Systematic Use of Random Self-Reducibility in Cryptographic Code against Physical Attacks

Ferhat Erata
Yale University
New Haven, Connecticut, USA

TingHung Chiu
Virginia Tech
Blacksburg, Virginia, USA

Anthony Etim
Yale University
New Haven, Connecticut, USA

Srilalith Nampally
Virginia Tech
Blacksburg, Virginia, USA

Tejas Raju
Virginia Tech
Blacksburg, Virginia, USA

Rajashree Ramu
Virginia Tech
Blacksburg, Virginia, USA

Ruzica Piskac
Yale University
New Haven, Connecticut, USA

Timos Antonopoulos
Yale University
New Haven, Connecticut, USA

Wenjie Xiong
Virginia Tech
Blacksburg, Virginia, USA

Jakub Szefer
Yale University
New Haven, Connecticut, USA

ABSTRACT

This work presents a novel, black-box software-based countermeasure against physical attacks including power side-channel and fault-injection attacks. The approach uses the concept of random self-reducibility and self-correctness to add randomness and redundancy in the execution for protection. Our approach is at the operation level, is not algorithm-specific, and thus, can be applied for protecting a wide range of algorithms. The countermeasure is empirically evaluated against attacks over operations like modular exponentiation, modular multiplication, polynomial multiplication, and number theoretic transforms. An end-to-end implementation of this countermeasure is demonstrated for RSA-CRT signature algorithm and Kyber Key Generation public key cryptosystems. The countermeasure reduced the power side-channel leakage by two orders of magnitude, to an acceptably secure level in TVLA analysis. For fault injection, the countermeasure reduces the number of faults to 95.4% in average.

CCS CONCEPTS

• **Security and privacy** → **Hardware attacks and countermeasures**; *Side-channel analysis and countermeasures*; • **Software and its engineering** → Software fault tolerance.

KEYWORDS

Random Self-Reducibility, Fault Injection Attacks, Power Side-Channel Attacks, Countermeasure, NTT, PQC, RSA-CRT

ICCAD '24, October 27–31, 2024, New York, NY, USA

ACM ISBN 979-8-4007-1077-3/24/10
https://doi.org/10.1145/3676536.3689920

ACM Reference Format:
Ferhat Erata, TingHung Chiu, Anthony Etim, Srilalith Nampally, Tejas Raju, Rajashree Ramu, Ruzica Piskac, Timos Antonopoulos, Wenjie Xiong, and Jakub Szefer. 2024. Systematic Use of Random Self-Reducibility in Cryptographic Code against Physical Attacks. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD '24), October 27–31, 2024, New York, NY, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3676536.3689920

1 INTRODUCTION

Smart devices and IoT devices with sensors, processing capability, and actuators are becoming ubiquitous today in consumer electronics, healthcare, manufacturing, etc. These devices often collect sensitive or security-critical information and need to be protected. However, when deployed in the field, such devices are vulnerable to physical attackers who can have direct access to the devices.

Physical attacks can be categorized as passive attacks or active attacks. In passive attacks, such as *Side-Channel Attacks (SCA)*, the attackers do not tamper with the execution, but can collect power traces, electromagnetic (EM) field traces, or traces of acoustic signals, and analyze the signals to learn information that is processed on the device. In active attacks, such as *Fault Injection (FI) attacks*, the attackers can inject faults through a voltage glitch, clock glitch, EM field, or laser to cause a malfunction in the processing unit or memory to tamper with the execution to obtain desired results. It has been shown that both types of physical attacks have been able to break cryptography implementations to leak secret keys, for example [10, 46].

Even though the assumptions on the attacker's capability are similar for SCA and FI, the existing mitigation techniques treat the two types of attacks separately. For side-channel attacks, the mitigation techniques usually use randomness or noise to decouple the signal observable by the attacker from the data value [24, 50]. For fault injection attacks, there are typically two solutions: one is attack detection and one is to have redundancy in the execution for error correction. The detection will detect when the execution

has abnormal behavior, and then handle it as an exception. The error correction uses redundancy in the execution and uses the redundancy to correct execution error if there is [36]. However, when we consider both SCA and FI attacks in the same system, separate mitigation for the two does not protect both attacks efficiently. For example, existing work [14] showed that instruction duplication as a fault tolerance mechanism amplifies the information leakage through side channels. Detection methods such as full, partial, encrypt-decrypt duplication & comparison of a cipher [28] produce repetitions of intermediate values that are exploitable by the side-channel adversary.

In this work, we propose a joint solution for both SCA and FI attacks. With a combination of random obfuscation using the Random Self-Reducibililty (RSR) property and redundancy for error correction, our proposed countermeasure is particularly effective against FI outperforming traditional redundancy-based methods. The randomness disrupts the attacker's observation of the statistics in fault attacks, thereby nullifying the effectiveness of statistical analysis as a tool for security compromise. This aspect is crucial in the face of increasingly sophisticated FI analysis techniques. In addition to its effectiveness against FI, the countermeasure also resists SCA, by rendering power consumption variations less useful to attackers. The countermeasure significantly enhances system security, particularly in environments where physical attacks are prevalent.

The proposed countermeasure offers significant benefits as a black box *operation-level* solution to both SCA and FI attacks, and it is independent of the implementation of the target algorithm being protected. This means there is no need for detailed knowledge of the implementation. *The basis for the solution is to implement protection at low-level of operations such as modular exponentiation, modular multiplication, polynomial multiplication, and number theoretic transforms.* Also, we assume a generic fault model, and thus, there is no special fault profiling of a targeted device necessary. Therefore, the proposed protection techniques can be applied directly in software without extensive system-specific adjustments. In our evaluation, we showcase how the proposal protection techniques can be adopted to protect two different cryptosystems.

Our protection requires a small number of steps to implement. It can be implemented at C or high-level and is independent of the compiler or underlying architecture; assuming the compiler. First, target software is identified. Second, we locate low-level operations such as modular exponentiation, modular multiplication, polynomial multiplication, or number theoretic transforms. These operations can be protected with the idea of Random Self-Reducibility (RSR). Each instance of the low-level operation is replaced with an equivalent RSR operation. Each RSR operation requires querying a randomness source and then executing the low-level operations multiple times with original input values modified with the random values. Typically, multiple RSR operations are instantiated and majority voting is performed on the output of RSR operations. Because the protection works at the low-level operations such as modular exponentiation, modular multiplication, polynomial multiplication, or number theoretic transforms, it is independent of the higher-level algorithm or application. Since it does not rely on any hardware tricks, it is independent of the architecture and agnostic to the underlying compiler.

