Power Side-Channel Key Recovery Attack On a Hardware Implementation of BIKE

Luke Beckwith[1,2], Huizhen Zhou[1], Jens-Peter Kaps[1], Kris Gaj[1]
[1]*George Mason University*, Fairfax, VA, 22030, USA
[2]*PQSecure Technologies*, Boca Raton, FL, 33431, USA
{lbeckwit, hzhou9, jkaps, kgaj}@gmu.edu

***Index Terms*—Side-channel Analysis; Post-Quantum-Cryptography; Correlation Power Analysis**

***Abstract*—BIKE is a code-based Key Encapsulation Mechanism (KEM) currently under consideration for standardization by the National Institute of Standards and Technology (NIST). BIKE, along with several other candidates, is being evaluated in the fourth round of the NIST Post-Quantum Cryptography (PQC) competition. In comparison to the lattice-based candidates, relatively little effort has been focused on analyzing this algorithm for side-channel vulnerabilities, especially in hardware. There have been several works on side-channel attacks and countermeasures on software implementations of BIKE, but as of yet, there have been no works focused on hardware. This work presents the first side-channel attack on a hardware implementation of BIKE. The attack targets a public implementation of the algorithm and is able to fully recover the long-term secret key with only several dozen traces. This work reveals BIKE's significant susceptibilities to side-channel attacks when implemented in hardware and the need for investigation of hardware countermeasures.**

I. Introduction

Current cryptographic algorithms are based on mathematical problems that are difficult to solve on classical computers but are trivial to solve using a quantum computer. Thus, our current cybersecurity infrastructure can be broken by a sufficiently large quantum computer. Fortunately, over the past eight years, NIST has coordinated focused research to identify new cryptographic algorithms that are secure against classical and quantum computing attacks. While these algorithms are secure against computational attacks, their implementations are still vulnerable to side-channel attacks. These attacks seek to recover sensitive information through physical characteristics of the implementation, such as latency or power consumption.

Cryptographic engineers have been at work identifying sensitive operations and devising methods of protection against these attacks. Much of this effort has been focused on the popular lattice-based family of algorithms. The newly selected standards, ML-KEM (based on CRYSTALS-Kyber) [1] and ML-DSA (based on CRYSTALS-Dilithium) [2], have been well analyzed for side-channel weaknesses and methods of protection. ML-KEM has received both protected software and protected hardware implementations [3]–[7]. One public work has been published for a software-protected implementation of ML-DSA [8].

NIST has made it clear that they also intend to standardize other families of algorithms. In particular, three code-based Key Encapsulation Mechanisms (KEM) are being evaluated for future standardization: Classic McEliece, BIKE, and HQC. Of these algorithms, both BIKE and HQC have orders of magnitude smaller public keys than Classic McEliece, making them more practical candidates for standardization. Of the two, HQC has slightly better performance, but BIKE has keys and ciphertexts that are roughly half the size of those for HQC. Thus, BIKE is a very competitive algorithm for standardization.

These algorithms have received significantly less effort in terms of protection and attacks. Several attacks have been performed on HQC, but only in software, and no protected implementation has been developed [9]–[11]. For BIKE, a previous work performed a single-trace power analysis attack on a software implementation [12]. Another work performed a side-channel attack on a software implementation of QcBits, which is in the same family of algorithms as BIKE [13]. In terms of protected implementations, one work presented a fully protected software implementation [14], and two works presented protected gadgets related to the sampling and inversion of polynomials [15], [16].

Contribution: This work presents the first effort on side-channel attacks on hardware implementations of BIKE. The target of the attack is a publicly available implementation of BIKE [17]. The sequential design of the polynomial multiplier enables a large number of attack points to be extracted from each power trace. On average, the attack requires two dozen traces and does not rely on any strong assumptions such as chosen ciphertext or the ability to create templates for the target device. The CPA attack code is available online[1]. Due to the large size of the trace data, only a subset of the power traces are included. These results show the need to investigate side-channel countermeasures for hardware implementations of BIKE.

