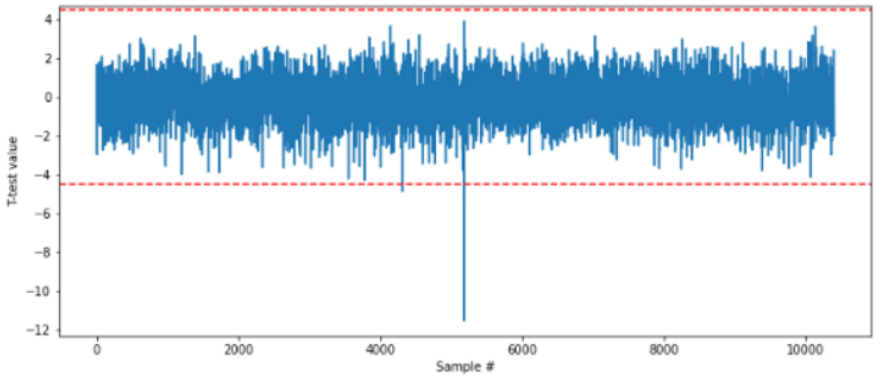

Figure 4.8: Second TVLA on Masked Saber Keccak Absorb step from ChipWhisperer.

The ChipWhisperer measuring inverse-NTT from masked Saber inside decryption is sampled at 29.5 MS/s with a total of 256,268 samples. Since the total samples is well above the buffer size, the measurement is downsampled by two and only captures the start of the inverse-NTT step. This leaves the trace to be the first 24,400 samples out of the downsampled trace with length 128,134. Out of the 24,400 samples at the start of inverse-NTT, it fails 30 of the samples. The TVLA result is found in Figure 4.9.

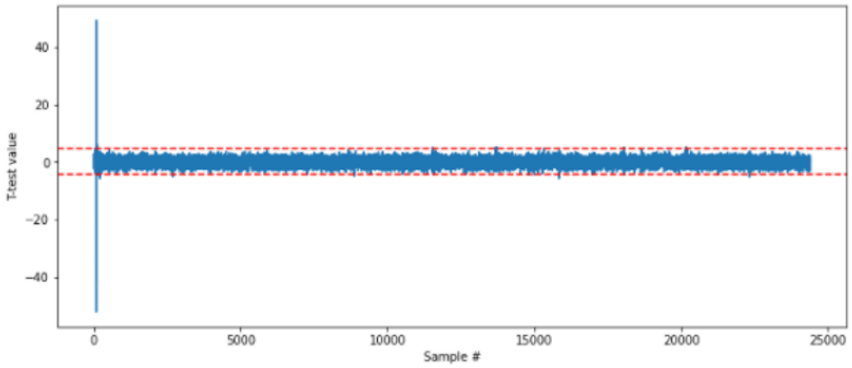

Figure 4.9: TVLA on Masked Saber inverse-NTT step from ChipWhisperer.

4.4 Saber Oscilloscope Measurements

When measuring unmasked Saber with the oscilloscope on the Keccak absorb step, the sampling rate was clocked at 500 MS/s. A total of 100,000 samples were measured, with 70,555 samples being within the triggered area. Out of the 70,555 samples, a total of 266 samples failed the t-test. The TVLA result is pictured in Figure 4.10.

The masked Saber implementation when measuring the Keccak absorb step with the oscilloscope had a sampling rate of 500 MS/s. A total of 300,000 samples were collected with the triggered section having 176,117 samples. Out of the 176,117 samples, none of the samples failed the t-test. The resulting TVLA can be seen in Figure 4.11.