Our protection can be applied to any program or algorithm that uses modular exponentiation, modular multiplication, polynomial multiplication, and number theoretic transforms to process secret or sensitive information. This encompasses major cryptogrpahic algorithms from ElGamal [18] and RSA [43] to post-quantum cryptography such as Kyber [5] and Dilithium [17]. In our evaluation, we show how our protection can be applied to RSA-CRT and Kyber's Key Generation algorithms. Our contributions are summarized as follows:

- We propose a new software-based countermeasure against power side-channel (Section 3.2) and fault injection (Section 3.3) attacks, by randomizing the intermediate values of the computation using the notion of random self-reducibility (Section 3).
- We formalize the security of the countermeasure in relation to an attacker's fault injection capability, parameterize it, and quantify its effectiveness against fault-injection attacks (Section 3.4).
- End-to-end implementation of the countermeasure for RSA-CRT and Kyber's Key Generation public key cryptosystems (Section 4).
- Emprical evaluation of the countermeasure against power side-channel and fault-injection attacks over modular exponentiation, modular multiplication operations, polynomial multiplication, number theoretic transform (NTT) operations, RSA-CRT, and Kyber's Key Generation (Section 5).

2 BACKGROUND AND THREAT MODEL

Computing devices today are vulnerable to physical attacks such as side-channel and fault injection attacks to leak critical information. It is a well-known fact that the power consumption during certain stages of a cryptographic algorithm exhibits a strong correlation with the Hamming weight of its underlying variables, i.e., Hamming weight leakage model [11, 25]. This phenomenon has been widely exploited in the cryptographic literature in various attacks targeting a broad range of schemes, particularly post-quantum cryptographic implementations [3, 20, 22, 47, 51]. Therefore, we use the Hamming weight leakage model in the evaluation of the robustness of the countermeasure.

Test Vector Leakage Assessment (TVLA) [21] identifies if two sets of side channel measurements are *distinguishable* by computing the Welch's t-test for the two sets of measurements. It is being used in the literature to confirm the *presence* or *absence* of side-channel leakages for power traces, and has become the de facto standard in the evaluation of side-channel measurements [42, 45]. In side-channel analysis, the recommended thresholds for t-values are specifically tailored to detect potential information leakage in cryptographic systems. A t-value threshold of ±4.5 or ±5 is often considered in side-channel analysis. This threshold corresponds to a very high confidence level, rejecting the null hypothesis with a confidence greater than 99.999% for a significantly large number of measurements. The null hypothesis typically being that all samples are drawn from the same distribution, a t-value outside this range indicates distinguishable distributions of the two sets and thus the existence of side-channel leakage [49]. The choice of these thresholds is influenced by the need to balance the risk of false positives (incorrectly identifying information leakage when there is none) against the risk of false negatives (failing to detect actual information leakage).

In the real world, there is a possibility that the devices will malfunction or be damaged, resulting in generating the error output, and we may ignore it. However, if the attacker intentionally induced the fault during the device operation, e.g., cryptographic calculation, he or she can recover the secret by analyzing the original and fault outputs. Most of the classical cryptographic algorithms can be attacked by fault injection attacks [33, 37]. Even the post-quantum cryptographic algorithms [39], which can protect against quantum computing, can be vulnerable to fault attacks. Therefore, it is necessary to have efficient FI attack protections that can be easily deployed.

On embedded processors, a fault model in which an attacker can skip an assembly instruction or equivalently replace it by a nop has been observed on several architectures and for several fault injection means [32]. Moro et al. in [31] assume that the effect of the injected fault on a 32-bit microcontroller leads to an instruction skip. Moro et al. [32] and Barenghi et al. [7] have proposed implementations of the *Instruction Redundancy* technique as a countermeasure against this fault model. Instruction skips correspond to specific cases of instruction replacements: replacing an instruction with another one that does not affect any useful register has the same effect as a nop replacement and so is equivalent to an instruction skip.

In our threat model, we consider an attacker with physical access to a device, capable of injecting faults such as voltage glitches during the computation of a critical function like the number theoretic transform. These faults can corrupt or skip instructions and happen anywhere multiple times but does not crash the device. Furthermore, the model permits the attacker to perform basic power side-channel analysis, collecting power trace samples. By correlating data-dependent power consumption with the Hamming weight leakage model, the attacker can expose vulnerabilities in cryptographic computations. This underscores the crucial need for robust defenses against both fault injection and side-channel attacks.

3 OVERVIEW OF THE COUNTERMEASURE

We use the notion of *random self-reducibility* [8, 44] to develop a new software-based countermeasure against fault-injection attacks and simple power side-channel attacks. Therefore, in this section, we provide the necessary background on random self-reducibility. Since we apply our countermeasure to number-theoretic operations, we also provide the necessary background on number theoretic transforms.

3.1 Random Self-Reducibility

Informally, a function f is random-self-reducible if the evaluation of f at any given instance x can be reduced in polynomial time to the evaluation of f at one or more random instances.

Definition 1 (Random Self-Reducibililty (RSR) [8, 44]). *Let $x \in \mathbb{D}$ and $c > 1$ be an integer. We say that f is c-random self-reducible if f can be computed at any particular input x via:*

$$F\left[f(x), f\left(a_1\right), \ldots, f\left(a_k\right), a_1, \ldots, a_k\right] = 0 \tag{1}$$

where F can be computed asymptotically faster than f and the a_i's are uniformly distributed, although not necessarily independent; e.g., given the value of a_1 it is not necessary that a_2 be randomly distributed in $\mathbb{D}$. This notion of random self-reducibility is somewhat different than other definitions given by [9, 19], where the requirement on F is that it be computable in polynomial time.

It is shown by Blum et al. [8] that self-correctors exist for any function that is *random self-reducible*. A *self-corrector* for f takes a program P that is correct on most inputs and turns it into a program that is correct on every input with high probability.

We have incorporated the concept of self-correctness to safeguard against fault-injection attacks, and the principles of random self-reducibility and randomly-testable functions to defend against power side-channel attacks. These notions are investigated and applied as a countermeasure against physical attacks in the literature.

3.2 RSR against Power Side Channels

At the heart of this method is the generic, randomized Algorithm 1, which is founded on the principle described in Definition 1. Additionally, Algorithm 2 boosts the effectiveness of the randomized Algorithm 1 through majority voting and probability amplification [48]. Consider a correct program P that has an associated random self-reducible property, which takes the form of a functional equation p. This property is deemed satisfied if, in the equation p, we can substitute P for the function f and the equation remains true.

Algorithm 1: c-secure-countermeasure PSCA (P, x, c).

Input : Program: P, Sensitive input: x, Security: c
Output: $P(x)$

```
Randomly split a_1, ..., a_c based on x.
for i = 1, ..., c do
    α_i ← P(a_i)
return F[x, a_1, ..., a_c, α_1, ..., α_c]
```

Generic c-secure-countermeasure PSCA (P, x, c) defined Algorithm 1 takes a program P, a sensitive input x, and a security parameter c. The algorithm randomly splits x into c shares $a_1, \ldots, a_c$ such that $x = a_1 + \cdots + a_c$, and calls P on each share a_i to obtain $\alpha_i = P(a_i)$. Finally, the algorithm returns the result of the function F on $x, a_1, \ldots, a_c, \alpha_1, \ldots, \alpha_c$. The function basis F is defined based on the random self-reducible property of the function f that P implements (cf. Definition 1).