II. Background

A. BIKE Algorithm

BIKE, described in [18], is a code-based KEM constructed using Quasi-Cyclic Moderate Density Parity Check (QC-MDPC) codes. These codes define their generator and parity

979-8-3503-6806-2/24/$31.00 ©2024 IEEE

[1]https://cryptography.gmu.edu/artifacts/AsianHOST24/BIKE-SCA/

check matrices using a single vector of length r with binary coefficients. Each row of the matrix is defined by cyclically shifting the vector by one position. Thus, a matrix-vector multiplication is equivalent to polynomial multiplication in the ring $\mathbb{Z}_2[x]/(x^r + 1)$.

TABLE I: BIKE ALGORITHM PARAMETERS.

Security Level	r	ω	t
1	12,323	142	134
3	24,659	206	199
5	40,973	274	264

The secret key of BIKE consists of two sparse polynomials h_0, h_1 of degree r, each having exactly $\omega/2$ non-zero entries. Together, these polynomials define the parity check matrix. The public key is calculated from the secret key by $h = h_1 * h_0^{-1}$. During encapsulation, the message is used as the seed for the SHA3 XOF function from which two sparse polynomials e_0, e_1 with combined weight t are sampled. These values are used to calculate the first component of the ciphertext, $c_0 = e_0 + e_1 * h$. The two vectors are hashed and added to the message, which is included as the second ciphertext component, c_1. During decapsulation, the secret key decoding algorithm is applied to c_0, which allows recovery of e_0 and e_1. These values are then hashed and added to c_1 to recover the message.

There are two notable features of BIKE that are relevant to the attack presented in this paper. First, recovery of either h_0 or h_1 is sufficient for full recovery of the secret key. If h_0 is known, then we can calculate $h \times h_0 = (h_1 \times h_0^{-1}) \times h_0 = h_1$. If h_1 is known, we can calculate $h^{-1} \times h_1 = (h_1^{-1} \times h_0) \times h_1 = h_0$. Second, the structure of the cyclic code means that rotations of the secret key polynomials h_0, h_1 generate equivalent codes. The structure of the h_0 component of the parity check matrix is as follows:

$$H_0 = \begin{pmatrix} rotate(h_0, 0) \\ rotate(h_0, 1) \\ \vdots \\ rotate(h_0, r-1) \end{pmatrix}$$

Multiplication of h_0 by a polynomial x^k modulo $x^r + 1$ is equivalent bitwise to rotation by k. Thus, the polynomial $h_0' = h_0 \times x^k$ generates the following matrix:

$$H_0' = \begin{pmatrix} rotate(h_0, k \ mod \ r) \\ rotate(h_0, k+1 \ mod \ r) \\ \vdots \\ rotate(h_0, k+(r-1) \ mod \ r) \end{pmatrix}$$

The matrix H_0' is equivalent to H_0 by row swapping, so it generates the same code. Thus, two polynomials $h_0' = h_0 \times x^k, h_1' = h_1 \times x^k$ can be used to decapsulate a ciphertext encoded with public key $h = h_1 * h_0^{-1}$.

B. Power Side-Channel Attacks

Side-channel attacks are a category of implementation attacks that exploit information leaked from the physical implementation of a cryptographic algorithm rather than weaknesses in the cryptographic algorithms themselves. These attacks may target characteristics including operation latency, power consumption, or electromagnetic emissions. This work focuses on power side-channel attacks, which seek to recover sensitive information by analyzing the power consumption of the target device during the cryptographic operation. Two prominent types of side-channel attacks are Simple Power Analysis (SPA) and Differential Power Analysis (DPA). SPA attacks attempt to recover sensitive information by analyzing a single trace; DPA attacks analyze statistical differences between a larger number of traces to learn information about the secret value.