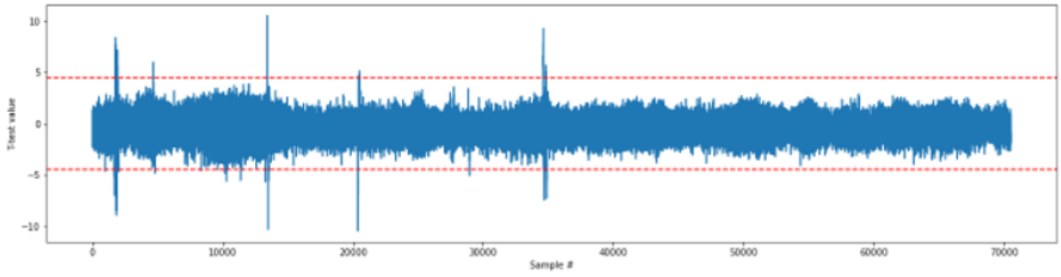

Figure 4.10: TVLA on Unmasked Saber Keccak Absorb step from Oscilloscope.

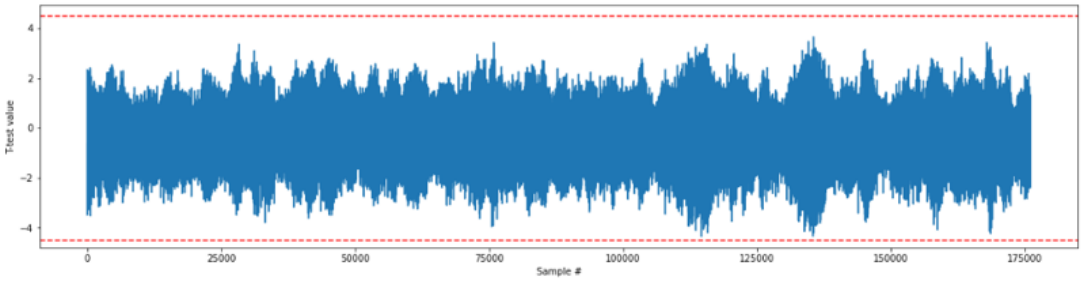

Figure 4.11: TVLA on Masked Saber Keccak Absorb step from Oscilloscope

5 Discussion

5.1 Masked Kyber Problems

The masked decoder implementation caused much trouble for both the oscilloscope and ChipWhisperer measurements. When measuring the unmasked implementation of Kyber, downsampling was not required to fit into the ChipWhisperer buffer and the oscilloscope could capture at a decent sampling rate without a significant amount of points. The masked implementation added a large amount of overhead to the point where sampling rates had to be cut down by a large amount for the ChipWhisperer and total sample points for the oscilloscope had to be increased. For the oscilloscope, a total of 1,345,943 samples had to be captured at 25 MS/s which has a sampling rate less than four times the actual victim's clock speed (7.37 MHz). The ChipWhisperer had to downsample 17 times in order to fit the trace into the sample buffer, making the sampling rate much lower than the original 7.37 MS/s. The reason for this overhead could be due to the fact that the original masking implementation is based on the reference Kyber implementation rather than being specifically designed for the Cortex-M4 like the masked Saber used in this paper. If one were to adapt this masked decoder Kyber512 implementation specifically for the M4 platform, the previously mentioned problems would not occur.

5.2 Reduction of Leakage Samples

When comparing the percentage of leakage of unmasked implementations against the masked implementations, both algorithms, no matter the measuring tool, get less leakage. On the ChipWhisperer platform, all measured parts decrease leakage by a large amount but not to zero. For the case of Kyber, a PRNG was utilized for the randomness that was involved with both the ciphertexts and the masking. It may be possible that masked Kyber could perform better on the TVLA if a TRNG were utilized for the fresh masking of the two shares.

One thing to note is that both masking implementations measured in this paper use first-order masking. The TVLA conducted in this paper only checks first-order leakage and can be extended to higher-orders [38]. It would be interesting to see how well first-order masking of both Kyber and Saber will do against higher-order leakage assessments. Another interesting idea would to see how well the first-order masking works against first-order attacks or if this leakage detected by the TVLA test can actually be exploited to gain information.