To ensure minimum security, splitting the secret input into two shares would suffice. However, for enhanced security, the secret input can be divided into additional shares. It's important to view the security parameter c as an invocation to P, especially in the context of bivariate functions, rather than merely the number of shares.

Masking with Random Self-Reducibility. If a cryptographic operation has a random self-reducible property, then it is possible to protect it against power side-channel attacks by masking with arithmetic secret sharing.

3.3 Self-Correctness against Fault Injections

Fault injection attacks rely on obtaining a faulty output or correlating the faulty output with the input or secret-dependent intermediate values. By introducing redundancy and majority voting, we can obtain correct results even if some results are incorrect due to injected faults.

In Algorithm 2, we show how to apply the fault injection countermeasure approach on top of the power side-channel countermeasure. To protect a program P that implements a function f having a random self-reducible property, the algorithm calls P's c-secure-countermeasure n times and returns the majority of the answers. The function c-secure-countermeasure takes a program P, a sensitive input x, and a security parameter c.

Algorithm 2: n-secure countermeasure FIA (P, x, n, c).

Input : Program: P, Sensitive input: x, Security: n, c
Output: $P(x)$
1 **for** $m = 1, \ldots, n$ **do**
2 answer$_m \leftarrow$ call c-secure-countermeasure(P, x, c)
3 **return** the majority in $\{\text{answer}_m\colon m = 1, \ldots, n\}$

Note that c and n are independent security parameters. The security parameter c represents the number of calls to the unprotected program used in the PSCA countermeasure, whereas n signifies the number of iterations in the FIA countermeasure. The security parameter n is associated with the attacker's capability to inject effective faults. Owing to redundancy, an increase in the security parameter c results in a decreased likelihood of the attacker successfully injecting a fault.

Algorithm 3: (c, n)-secure mod operation (P, R, x, c, n).

Input : Program: P, Sensitive input: x, Security: n, c
Output: $P(x)$
1 **for** $m = 1, \ldots, n$ **do**
2 $x_1, x_2, \ldots, x_c \leftarrow\$$ Random-Split$(R2^n, x)$
3 answer$_m \leftarrow P(x_1, R) +_R P(x_2, R) \ldots +_R P(x_c, R)$
4 **return** the majority in $\{\text{answer}_m\colon m = 1, \ldots, n\}$

Algorithm 3 presents an example of a combined and configurable countermeasure, effective against both PSCA and FIA, applied to the modular multiplication operation. In Line 2, the algorithm divides the input x into c shares $x_1, x_2, \ldots, x_c$, satisfying $x = x_1 + x_2 + \cdots + x_c$.

Self-Correctness with Majority Voting. Fault injection attacks rely on faulty output. By majority voting, we can obtain correct results even if some results are incorrect.

3.4 n and attacker's probability of success

Fault injection occurs at the hardware level and is both challenging and unpredictable to control. When a successful fault is induced, it transforms a previously correct victim program into an incorrect one. Consequently, the essence of a fault injection attack is its probabilistic nature. This concept is abstracted in terms of the attacker's probability of success in our work.

Definition 2 (ε-fault tolerance). *Let ε be the upper bound on the attacker's probability of injecting a fault successfully at an unprotected program P that correctly implements a function f. Say that the program P is ε-fault tolerant for the function f provided $P(x) = f(x)$ for at least $1 - \varepsilon$ of any input x. We assume each fault injection is independent of the others:* $\Pr_{fault}[P(x) \neq f(x)] < \varepsilon$.

Algorithm 1 is a randomized algorithm and Algorithm 2 is also a randomized algorithm that repeats the computation n times by calling Algorithm 1 and uses majority voting to pick the correct answer. Therefore, we can use Chernoff bounds [48] to show that the probability of getting the correct answer is at least $1 - \delta$.

A simple and common use of Chernoff bounds is for "boosting" of randomized algorithms. If one has an algorithm that outputs a guess that is the desired answer with probability $p > 1/2$, then one can get a higher success rate by running the algorithm $n = \log(1/\delta)2p/(p - 1/2)^2$ times and outputting a guess that is output by more than $n/2$ runs of the algorithm. Assuming that these algorithm runs are independent, the probability that more than $n/2$ of the guesses is correct is equal to the probability that the sum of independent Bernoulli random variables X_k that are 1 with probability p is more than $n/2$. This can be shown to be at least $1 - \delta$ via the multiplicative Chernoff bound ($\mu = np$) [15]: $\Pr[X > n/2] \geq 1 - e^{-n(p-1/2)^2/(2p)} \geq 1 - \delta$.

Theorem 1 (Derived from Theorem 3.1 in [26]). *Suppose that f is randomly self-reducible and that P is ε-fault tolerant for the function f. Consider a c-secure countermeasure $\widetilde{C}(x)$ (Line 4 in Algorithm 1):*

$$\textbf{\textit{return}}\ F[x, a_1, \ldots, a_c, P(a_1), \ldots, P(a_c)]$$

Then, for any x, $\widetilde{C}(x)$ is equal to $f(x)$ with probability at least $1 - \varepsilon c$.

Proof. Fix an input x. Clearly, the probability that $\widetilde{C}(x)$ is correct is at least the probability that for each $i, P(a_i) = f(a_i)$. This follows since f is random self-reducible with respect to the number of calls to P is done. It therefore follows that $\widetilde{C}$ returns correct results at least $1 - \varepsilon c$ of the time. □

In the next sections, we will present a number of examples of c-secure countermeasures whose security parameter is mostly $c = 2$. Thus, for these functions, Theorem 1 says that, for ε equal to $1/100$, the probability that $\widetilde{C}$ returns correct results is at least 0.98. We can amplify the probability of success by repeating the computation n times and using majority voting. In addition, we can select a bigger n by adjusting δ as the confidence parameter:

Lower bound for n. The attacker's probability of success is ε, and for a c-secure countermeasure, the lower bound for n is defined as: $n = \log(1/\delta)2(1 - \varepsilon c)/(\varepsilon c/2)^2$, where δ is the confidence parameter.

Algorithm 1 makes calls to a program P that implements a function f having a random self-reducible property. However, we do not need to know the implementation of the function f, we just need to know the mathematical definition of the function f to configure the Algorithm 1 and 2. Therefore, one further advantage of our countermeasure is that it follows "black-box" approach. The fault injection attacks are hardware attacks, and the attacker does not have access to the software implementation of the function. Therefore, the attacker can only observe the input and output of the function. By using the black-box approach, we basically make the countermeasure robust at the hardware level.

Black-box. If we replace the f function with a program P that computes the function f, then our countermeasure $\widetilde{C}$ access P as a black-box and computes the function f using the random self-reducible properties of f.

4 END-TO-END IMPLEMENTATIONS

In this section, we introduce implementations of the RSA-CRT signature algorithm and Kyber's key generation algorithm, detailing existing vulnerabilities and how we can protect them against them using our methods.