In this work, we perform a Correlation Power Analysis (CPA) attack, which is a type of DPA attack. To perform a CPA attack, the attacker chooses a target operation of the form $c = f(a, s)$ where a is a known value, s is the target secret value, and c is a signal that has a significant impact on the power consumption of the device (i.e., a value written to a register in an FPGA). The attacker enumerates all possible values of s, referred to as key guesses, and calculates the corresponding hypothetical values of c. The attack then uses the hypothetical values of c to generate a power model based on a leakage model, such as the Hamming weight or Hamming distance. In the case of FPGAs, the power consumption is generally proportional to the number of bit flips that occur. Thus, the Hamming distance model is the most appropriate leakage model. The assumption is that the calculated power model of the correct key guess will have the strongest correlation with the actual power consumption of the device. To determine the correlation between the power model for all key guesses and the power consumption of the device, Pearson's correlation coefficient between each key guess and each point of the power trace is calculated.

C. BIKE Multiplier Architecture Hardware

In this work, we target a publicly available hardware implementation of BIKE [17]. The hardware architecture implements all three security levels of the algorithm and has a configurable datapath width. We provide an overview of the most relevant module, the polynomial multiplier, but refer the reader to the implementers' publication for further details.

The block diagram of the multiplier is shown in Figure 1. The implementation takes advantage of the fact that all multiplications in BIKE are performed between one dense polynomial, with an arbitrary number of non-zero entries, and a sparse polynomial, with a small, fixed number of non-zero entries. Instead of using traditional schoolbook multiplication, the multiplier uses a combination of shift and XOR operations. Since the field of the coefficients is $GF(2)$ and the polynomial modulus is $x^r + 1$, the multiplication of a dense polynomial by a single value x^i is equivalent to rotation of the coefficients by i positions. That is:

$$(a_0 + a_1 x^1 + \cdots + a_{r-1} x^{r-1}) * (x^i) = \\ (a_{(r-i) mod \ r} + a_{(r-i+1) \ mod \ r} x + \cdots + a_{(r-i+r-1) \ mod \ r} x^{r-1})$$

Any polynomial multiplication can be viewed as a series of $a(x) * x^i$ operations that are summed together. For each non-zero coefficient of the sparse input, the dense input is rotated

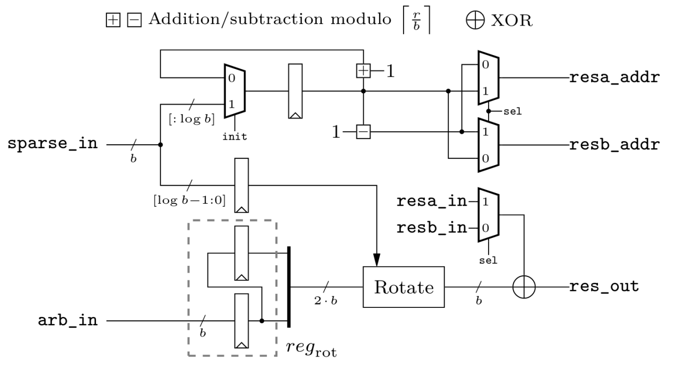


Fig. 1: Block diagram of BIKE hardware polynomial multiplier. Figure taken from [17].

based on the non-zero coefficient's index and accumulated into the resulting state. In Figure 1, `sparse_in` is the sparse coefficient index input which is used to determine how many bits to rotate the dense input, `arb_in`. Accumulation of the result state is performed by XORing the previous state, read from the memory interface `resa` or `resb`, with the shifted input before writing back to memory through the `res_out` signal. The rotation is split into two steps: the bit-wise shift and the word-wise shift. The bit-wise shift is determined by the lower bits of `sparse_in` and is performed using the `Rotate` module. The word-wise shift is determined by the upper bits of `sparse_in` and is performed by adjusting the memory address offset when reading and writing the result to memory. For each sparse index input, $\lceil r/b \rceil$ cycles of shift and XOR are performed. Thus the latency of this module is proportional to $\lceil r/b \rceil * \frac{\omega}{2}$, where $\frac{\omega}{2}$ is the weight of the sparse input.