5.3 Oscilloscope and ChipWhisperer

The main difference between the oscilloscope and the ChipWhisperer is how they sample. The oscilloscope samples asynchronously, meaning that the sampling clock is not tied to the target board. The ChipWhisperer samples synchronously and the sampling clock is directly synchronized to the target board's clock. As mentioned before, Colin et al. [36] stated that a synchronous sampling rate of 96 MS/s was comparable to a asynchronous sampling rate of 2 GS/s. When comparing the results of the ChipWhisperer to the oscilloscope, the ChipWhisperer discovered much more leakage (2.99 times more on Kyber and 18.8 times more on Saber) on the unmasked implementations. Same can be said for measuring the Keccak absorb step in Saber where the oscilloscope detects no leakage at 500 MS/s while the ChipWhisperer detects a few samples in two different TVLAs at 29.5 MS/s. The same cannot be said for comparing the measurements on the masked decoder of Kyber. The oscilloscope captures more leakage compared to the ChipWhisperer. One possible reasoning for this is that the ChipWhisperer measurement was downsampled by 17, effectively making the sample rate $\sim 433.53KS/s$.

6 Conclusion and Future Work

In this paper, two PQC algorithms Kyber and Saber, each with an unmasked and a first-order masked version, are analyzed using the TVLA on the Cortex-M4 platform. The two measurement systems used involve the ChipWhisperer platform and a Tektronix oscilloscope on either the STM32F3 or STM32F4 microcontroller. Since the ChipWhisperer measures synchronously, it can measure leakage at a much lower sampling-rate than the oscilloscope and still has comparable results. The ChipWhisperer-Lite has a limited buffer size of roughly 24,400 samples, meaning that only smaller functions of the PQC algorithms can be measured. Although the ChipWhisperer suffers with buffer size, it comes at a lower cost compared to a higher-end oscilloscope and still has nearly the same capability as a high sample-rate oscilloscope. The ChipWhisperer UFO board also allows easy oscilloscope usage in case the user wants to use one. All one has to do is obtain a probe and measure the pins around the shunt resistor and attach a logic probe to the trigger pin. Sometimes, one would have to do invasive actions to get a measurement of an embedded system such as soldering on a shunt resistor to a specific location we want. If one uses the ChipWhisperer UFO board, no invasive acts are needed to measure power waveforms with an oscilloscope. When performing leakage assessment, the masking countermeasure mitigates and nearly removes the number of failed samples found within both the oscilloscope and the ChipWhisperer measurements. When comparing the unmasked and masked implementations on Saber's Keccak absorb step, the number of failed samples had a reduction of nearly 172 times on the ChipWhisperer.

It would be interesting to implement every finalist and alternative algorithm in the third round on the ChipWhisperer platform for future side-channel analysis for other groups to use. As long as the memory used by the candidate algorithm fits on the STM32F3/F4 memory, it should be able to be implemented. Many of the PQM4 implementations could possibly be ported over to the ChipWhisperer platform. One finalist signature algorithm named Dilithium has been implemented in this work but has not been thoroughly tested nor does it have a comparison for a unmasked and masked implementation.

Another interesting thought is to see how well the leakage assessment fares when using different tools. In this work, passive probes were utilized to complete measurements around a shunt resistor. Some works have used a field probe directly on a chip to obtain power measurements and some other have used active probes. Passive probes are cheaper when compared to a field or an active probe. Using the latter two could yield different results in detecting leakage and possibly less

invasive measurements.

One final thought that could be explored would be actually exploiting the areas where the TVLA fails. Although TVLA shows possible leakage, it's unknown whether or not we can actually exploit it. Welch's t-test determines whether or not the difference of means between two datasets is significant or not. Although the difference may be significant, it may difficult to actually exploit or find a leakage model to learn the differences and gain information.

Bibliography

[1] P. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *Proceedings 35th Annual Symposium on Foundations of Computer Science*, 1994, pp. 124–134.

[2] L. Chen, L. Chen, S. Jordan, Y.-K. Liu, D. Moody, R. Peralta, R. Perlner, and D. Smith-Tone, *Report on post-quantum cryptography*. US Department of Commerce, National Institute of Standards and Technology, 2016, vol. 12.

[3] L. K. Grover, "A fast quantum mechanical algorithm for database search," 1996. [Online]. Available: https://arxiv.org/abs/quant-ph/9605043

[4] D. J. Bernstein, "Grover vs. mceliece," in *Post-Quantum Cryptography*, N. Sendrier, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 73–80.