4.1 Securing RSA-CRT Algorithm.

RSA is a cryptographic algorithm commonly used in digital signatures and SSL certificates. Due to the security of RSA, which relies on the difficulty of factoring the product of two large prime numbers, the calculation of RSA is relatively slow. Therefore, it is seldom used to encrypt the data directly.

For efficiency, many popular cryptographic libraries (e.g., OpenSSL) use RSA based on the Chinese remainder theorem (CRT) for encryption or signing messages. Algorithm 4 is the RSA-CRT signature generation algorithm. With the private key, we pre-calculate the values $d_p = d \mod (p-1)$, $d_q = d \mod (q-1)$ and $u = q^{-1} \mod p$, then generate the intermediate value $s_p = m^{d_p} \mod p$, $s_q = m^{d_q} \mod q$. Finally, combine two intermediate value s_p, s_q with the Garner's algorithm $S = s_q + (((s_p - s_q) \cdot u) \mod p) \cdot q$ The RSA based on CRT is about four times faster then classical RSA.

Algorithm 4: RSA-CRT Signature Generation Algorithm

Input: A message M to sign, the private key (p, q, d), with $p > q$, pre-calculated values $d_p = d \mod (p-1)$, $d_q = d \mod (q-1)$, and $u = q^{-1} \mod p$.
Output: A valid signature S for the message M.

1 $m \leftarrow$ Encode the message M in $m \in \mathbb{Z}_N$
2 $s_p \leftarrow m^{d_p} \mod p$ ▷ Protection with Alg. 5
3 $s_q \leftarrow m^{d_q} \mod q$ ▷ Protection with Alg. 5
4 $t \leftarrow s_p - s_q$
5 **if** $t < 0$ **then** $t \leftarrow t + p$
6 $S \leftarrow s_q + ((t \cdot u) \mod p) \cdot q$
7 **return** *S as a signature for the message M*

However, using CRT to improve RSA operation efficiency makes RSA vulnerable. For instance, in [4], Aumüller et al. provided the fault-based cryptanalysis method of RSA-CRT that the attacker can intentionally induce the fault during the computation, which changes s_p to faulty $\hat{s_p}$, to obtain the faulty output and factorize N by using the equation $q = gcd((s'^e - m) \mod N, N)$ to recover the secret key. Sung-Ming et al. provided another equation that can factorize N with faulty signature in [53]. There are two scenarios that the attacker can break the RSA-CRT. If the attacker knows the value of the message and faulty output, they can factorize N with the previous equation. On the other hand, if the attacker knows the value of correct and faulty signatures, they can factorize N with the equation $q = gcd((\hat{s} - s) \mod N, N)$.

We protect modular exponentiation at Line 2 and Line 3 of Algorithm 4 using the proposed countermeasure against the attack introduced in [4]. Its self-correcting program is very simple to code. The hardest operation to perform is the modular multiplication $P(a, x_1, R) \cdot_R P(a, x_2, R)$.

The self-correcting program can compute this multiplication directly without using random self-reducible property, however, for extra protection, 2-secure modular multiplication can be used (cf. Algorithm 6). Let's consider multiplication of integers mod R for a positive number R. In this case, $f(x, y, R) = x \cdot_R y$. Suppose that both x and y are in the range $\mathbb{Z}_{R2^n}$ for some positive integer n. Algorithm 6 shows a possible implementation for the protected modular multiplication with a c security parameter set to 2.

Algorithm 5: 2-secure mod. exponentiation (P, R, a, x)

1 $x_1, x_2 \leftarrow_\$$ Random-Split$(\phi(R)2^n, x)$
2 **return** $\leftarrow P(a, x_1, R) \cdot_R P(a, x_2, R)$ ▷ calls Alg. 6

Algorithm 6: 2-secure mod. multiplication (P, R, x, y)

1 $x_1, x_2 \leftarrow_\$$ Random-Split$(R \times 2^n, x)$
2 $y_1, y_2 \leftarrow_\$$ Random-Split$(R \times 2^n, y)$
3 **return** $P(x_1, y_1, R) +_R P(x_2, y_1, R) +_R P(x_1, y_2, R) + P(x_2, y_2, R)$

4.2 Securing Kyber Key Generation Algorithm.

The NIST standardization process for post-quantum cryptography [34] has finished its third round, and provided a list of new public key schemes for new standardization [2]. While implementation performance and theoretical security guarantees served as the main criteria in the initial rounds, resistance against side-channel attacks (SCA) and fault injection attacks (FIA) emerged as an important criterion in the final round, as also clearly stated by NIST at several instances [40].

Transforms used in signal processing such as the Fast Fourier Transform (FFT) or Number Theoritic Transform (NTT) or their inverse can be protected with our countermeasure. NTT over an n point sequence is performed using the well-known butterfly network, which operates over several layers/stages. The atomic operation within the NTT computation is denoted as the butterfly operation. A butterfly operation takes as inputs $(a, b) \in \mathbb{Z}_q^2$ and a twiddle constant w, and produces outputs $(c, d) \in \mathbb{Z}_q^2$.

Consider a transformation $T(x_1, \ldots, x_n)$ where the values x_i are fixed point numbers, i.e., 2-complement's arithmetic of some fixed size. This follows since the transformation is linear. Thus, $T(x_1, \ldots, x_n) = T(x_1 + \tilde{r}_1, \ldots, x_n + \tilde{r}_n) - T(\tilde{r}_1, \ldots, \tilde{r}_n)$.

Algorithm 7: 2-secure NTT $(P, x_1, \ldots, x_n \in \mathbb{Z}_q^2)$.

1 Choose $\tilde{r}_1, \ldots, \tilde{r}_n \in_\mathbb{U} \mathbb{Z}_q^2$
2 **return** NTT$(x_1 + \tilde{r}_1, \ldots, x_n + \tilde{r}_n)$ − NTT$(\tilde{r}_1, \ldots, \tilde{r}_n)$

The key point here is that since fixed-point values are a group under addition, the value $x_i + \tilde{r}_i$ is a uniform random value. The countermeasure for NTT is given in Algorithm 7.

They typically operate over polynomials in polynomial rings, and notably, polynomial multiplication is one of the most computationally intensive operations in practical implementations of these schemes. Among the several known techniques for polynomial multiplication such as the schoolbook multiplier, Toom-Cook [52] and

Algorithm 8: CPA Secure Kyber PKE (CPA.KeyGen)

1 $seed_A \leftarrow \text{Sample}_U()$
2 $seed_B \leftarrow \text{Sample}_U()$
3 $\hat{A} \leftarrow \text{NTT}(A)$
4 $s \leftarrow \text{Sample}_B(seed_B, coins_s)$
5 $e \leftarrow \text{Sample}_B(seed_B, coins_e)$
6 $\hat{s} \leftarrow \text{NTT}(s)$ ▷ Protection with Algorithm 7
7 $\hat{e} \leftarrow \text{NTT}(e)$
8 $\hat{t} \leftarrow \hat{A} \odot \hat{s} + \hat{e}$
9 **return** $pk = (seed_A, \hat{t}), sk = (\hat{s})$

Karatsuba [23], the Number Theoretic Transform (NTT) based polynomial multiplication [16] is one of the most widely adopted techniques, owing to its superior run-time complexity. Over the years, there has been a sustained effort by the cryptographic community to improve the performance of NTT for lattice-based schemes on a wide-range of hardware and software platforms [1, 13]. As a result, the use of NTT for polynomial multiplication yields the fastest implementation for several lattice-based schemes. In particular, the NTT serves as a critical computational kernel used in Kyber [6] and Dilithium [29], which were selected as the first candidates for PQC standardization [41].