III. Methodology

A. Target Operation and Attack Method

In this section, we describe the approach used for our CPA key recovery attack on the hardware implementation of BIKE. Our goal in this attack is full recovery of the long-term secret key composed of (h_0, h_1). Recovery of the secret key enables decryption of the ciphertext and thus decryption of any information encrypted under that shared secret key. Decapsulation is the target of our attack as it is the only operation that repeatedly uses the same secret key. The target operation is the multiplication of the known ciphertext value c_0 with the secret polynomial h_0 at the beginning of the decapsulation operation. In most phases of the attack, the target signal is the memory output data signal, `res{a,b}_din`, which has the stronger effect on power consumption than the memory input signal based on our experiments. The memory input can also be targeted, but it has less effect on power consumption and is thus more susceptible to noise.

Since the result of one round of the multiplier is impacted by the previous rounds, we must attack the coefficients in sequence. Recovery of $h_0[i]$ is dependent on the correct recovery of all previous coefficients. The attack is described in its three phases: (1) Recovery of the first coefficient $h_0[0]$, (2) recovery of $h_0^w[1]$, the most significant bits (MSBs) of $h_0[1]$ that determine the word-shift of that round, (3) and the remaining rounds of the attack that target $h_0^w[i]$ and $h_0^b[i-1]$, where $h_0^b[i-1]$ represents the least significant bits (LSBs) of $h_0[i-1]$ which determine the bit-shift.

Recovery of $h_0[0]$: The first step of the attack is to recover the first coefficient, $h_0[0]$. This stage of the attack relies solely on the memory write of the rotated ciphertext polynomial. The word-shift is primarily applied by adjusting the read and write address signals of the result memory. The ciphertext polynomial, received through the `arb_in` signal, is always read starting at index 0. The bit-shift is applied by the `rotate` module before the write back to memory. The memory write of the rotated data is the target for this stage of the attack.

The bit-shift, determined by $h_0^b[0]$, is straightforward to attack since every word of the shifted data is affected. Even though the access order of the ciphertext polynomial is the same for all key values, the word-shift, determined by $h_0^w[0]$, can still be detected. As shown in Figure 2, the value of $h_0^w[0]$ determines where the wrapping caused by the polynomial reduction occurs. If a separate power model is created for each intermediate result of the multiplication, the key guesses with the correct bit-shift and word-shift will be strongly correlated with all intermediate results, while those with only the correct bit-shift will be strongly correlated with only a subset of the results.

Even though there is a distinction between all keys, neighboring keys will only have a single-bit difference in their power model. Because of this, recovery of $h_0^w[0]$ is the most challenging step of the attack. Fortunately, two factors make it easier: (1) there are no parallel operations adding noise to the measurement, primarily due to the data dependencies of the decapsulation algorithm, and (2) exact recovery is not required. As discussed previously, $h_0 \times x^i$ produces an equivalent code to h_0. Thus, if the entire key is offset by a constant value, it is equivalent to the original key. If the recovered $h_0^w[0]$ is close to the correct value, the attack can proceed. All the following coefficients will simply be offset by $h_0^\Delta = h_0[0] - h_0'[0]$.

There are cases where a non-zero h_0^Δ value can initially prevent recovery of a coefficient. If the value of h_0^Δ causes the target coefficient $h_0[i]$ to wrap around the modulus (i.e. $h_0[i] + h_0^\Delta > r$ or $h_0[i] + h_0^\Delta < 0$), then the location of the wrapping caused by polynomial reduction will be significantly offset. This can cause a mismatch in most or all of the power model calculations. However, this can still be addressed. If the attack fails at a coefficient where $h_0^w[i]$ is close to r, the attacker can conclude that this boundary condition has occurred and attempt a brute-force correction by subtracting or adding b from all previously recovered coefficients and attempting the attack again. This process can be repeated multiple times until the coefficient is successfully recovered or the attacker decides to halt the attack. The approach was applied in our script, and we were able to continue the attack successfully when this edge case occurred.