[5] D. Moody and the NIST-PQC team, "Status update on the 3rd round," in *Third PQC Standardization Conference*, 2021, pp. 9–11.

[6] L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé, "Crystals-dilithium: A lattice-based digital signature scheme," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2018, no. 1, p. 238–268, Feb. 2018. [Online]. Available: https://tches.iacr.org/index.php/TCHES/article/view/839

[7] P.-A. Fouque, J. Hoffstein, P. Kirchner, V. Lyubashevsky, T. Pornin, T. Prest, T. Ricosset, G. Seiler, W. Whyte, and Z. Zhang, "Falcon: Fast-fourier lattice-based compact signatures over ntru.(2018)," *Submission to the NIST PQC project*, vol. 3, 2018.

[8] J. Ding and D. Schmidt, "Rainbow, a new multivariable polynomial signature scheme," in *Lecture Notes in Computer Science*, vol. 3531. Springer Verlag, 2005, pp. 164–175. [Online]. Available: https://link.springer.com/chapter/10.1007/11496137_12

[9] A. Casanova, J.-C. Faugere, G. Macario-Rat, J. Patarin, L. Perret, and J. Ryckeghem, "Gemss: A great multivariate short signature," *Submission to NIST*, 2017.

[10] M. Chase, D. D. , S. G. , C. O. , S. R. , C. R. , D. S. , and G. Zaverucha, "Post-quantum zero-knowledge and signatures from symmetric-key primitives," *Cryptology ePrint Archive, Report 2017*, vol. 279, March 2017. [Online]. Available: https://www.microsoft.com/en-us/research/publication/post-quantum-zero-knowledge-signatures-symmetric-key-primitives/

[11] D. J. Bernstein, A. Hülsing, S. Kölbl, R. Niederhagen, J. Rijneveld, and P. Schwabe, "The sphincs+ signature framework," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 2129–2146.

[12] R. McEliece, "A public key cryptosystem based on algebraic coding theory," 1978.

[13] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehle, "Crystals - kyber: A cca-secure module-lattice-based kem," in *2018 IEEE European Symposium on Security and Privacy (EuroS P)*, 2018, pp. 353–367.

[14] J. Hoffstein, J. Pipher, and J. Silverman, "Ntru: A ring-based public key cryptosystem," in *ANTS*, 1998, pp. 267–288.

[15] J.-P. D'Anvers, A. Karmakar, S. Sinha Roy, and F. Vercauteren, "Saber: Module-lwr based key exchange, cpa-secure encryption and cca-secure kem," in *Progress in Cryptology – AFRICACRYPT 2018*, A. Joux, A. Nitaj, and T. Rachidi, Eds. Cham: Springer International Publishing, 2018, pp. 282–305.

[16] N. Aragón, P. S. L. M. Barreto, S. Bettaieb, L. Bidoux, O. Blazy, J.-C. Deneuville, P. Gaborit, S. Gueron, T. Güneysu, C. Melchor, R. Misoczki, E. Persichetti, N. Sendrier, J. Tillich, and G. Zémor, "Bike: Bit flipping key encapsulation," 2017.

[17] J. Bos, C. Costello, L. Ducas, I. Mironov, M. Naehrig, V. Nikolaenko, A. Raghunathan, and D. Stebila, "Frodo: Take off the ring! Practical, quantum-secure key exchange from LWE," in *Proceedings of the ACM Conference on Computer and Communications Security*, vol. 24-28-October-2016. New York, NY, USA: Association for Computing Machinery, oct 2016, pp. 1006–1018. [Online]. Available: https://dl.acm.org/doi/10.1145/2976749.2978425

[18] C. A. Melchor, O. Blazy, J. Deneuville, P. Gaborit, and G. Zémor, "Efficient encryption from random quasi-cyclic codes," *CoRR*, vol. abs/1612.05572, 2016. [Online]. Available: http://arxiv.org/abs/1612.05572

[19] C. A. Melchor, N. Aragon, S. Bettaieb, L. Bidoux, O. Blazy, J.-C. Deneuville, P. Gaborit, E. Persichetti, G. Zémor, and I.-C. Bourges, "Hamming quasi-cyclic (hqc)," *NIST PQC Round*, vol. 2, pp. 4–13, 2018.