A recent fault injection attack [41] that exposes a significant vulnerability in NTT-based polynomial multiplication, allowing the zeroization of all twiddle constants through a single targeted fault. This vulnerability enables practical key/message recovery attacks on Kyber KEM and forgery attacks on Dilithium. Moreover, the proposed attacks are also shown to bypass most known fault countermeasures for lattice-based KEMs and signature schemes.

To safeguard polynomial multiplication, we can protect individual NTT operations using Algorithm 7. In this paper, we focus on securing the NTT operation targeted by Ravi et al.[41] using Algorithm7. Consequently, we reinforce Line 6 of Algorithm 8 with our proposed countermeasure against the attack delineated in [41].

5 EVALUATION

We conducted three experimental sets to assess our countermeasure's effectiveness against fault injection and power side-channel attacks. Initially, we evaluated protected operations individually, including modular multiplication, modular exponentiation, and NTT. Subsequently, we assessed our countermeasure's robustness within RSA-CRT and Kyber key generation algorithms. Finally, we examined the latency overhead introduced by our countermeasure.

To capture power traces, for our experiments we use an ATSAM4S-based target board. SAM4S is a microcontroller based around the 32-bit ARM cortex-m4 processor core, which is commonly used in embedded systems such as IoT devices. The specific target board comes with the ChipWhisperer Husky [35], which is the equipment that we used for power trace collection.

The voltage fault injection test bed is created using Riscure's VC Glitcher product[1] that generates an arbitrary voltage signal with a pulse resolution of 2 nanoseconds. We use a General Purpose Input Output (GPIO) signal to time the attack which allows us to inject a glitch at the moment the target is executing the targeted code. The target's reset signal is used to reset the target prior to each experiment to avoid data cross-contamination. All fault injection experiments are performed targeting an off-the-shelf development platform built around an STM32F407 MCU, which includes an ARM Cortex-M4 core running at 168 MHz. This Cortex-M4 based MCU has an instruction cache, a data cache and a prefetch buffer.

In power side-channel evaluation, we use the Hamming Weight leakage model and the Test Vector Leakage Assessment (TVLA) [21] to evaluate the effectiveness of our countermeasure. The instantaneous power consumption measurement corresponding to a single execution of the target algorithm is referred to as power trace. Each power trace is therefore a vector of power samples, and the t-test has to be applied sample-wise. The obtained vector is referred to as t-trace.

To detect Points-of-Interest, we employ the Sum of Squared pairwise T-differences (SOST) [12] method, setting the threshold at 20% of the maximum. The t-test window size is uniformly set to ±8 for all operations. We define the power side-channel security parameter as $c = 2$ in the c-secure countermeasure in Algorithm 1 applicable to all operations. In the mod operation and modular multiplication, the entire operation is targeted, while in modular exponentiation and NTT, attacks are focused on the constant-time Montgomery ladder [27, 30] modular exponentiation function. For TVLA analysis, two sets of test vectors were created: one with random numbers of Hamming weight 12 and another with a Hamming weight of 4, using 1000 random numbers for each. These vectors were used for evaluating both protected and unprotected cryptographic operations.

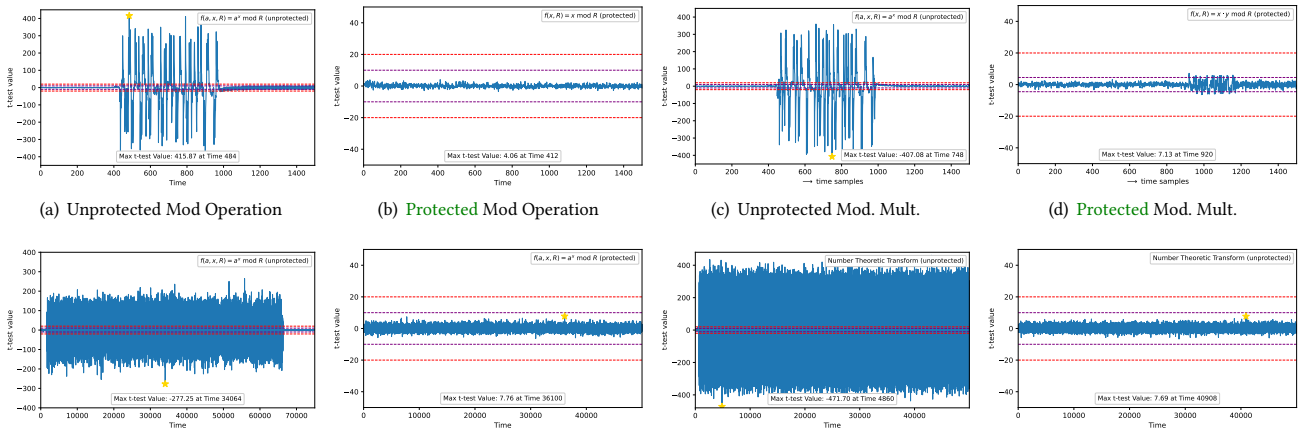
In our study, we also evaluated the distinguishability of total power consumption in modular operations and modular multiplication. For modular multiplication, we maintained one operand's value constant while varying the other operand among numbers with different Hamming Weights. This approach enables a comparative analysis of power consumption patterns in modular operations, particularly between unprotected and protected versions, offering insights into how variations in Hamming Weight influence power consumption in these protected cryptographic operations.

Our evaluation indicates that the RSR countermeasure significantly reduced t-test results, bringing them into acceptable regions. For example, in the mod operation, the maximum t-test result decreased from 415.7 to 4.12, and for NTT, it dropped from 417.7 to 7.69. These results, which are detailed in Table 1, demonstrate an average reduction of two orders of magnitude, highlighting the effectiveness of the RSR countermeasure in enhancing the security of cryptographic operations against side-channel attacks.

In the fault injection attack evaluation, we use the model of injecting faults to cause changes to the desired output, comparing the desired output to the one of the fault. We set the fault injection security parameter as $n = 10$ for n-secure countermeasure 2 for all operations.