Recovery of $h_0^w[1]$: In the next phase of the attack, we aim to recover the MSBs of the second coefficient, $h_0^w[1]$, from the memory read of the second round of multiplication. Since $h_0[0]$ has already been recovered, the values stored in the result memory are known. The MSBs of $h_0[1]$ determine

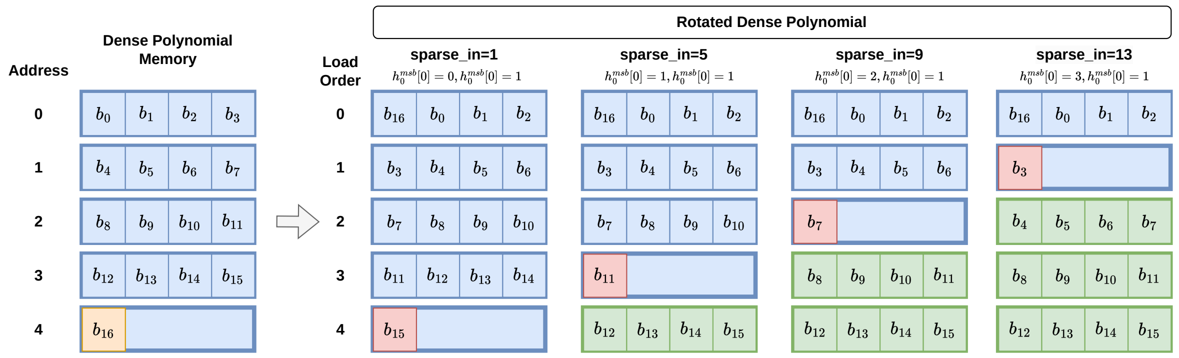

Address
Dense Polynomial Memory
Load Order
Rotated Dense Polynomial
sparse_in=1
$h_0^{msb}[0]=0, h_0^{msb}[0]=1$
sparse_in=5
$h_0^{msb}[0]=1, h_0^{msb}[0]=1$
sparse_in=9
$h_0^{msb}[0]=2, h_0^{msb}[0]=1$
sparse_in=13
$h_0^{msb}[0]=3, h_0^{msb}[0]=1$

Fig. 2: Example of input data for various rotation values for simplified parameter $r = 17$. Partial blocks are colored red, blocks before the rotation wrap are blue, and blocks after the rotation wrap are green.

the read offset of the result memory, which determines the value on target the `res{a,b}_din` signal. Thus, we create our power model for the jth intermediate results using the following equation, where `res_arr` represents the memory array of b-bit words representing the current result state.

$$\texttt{res\{a,b\}\_din = res\_arr}[h_0^w[1]+j]$$

Recovery of $h_0^w[i]$ and $h_0^b[i-1]$: Recovery of the remaining coefficients targets the memory read of the ith round and follows a single structure. This stage of the attack assumes the $h_0[0], ..., h_0[i-2]$ and $h_0^w[i-1]$ are known. The goal is to recover $h_0^b[i-1]$ and $h_0^w[i]$. The known information is used to create the power model based on the following equations. First, we determine the theoretical value of the rotated ciphertext in the previous round using the known word-shift and the current key guesses for the bit shift. The `arb_arr` variable represents the memory array of b-bit words representing the ciphertext polynomial.

$$(1)\ \texttt{arb\_in\_rot} = \texttt{rotate(arb\_arr},h_0^w[i-1] \times b + h_0^b[i-1])$$

Then, we calculate the theoretical value of the result memory using the known word-shift and the previously calculated ciphertext rotation guess. The value of `res_arr[i-1]` can be calculated using $h_0[0], ..., h_0[i-2]$. This represents the data stored in the result memory after the previous rounds' calculation.