[20] D. J. Bernstein, C. Chuengsatiansup, T. Lange, and C. van Vredendaal, "Ntru prime: reducing attack surface at low cost," in *International Conference on Selected Areas in Cryptography*. Springer, 2017, pp. 235–260.

[21] D. Jao, R. Azarderakhsh, M. Campagna, C. Costello, L. De Feo, B. Hess, A. Jalili, B. Koziel, B. Lamacchia, P. Longa *et al.*, "Sike: supersingular isogeny key encapsulation," 2017.

[22] R. M. Avanzi, J. W. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé, "Crystals-kyber algorithm specifications and supporting documentation," 2021.

[23] E. Alkim, Y. Alper Bilgin, M. Cenk, and F. Gérard, "Cortex-m4 optimizations for R,M lwe schemes," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2020, no. 3, p. 336–357, Jun. 2020. [Online]. Available: https://tches.iacr.org/index.php/TCHES/article/view/8593

[24] M. J. Kannwischer, J. Rijneveld, P. Schwabe, and K. Stoffelen, "PQM4: Post-quantum crypto library for the ARM Cortex-M4," https://github.com/mupq/pqm4.

[25] A. Basso, J. P. D'Anvers, . A. Karmakar, J. M. B. Mera, S. S. Roy, M. V. Beirendonck, and F. Vercauteren, "Saber round 3 update," in *Third PQC Standardization Conference*, 2021, p. 4.

[26] A. Abdulrahman, J.-P. Chen, Y.-J. Chen, V. Hwang, M. J. Kannwischer, and B.-Y. Yang, "Multi-moduli ntts for saber on cortex-m3 and cortex-m4," Cryptology ePrint Archive, Report 2021/995, 2021, https://ia.cr/2021/995.

[27] A. Aysu, Y. Tobah, M. Tiwari, A. Gerstlauer, and M. Orshansky, "Horizontal side-channel vulnerabilities of post-quantum key exchange protocols," in *2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2018, pp. 81–88.

[28] D. Amiet, A. Curiger, L. Leuenberger, and P. Zbinden, "Defeating newhope with a single trace," Cryptology ePrint Archive, Report 2020/368, 2020, https://ia.cr/2020/368.

[29] P. Ravi, S. Sinha Roy, A. Chattopadhyay, and S. Bhasin, "Generic side-channel attacks on cca-secure lattice-based pke and kems,"

IACR Transactions on Cryptographic Hardware and Embedded Systems, vol. 2020, no. 3, p. 307–335, Jun. 2020. [Online]. Available: https://tches.iacr.org/index.php/TCHES/article/view/8592

[30] M. J. Kannwischer, P. Pessl, and R. Primas, "Single-trace attacks on keccak," Cryptology ePrint Archive, Report 2020/371, 2020, https://ia.cr/2020/371.

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

[32] 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.

[33] B. J. Gilbert Goodwill, J. Jaffe, P. Rohatgi *et al.*, "A testing methodology for side-channel resistance validation," in *NIST non-invasive attack testing workshop*, vol. 7, 2011, p. 115–136.

[34] S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi, "Towards sound approaches to counteract power-analysis attacks," in *Advances in Cryptology — CRYPTO' 99*, M. Wiener, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 398–412.

[35] K. Ngo, E. Dubrova, Q. Guo, and T. Johansson, "A side-channel attack on a masked ind-cca secure saber kem implementation," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2021, no. 4, p. 676–707, Aug. 2021. [Online]. Available: https://tches.iacr.org/index.php/TCHES/article/view/9079

[36] C. O'Flynn and Z. D. Chen, "Chipwhisperer: An open-source platform for hardware embedded security research," Cryptology ePrint Archive, Report 2014/204, 2014, https://ia.cr/2014/204.