Figure 2 presents the results of our fault attack experiment. We employed voltage glitches for the fault injection attacks. The Glitch Offset is the time between when the trigger is observed and when the glitch is injected. The Glitch Length is the time for which the Glitch Voltage is set. Glitch Offset and Glitch Length are the two parameters that we varied to inject faults, they correspond to the start

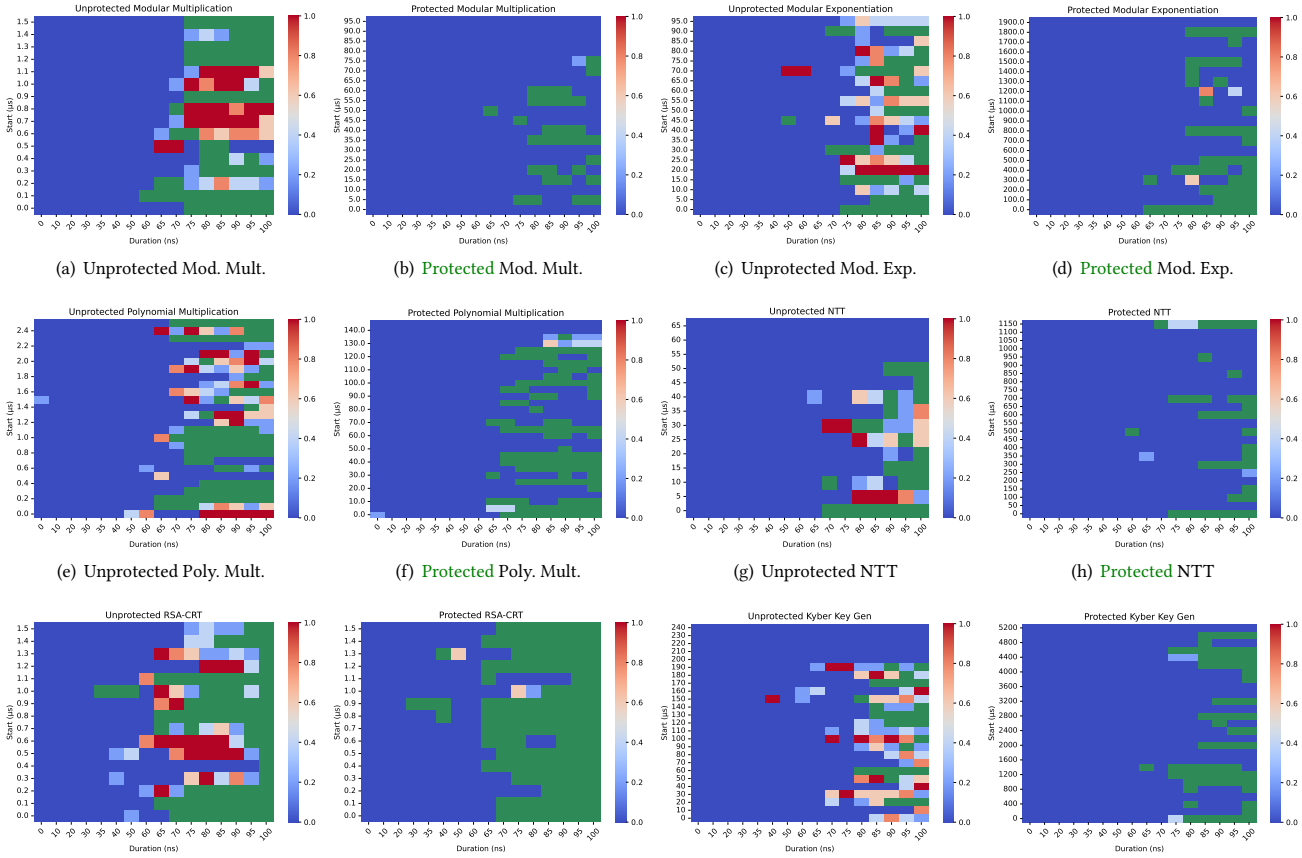
[1] https://www.riscure.com/products/vc-glitcher/



(a) Unprotected Mod Operation (b) Protected Mod Operation (c) Unprotected Mod. Mult. (d) Protected Mod. Mult.

(e) Unprotected Mod. Exp. (f) Protected Mod. Exp. (g) Unprotected NTT (h) Protected NTT

Figure 1: Power Side-Channel Attack Evaluation t-tests



(a) Unprotected Mod. Mult. (b) Protected Mod. Mult. (c) Unprotected Mod. Exp. (d) Protected Mod. Exp.

(e) Unprotected Poly. Mult. (f) Protected Poly. Mult. (g) Unprotected NTT (h) Protected NTT

(i) Unprotected RSA-CRT (j) Protected RSA-CRT (k) Unprotected Kyber Key Gen. (l) Protected Kyber Key Gen.

Figure 2: Fault Injection Attack Evaluation Heatmaps

Table 1: Reduction in Faults for Different Operations

Operation	Unprotected	Protected	Reduction
Mod. exponentiation	165	9	94.55%
Mod. multiplication	168	1	99.4%
NTT	63	5	92.06%
Poly. multiplication	196	14	92.86%
RSA-CRT	168	7	95.83%
Kyber Key. Gen.	172	4	97.67%

time and duration of the glitch, respectively. For each combination of start time and duration, we executed each target function five times, resulting in a total of 1280 test data points for each function. We used heatmaps to illustrate the ratio of faulty to correct outputs. This experiment yielded three types of outputs: faulty, correct, and board reset. In the heatmaps, colors closer to red indicate a higher likelihood of voltage glitches causing faulty outputs (red = 100%), whereas blue signifies a lower likelihood (blue = 0%). Green indicates instances where all test outputs resulted in the board being reset. We treated any output that is not the correct output as a fault, this is very conservative, as some of the outputs may not be effective faults. From these heatmaps, the unprotected functions exhibit a significantly higher number of red dots, indicating more faults. Furthermore, Table 1 demonstrates the reduction of faults in target functions, with our protection method reducing approximately 95.4% of faulty outputs in average, up to 99.4% in modular multiplication. Collectively, these results affirm the effectiveness of our protection method in safeguarding the functions.

We observed fault injection sometimes breaks memory allocations (malloc) without causing the target to crash. This is due to the fact that the target is not designed to handle such faults. We believe that this is a potential avenue for future work, as it may lead to new types of attacks. We simply reset the target in such cases, as we are not interested in the results of these attacks. However, in some cases, the fault progresses to the next operation silently without causing a crash and the target continues to operate. We registered these cases as successful attacks in the heatmaps.

We additionally protected the fault-injection countermeasure method (Algorithm 2) using classical techniques. After exiting the loop, the code verifies loop completed successfully. If not, the code resets the target. This is a simple and effective way to protect the countermeasure from fault injection attacks. This led to a reduction in faults 4.56% in average.