$$(2)\ \texttt{res\_arr}_i\texttt{[j]} = \texttt{res\_arr}_{i-1}[h_0^w[i-1]+j] \oplus \texttt{arb\_in\_rot}[j*b+:b]$$

Finally, we calculate the theoretical value of the memory output signal during the jth intermediate operation based on the word-shift key guess.

$$(3)\ \texttt{res\{a,b\}\_din = res\_arr}_i[h_0^w[i]+j]$$

In summary, the value of the data in the result memory is determined by previously recovered coefficients and the unknown bit-shift value $h_0^b[i-1]$, while the read offset that is observed on `res{a,b}_din` is determined by $h_0^w[i]$. Thus, we can target $h_0^b[i-1]$ and $h_0^w[i]$ for recovery, which has a search space of a single coefficient, $\log_2(r)$ bits.

This round of the attack is repeated for all coefficients. The final coefficient is only partially recovered since in the final round of multiplication; we target $h_0^b[\frac{\omega}{2}-2]$ and $h_0^w[\frac{\omega}{2}-1]$. However, the remaining $log_2(b)$ bits can be trivially brute-forced.

B. Impact of Security Level

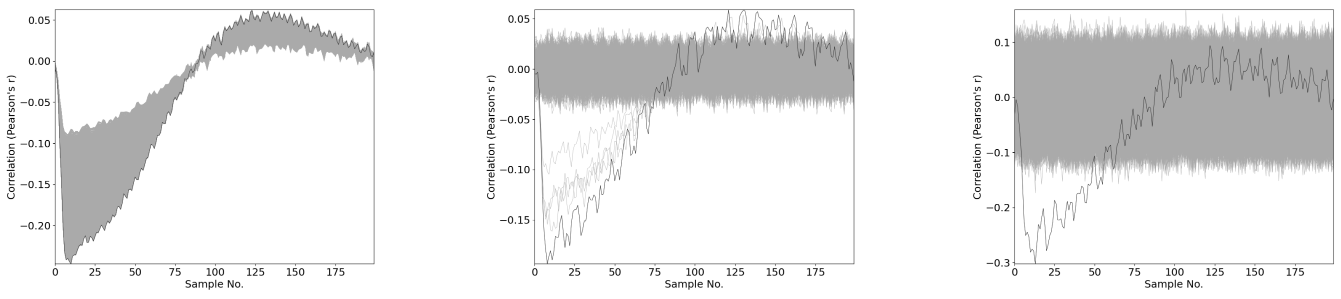
The relevant parameters, listed in Table I, are the polynomial degree r, which determines the search space of each coefficient, and ω which defines the number of non-zero coefficients of h_0 and thus the number of coefficients that need to be recovered. As previously stated, the first coefficient requires significantly more traces to recover than the following coefficients. Thus, while larger values of ω increase the processing time of the attack, they do not increase the number of traces required. Intuitively, it seems that increasing the value of r would proportionally increase the number of traces required. However, for the second and third steps of the attack, increasing r also increases the number of intermediate multiplication values that can be targeted in each trace. So, the increase in the search space is proportional to the increase in information in each trace. For the first round, key guesses that are far from the correct key have a larger number of mismatched intermediate values, as demonstrated in Figure 2. The primary challenge is distinguishing between key guesses that neighbor the correct key. Therefore, increasing the security level has minimal impact on the difficulty of the attack.

C. Measurement Setup

The power trace measurements were performed using a FOBOS workbench [19]. The measurements were taken using a Picoscope 3203D oscilloscope. This oscilloscope has an 8-bit ADC and was configured to sample at 1 Gigasample/s. The target board was the CW305 Artix-7 FPGA board which was run with a clock of 5 MHz.

IV. Results

The attack described in the previous sections was applied to the hardware implementation presented in [17]. We were able to successfully recover the full h_0 vector, which is sufficient for full recovery of the secret key. No strong assumptions were



(a) Correlation graph for attack step 0 targeting $h_0[0]$

(b) Correlation graph for attack step 1 targeting $h_0^w[1]$

(c) Correlation graph for attack step 2 targeting $h_0^b[1]$ and $h_0^w[2]$

Fig. 3: Selected Correlation Graphs. Each trace represents the correlation of one key guess. The correct key's trace is highlighted.