[37] P. Pessl and L. Prokop, "Fault attacks on cca-secure lattice kems," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2021, no. 2, p. 37–60, Feb. 2021. [Online]. Available: https://tches.iacr.org/index.php/TCHES/article/view/8787

[38] T. Schneider and A. Moradi, "Leakage assessment methodology," in *Cryptographic Hardware and Embedded Systems – CHES 2015*, T. Güneysu and H. Handschuh, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 495–513.

A Kyber PKE Functions

Algorithm 1 Kyber.CPAPKE Key Generation

Input: $(\rho, \sigma) \xleftarrow{\$} \{0,1\}^{256} \times \{0,1\}^{256}$
Input: $A \leftarrow U(q)^{k \times k}$
Output: public key $pk = (\hat{\mathrm{b}},\ \rho)$
Output: secret key $sk = \hat{\mathrm{s}}$
seed $\xleftarrow{\$} \{0, ..., 255\}^{32}$
$\rho, \sigma \leftarrow$ SHAKE256(64,*seed*)
$\hat{A} \leftarrow$ GenMatrixA(ρ)
$s \leftarrow$ SampleVec($\sigma, 0$)
$e \leftarrow$ SampleVec($\sigma, 1$)
$\hat{b} \leftarrow \hat{A} \circ \mathrm{NTT}(s) + \mathrm{NTT}(e)$

Algorithm 2 Kyber.CPAPKE Encryption

Input: public key $pk = (\hat{\mathrm{b}},\ \rho)$
Input: message $\mu \in \mathcal{R}_q$
Input: seed $coin \in \{0, ..., 255\}^{32}$
Output: ciphertext $(\hat{\mathrm{u}}', h)$
A $\leftarrow$ GenMatrixA(ρ)
$s' \leftarrow$ SampleVec($coin, 0$)
$e' \leftarrow$ SampleVec($coin, 1$)
$e'' \leftarrow$ SampleVec($coin, 2$)
$\hat{t} \leftarrow \mathrm{NTT}(s')$
$u \leftarrow \mathrm{INTT}(\hat{A}^T \circ \hat{t}) + e'$
$v' \leftarrow \mathrm{INTT}(\hat{b}^T \circ \hat{t}) + e'' + \mu$
return(Compress(u), Compress(v'))

Algorithm 3 Kyber.CPAPKE Decryption

Input: ciphertext $c = (\hat{\mathrm{u}}', h)$
Input: secret key $sk = \hat{\mathrm{s}}$
Output: message $\mu \in \mathcal{R}_q$
$\mathrm{u} \leftarrow \mathrm{Decompress}(c_1, d_u)$
$v \leftarrow \mathrm{Decompress}(c_2, d_v)$
$m \leftarrow \mathrm{Compress}_q(v - \mathrm{INTT}(\hat{s} \circ \mathrm{NTT}(u)), 1)$

B Saber PKE Functions

Algorithm 4 Saber Key Generation

Output: public key $pk = (\text{seed}_A, b)$
Output: secret key $sk = \text{s}$
$\text{seed}_A \leftarrow \text{Sample}_U()$
$A \in R_q^{l \times l} \leftarrow \text{Expand}(\text{seed}_A)$
$s \in R_q^l \leftarrow \text{Sample}_B()$
$b \leftarrow \text{Round}(A^T \cdot s)$

Algorithm 5 Saber.CPAPKE Encryption

Input: m
Input: r
Input: public key $pk = (\text{seed}_A, b)$
Output: ciphertext $ct = (c, b')$
$\text{A} \in R_q^{l \times l} \leftarrow \text{Expand}(\text{seed}_A)$
$s' \in R_q^l \leftarrow \text{Sample}_B(r)$
$b' \leftarrow \text{Round}(As')$
$v' \leftarrow b^T (s' \bmod p)$
$c \leftarrow \text{Round}(v' - 2^{\varepsilon - 1} m)$

Algorithm 6 Saber.CPAPKE Decryption

Input: ciphertext $ct = (c, b')$
Input: secret key $sk = (s)$
Output: m
$\text{v} \leftarrow b'^T (s \bmod p)$
$m \leftarrow \text{Round}(v - 2^{\varepsilon_p - \varepsilon_T} c \bmod p)$