6 LIMITATIONS

Our study presents a novel software-based countermeasure against physical attacks such as power side-channel and fault-injection attacks, utilizing the concept of random self-reducibility and instance hiding for number theoretic operations. While our approach offers significant advantages over traditional methods, there are several inherent limitations. Firstly, the countermeasure's effectiveness is intrinsically linked to the random self-reducibility of the function being protected. This dependency means that our approach may not be universally applicable to all cryptographic operations. Secondly, redundancy and randomness inevitably introduce computational overhead. Nevertheless, each call to original function P can be easily parallelized in hardware or vectorized software implementations. This parallelization can potentially increase the noise, and we identify this as an avenue for future work. Finally, our approach is not tailored to defend against attacks targeting the random number generator itself. Nevertheless, there are also simple duplication based techniques to protect random number generators from physical attacks. For instance, one such technique involves comparing two successive random numbers to determine if they are identical or not, as discussed in the work of Ravi et al. [38].

7 CONCLUSION

In this work, we show that if a cryptographic operation has a random self-reducible property, then it is possible to protect it against physical attacks such as power side-channel and fault-injection attacks with a configurable security. We have demonstrated the effectiveness of our method through empirical evaluation across critical cryptographic operations including modular exponentiation, modular multiplication, polynomial multiplication, and number theoretic transforms (NTT). Moreover, we have successfully showcased end-to-end implementations of our method within two public key cryptosystems: the RSA-CRT signature algorithm and the Kyber Key Generation, to show the practicality and effectiveness of our approach. The countermeasure reduced the power side-channel leakage by two orders of magnitude, to an acceptably secure level in TVLA analysis. For fault injection, the countermeasure reduces the number of faults to 95.4% in average. Although the countermeasures were introduced as software-based, they can be more efficiently implemented in hardware, particularly on FPGAs. Each call to P can be parallelized in hardware, potentially increasing the noise. We identify this as an avenue for future work.

ACKNOWLEDGMENTS

At Virginia Tech, this project is partially supported by 4-VA, a collaborative partnership for advancing the Commonwealth of Virginia, Commonwealth Cybersecurity Initiative, and by the National Science Foundation (NSF) under grant CCF-2153748. At Yale University, this project is partially supported by NSF under grants CNS-2245344, CCF-2106845, CCF-2131476, CCF-2219995, and CCF-2318974.

REFERENCES

[1] Amin Abdulrahman, Jiun-Peng Chen, Yu-Jia Chen, Vincent Hwang, Matthias J Kannwischer, and Bo-Yin Yang. 2021. Multi-moduli NTTs for saber on Cortex-M3 and Cortex-M4. *Cryptology ePrint Archive* (2021).

[2] Gorjan Alagic, Daniel Apon, David Cooper, Quynh Dang, Thinh Dang, John Kelsey, Jacob Lichtinger, Carl Miller, Dustin Moody, Rene Peralta, et al. 2022. Status report on the third round of the NIST post-quantum cryptography standardization process. *US Department of Commerce, NIST* (2022).

[3] Amund Askeland and Sondre Rønjom. 2021. A Side-Channel Assisted Attack on NTRU. Cryptology ePrint Archive, Report 2021/790. https://ia.cr/2021/790.

[4] Christian Aumüller, Peter Bier, Wieland Fischer, Peter Hofreiter, and J-P Seifert. 2003. Fault attacks on RSA with CRT: Concrete results and practical countermeasures. In *Cryptographic Hardware and Embedded Systems (CHES)*. Springer, 260–275.

[5] Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. 2017. *CRYSTALS-Kyber algorithm specifications and supporting documentation.* Technical Report. NIST PQC Round.

[6] Roberto Avanzi, Joppe W. Bos, Leo Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, John Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. 2020. *CRYSTALS-Kyber (version 3.0): Algorithm specifications and supporting documentation.* Technical Report. Submission to the NIST post-quantum project. October 1, 2020.

[7] Alessandro Barenghi, Luca Breveglieri, Israel Koren, and David Naccache. 2012. Fault injection attacks on cryptographic devices: Theory, practice, and countermeasures. *Proc. IEEE* 100, 11 (2012), 3056–3076.

[8] Manuel Blum, Michael Luby, and Ronitt Rubinfeld. 1990. Self-testing/correcting with applications to numerical problems. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*. 73–83.

[9] Manuel Blum and Silvio Micali. 2019. How to generate cryptographically strong sequences of pseudo random bits. In *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*. 227–240.

[10] Claudio Bozzato, Riccardo Focardi, and Francesco Palmarini. 2019. Shaping the glitch: optimizing voltage fault injection attacks. *IACR transactions on cryptographic hardware and embedded systems* (2019), 199–224.

[11] Eric Brier, Christophe Clavier, and Francis Olivier. 2004. Correlation power analysis with a leakage model. In *International workshop on cryptographic hardware and embedded systems*. Springer, 16–29.

[12] Suresh Chari, Josyula R Rao, and Pankaj Rohatgi. 2002. Template attacks. In *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 13–28.

[13] Chi-Ming Marvin Chung, Vincent Hwang, Matthias J Kannwischer, Gregor Seiler, Cheng-Jhih Shih, and Bo-Yin Yang. 2021. NTT multiplication for NTT-unfriendly rings: New speed records for Saber and NTRU on Cortex-M4 and AVX2. *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2021), 159–188.

[14] Lucian Cojocar, Kostas Papagiannopoulos, and Niek Timmers. 2018. Instruction duplication: Leaky and not too fault-tolerant!. In *Smart Card Research and Advanced Applications: 16th International Conference, CARDIS 2017, Lugano, Switzerland, November 13–15, 2017, Revised Selected Papers*. Springer, 160–179.

[15] Wikipedia contributors. 2023. Chernoff bound — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Chernoff_bound#Applications [Online; accessed 7-September-2023].

[16] James W Cooley and John W Tukey. 1965. An algorithm for the machine calculation of complex Fourier series. *Mathematics of computation* 19, 90 (1965), 297–301.

[17] Léo Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. 2018. Crystals-dilithium: A lattice-based digital signature scheme. *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2018), 238–268.

[18] Taher ElGamal. 1985. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory* 31, 4 (1985), 469–472.

[19] Joan Feigenbaum and Lance Fortnow. 1993. Random-self-reducibility of complete sets. *SIAM J. Comput.* 22, 5 (1993), 994–1005.

[20] Aymeric Genêt, Natacha Linard de Guertechin, and Novak Kaluđerović. 2021. Full key recovery side-channel attack against ephemeral SIKE on the Cortex-M4. Cryptology ePrint Archive, Report 2021/858. https://ia.cr/2021/858.

[21] Benjamin Jun Gilbert Goodwill, Josh Jaffe, Pankaj Rohatgi, et al. 2011. A testing methodology for side-channel resistance validation. In *NIST non-invasive attack testing workshop*, Vol. 7. 115–136.

[22] Emre Karabulut, Erdem Alkim, and Aydin Aysu. 2021. Single-Trace Side-Channel Attacks on ω-Small Polynomial Sampling: With Applications to NTRU, NTRU Prime, and CRYSTALS-DILITHIUM. In *2021 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 35–45. https://ia.cr/2022/494.

[23] Anatolii Karatsuba. 1963. Multiplication of multidigit numbers on automata. In *Soviet physics doklady*, Vol. 7. 595–596.