TABLE II: Maximum of MTD for the 10 attacks performed at each security level.

Security Level	**Attack Max MTD**										
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**Average**
1	28	28	12	20	16	24	12	2	4	24	17
3	30	29	25	2	28	34	27	4	34	23	24
5	21	34	24	24	26	33	24	3	24	19	24

made. The ciphertext input values were known but not chosen, and no templates were used. The attack requires a very low number of traces, as shown in Table II. The Measurement to Disclosure (MTD) value signifies the number of traces needed before the correct key was recovered by the CPA attack. In this case, the largest MTD of all coefficients in the key is recorded. The low number of traces is only possible because there are a significant number of intermediate values that can be targeted for each trace. Our attack takes advantage of the $\lceil r/b \rceil$ operations per trace that are dependent on the same secret coefficient. This corresponds to 381, 771, and 1,281 attack points per trace for security levels 1, 3, and 5, respectively. Thus, the average number of attack points was 6,477 for security level 1, 18,504 for security level 3, and 30,744 for security level 5.

Figures 3a, 3b, and 3c show the correlation for the power model of each key guess at each sample of the power trace. Correct key guesses should produce power model values that are positively or negatively correlated with the actual power consumption of the device. Thus, the chosen key is most likely to be the key that corresponds to the power model value with the largest absolute correlation value.

Figure 3a shows the correlation results for the first coefficient of h_0. The target value is simply the bit-shifted ciphertext polynomial. Thus, neighboring key guesses produce values one bit-shift apart, meaning the largest possible difference in the Hamming distance between neighboring key guesses is one. This is why the key guesses have very similar shapes with only slight differences in magnitude. This coefficient requires the most traces of the attack, though the correct key still distinguishes itself in relatively few traces. In some cases, the exact value of the first coefficient was not recovered. However, the attack was still completed successfully by applying the techniques discussed in the previous section. Of the 30 experiments performed, the exact value of $h_0[0]$ was correctly recovered in 21. Of the 9 experiments where it was not exactly recovered, 6 were only offset by 32, and the largest offset was only 256. Brute-force key correction was only required in one case, and only one attempt was required. We were able to complete all attacks successfully.

Figure 3b shows the correlation results for the recovery of $h_0^w[1]$. Recovery of the MSBs of the second coefficient is significantly easier than recovery of the first coefficient. The closest neighboring key guesses have somewhat similar power models, but the distinction is not as close as for the first coefficient.

Figure 3c shows the correlation for the recovery of $h_0^b[1]$ and $h_0^w[2]$. All following rounds of the attack have similar results. In contrast to the first coefficient, the remaining coefficients provide very distinct differences in correlation. Since the previous state affects the target value, neighboring key guesses can produce results with any number of different bits, which makes the attack easier than for the first coefficient.

V. Countermeasures

In order to achieve provable security against this attack and all forms of DPA attacks, the primary approach is to apply masking. There are two works on the protection of specific suboperations, one implementing polynomial inversion [16] and one implementing polynomial sampling [15]. The fully protected software implementations presented in [14] applied boolean shares to mask the entire algorithm. Their result suggests the dense polynomial format is more efficient for protected implementations, which would require using a traditional polynomial multiplier as was used in [20]. This adds significant area and performance costs.

As an alternative to these conservative countermeasures, we propose two lightweight countermeasures. These countermeasures do not provide complete security but will add significant noise to the attacker's measurements, increasing the number of traces required for key recovery.

Secret Key Shuffling: Shuffling is a well-known lightweight countermeasure. It reduces DPA attack effectiveness since the target operation will not always occur on the expected clock cycle. In the case of sparse multiplication in BIKE, shuffling is extremely inexpensive as it can be applied directly to the key. The array representing the indices of the non-zero coefficients can be randomly shuffled before each decapsulation operation.