[24] Paul Kocher, Joshua Jaffe, Benjamin Jun, and Pankaj Rohatgi. 2011. Introduction to differential power analysis. *Journal of Cryptographic Engineering* 1 (2011), 5–27.

[25] Moritz Lipp, Andreas Kogler, David Oswald, Michael Schwarz, Catherine Easdon, Claudio Canella, and Daniel Gruss. 2021. PLATYPUS: Software-based power side-channel attacks on x86. In *IEEE Symposium on Security and Privacy (SP)*.

[26] Richard Lipton. 1991. New directions in testing. *Distributed computing and cryptography* 2 (1991), 191–202.

[27] Zhe Liu, Johann Großschädl, and Ilya Kizhvatov. 2010. Efficient and side-channel resistant RSA implementation for 8-bit AVR microcontrollers. In *Workshop on the Security of the Internet of Things-SOCIOT*, Vol. 10.

[28] Victor Lomné, Thomas Roche, and Adrian Thillard. 2012. On the need of randomness in fault attack countermeasures-application to AES. In *2012 Workshop on Fault Diagnosis and Tolerance in Cryptography*. IEEE, 85–94.

[29] Vadim Lyubashevsky, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Peter Schwabe, Gregor Seiler, Damien Stehlé, and Shi Bai. 2017. Crystals-Dilithium. Submission to the NIST Post-Quantum Cryptography Standardization.

[30] Peter L Montgomery. 1985. Modular multiplication without trial division. *Mathematics of computation* 44, 170 (1985), 519–521.

[31] Nicolas Moro, Karine Heydemann, Amine Dehbaoui, Bruno Robisson, and Emmanuelle Encrenaz. 2014. Experimental evaluation of two software countermeasures against fault attacks. In *Hardware-Oriented Security and Trust (HOST)*. 112–117.

[32] Nicolas Moro, Karine Heydemann, Emmanuelle Encrenaz, and Bruno Robisson. 2014. Formal verification of a software countermeasure against instruction skip attacks. *Journal of Cryptographic Engineering* 4 (2014), 145–156.

[33] Koksal Mus, Yarkın Doröz, M Caner Tol, Kristi Rahman, and Berk Sunar. 2023. Jolt: Recovering tls signing keys via rowhammer faults. In *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1719–1736.

[34] NIST. 2016. Submission Requirements and Evaluation Criteria for the Post-Quantum Cryptography Standardization Process.

[35] Colin O'Flynn and Zhizhang (David) Chen. 2014. ChipWhisperer: An Open-Source Platform for Hardware Embedded Security Research. Cryptology ePrint Archive, Report 2014/204. https://ia.cr/2014/204.

[36] Conor Patrick, Bilgiday Yuce, Nahid Farhady Ghalaty, and Patrick Schaumont. 2017. Lightweight fault attack resistance in software using intra-instruction redundancy. In *Selected Areas in Cryptography–SAC 2016: 23rd International Conference*. Springer, 231–244.

[37] Gilles Piret and Jean-Jacques Quisquater. 2003. A differential fault attack technique against SPN structures, with application to the AES and KHAZAD. In *Cryptographic Hardware and Embedded Systems-CHES 2003: 5th International Workshop, Cologne, Germany, September 8–10, 2003. Proceedings 5*. Springer, 77–88.

[38] Prasanna Ravi, Anupam Chattopadhyay, Jan Pieter D'Anvers, and Anubhab Baksi. 2022. Side-channel and fault-injection attacks over lattice-based post-quantum schemes (Kyber, Dilithium): Survey and new results. *ACM Transactions on Embedded Computing Systems* (2022).

[39] Prasanna Ravi, Mahabir Prasad Jhanwar, James Howe, Anupam Chattopadhyay, and Shivam Bhasin. 2019. Exploiting determinism in lattice-based signatures: practical fault attacks on pqm4 implementations of NIST candidates. In *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*. 427–440.

[40] Prasanna Ravi and Sujoy Sinha Roy. 2021. Side-channel analysis of lattice-based PQC candidates. Round 3 Seminars, NIST Post Quantum Cryptography.

[41] Prasanna Ravi, Bolin Yang, Shivam Bhasin, Fan Zhang, and Anupam Chattopadhyay. 2023. Fiddling the Twiddle Constants-Fault Injection Analysis of the Number Theoretic Transform. *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2023), 447–481.

[42] Oscar Reparaz, Benedikt Gierlichs, and Ingrid Verbauwhede. 2017. Fast Leakage Assessment. Cryptology ePrint Archive, Report 2017/624. https://ia.cr/2017/624.

[43] Ronald L Rivest, Adi Shamir, and Leonard Adleman. 1978. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM* 21, 2 (1978), 120–126.

[44] Ronitt Rubinfeld. 1994. *Robust functional equations with applications to self-testing/correcting*. Technical Report. Cornell University.

[45] Tobias Schneider and Amir Moradi. 2015. Leakage Assessment Methodology - a clear roadmap for side-channel evaluations. Cryptology ePrint Archive, Report 2015/207. https://ia.cr/2015/207.

[46] Bodo Selmke, Johann Heyszl, and Georg Sigl. 2016. Attack on a DFA protected AES by simultaneous laser fault injections. In *2016 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. IEEE, 36–46.

[47] Bo-Yeon Sim, Aesun Park, and Dong-Guk Han. 2021. Chosen-ciphertext Clustering Attack on CRYSTALS-KYBER using the Side-channel Leakage of Barrett Reduction. Cryptology ePrint Archive, Report 2021/874. https://ia.cr/2021/874.

[48] Alistair Sinclair. 2011. Class notes for the course "Randomness and Computation". http://www.cs.berkeley.edu/~sinclair/cs271/n13.pdf.

[49] Petr Socha, Vojtěch Miškovskỳ, and Martin Novotnỳ. 2022. A Comprehensive Survey on the Non-Invasive Passive Side-Channel Analysis. *Sensors* 22, 21 (2022), 8096.

[50] Raphael Spreitzer, Veelasha Moonsamy, Thomas Korak, and Stefan Mangard. 2017. Systematic classification of side-channel attacks: A case study for mobile devices. *IEEE communications surveys & tutorials* 20, 1 (2017), 465–488.

[51] Hauke Malte Steffen, Lucie Johanna Kogelheide, and Timo Bartkewitz. 2021. In-depth Analysis of Side-Channel Countermeasures for CRYSTALS-Kyber Message Encoding on ARM Cortex-M4. Cryptology ePrint Archive, Report 2021/1307. https://ia.cr/2021/1307.

[52] Andrei L Toom. 1963. The complexity of a scheme of functional elements simulating the multiplication of integers. In *Doklady Akademii Nauk*, Vol. 150. Russian Academy of Sciences, 496–498.

[53] Sung-Ming Yen, Sangjae Moon, and Jae-Cheol Ha. 2003. Hardware fault attack on RSA with CRT revisited. In *Information Security and Cryptology—ICISC 2002: 5th International Conference Seoul, Korea, November 28–29, 2002 Revised Papers 5*. Springer, 374–388.