Secret Key Shifting: As observed previously, a random shift of the h_0 and h_1 polynomials will define an equivalent code. Our attack exploited this fact to reduce the accuracy needed when recovering the first coefficient of h_0. However, we can also take advantage of this as a countermeasure. Before each decapsulation operation, the polynomials h_0 and h_1 can be multiplied by x^k for some random k value. This random shifting of the polynomial serves as an additional source of noise.

These lightweight countermeasures could provide some level of protection with little performance and area overhead. The exact effectiveness and cost of these countermeasures is an interesting topic for future work.

VI. Conclusions

This work has presented a successful key recovery attack on a hardware implementation of the BIKE algorithm. The attack applied was a Correlation Power Analysis (CPA) attack with a minimal attack model that does not require ciphertext manipulation or templates. The attack is able to exploit the sequential operation of the polynomial multiplier to extract multiple attack points from a single power trace. Due to this optimization, only a couple dozen traces are required to fully recover the secret key at any security level. This reveals the need for the development of efficient and effective countermeasures for BIKE implementations.

References

[1] NIST, *Module-Lattice-Based Key-Encapsulation Mechanism Standard*, FIPS 203, Aug. 2024.

[2] NIST, *Module-Lattice-Based Digital Signature Standard*, FIPS 204, Aug. 2024.

[3] L. Beckwith, A. Abdulgadir, and R. Azarderakhsh, "A Flexible Shared Hardware Accelerator for NIST-Recommended Algorithms CRYSTALS-Kyber and CRYSTALS-Dilithium with SCA Protection," in *Topics in Cryptology – CT-RSA*, 2023, pp. 469–490.

[4] T. Kamucheka, A. Nelson, D. Andrews, and M. Huang, "A Masked Pure-Hardware Implementation of Kyber Cryptographic Algorithm," in *2022 International Conference on Field-Programmable Technology (ICFPT).*

[5] J. W. Bos, M. Gourjon, J. Renes, T. Schneider, and C. v. Vredendaal, "Masking Kyber: First-and Higher-Order Implementations," *IACR Transactions on Cryptographic Hardware and Embedded Systems, 2021(4)*, 2024.

[6] A. Jati, N. Gupta, A. Chattopadhyay, and S. K. Sanadhya, "A Configurable CRYSTALS-Kyber Hardware Implementation with Side-Channel Protection," *ACM Transactions on Embedded Computing Systems*, vol. 23,

[7] D. Heinz, M. J. Kannwischer, G. Land, P. Schwabe, and A. Sprenkels, "First-Order Masked Kyber on ARM Cortex-M4," *Cryptology ePrint Archive No. 58*, 2022.

[8] M. Azouaoui, O. Bronchain, G. Cassiers, *et al.*, "Protecting Dilithium against Leakage: Revisited Sensitivity Analysis and Improved Implementations," *IACR Transactions on Cryptographic Hardware and Embedded Systems, 2023(4)*, pp. 58–79, 2023.

[9] G. Goy, J. Maillard, P. Gaborit, and A. Loiseau, "Single trace HQC shared key recovery with SASCA," *IACR Transactions on Cryptographic Hardware and Embedded Systems, 2024(2)*, pp. 64–87, 2024.

[10] G. Goy, A. Loiseau, and P. Gaborit, "A New Key Recovery Side-Channel Attack on HQC with Chosen Ciphertext," in *Post-Quantum Cryptography*, 2022.

[11] T. Schamberger, L. Holzbaur, J. Renner, A. Wachter-Zeh, and G. Sigl, "A Power Side-Channel Attack on the Reed-Muller Reed-Solomon Version of the HQC Cryptosystem," in *Post-Quantum Cryptography*, 2022.

[12] A. Cheriere, N. Aragon, T. Richmond, and B. Gérard, "BIKE Key-Recovery: Combining Power Consumption Analysis and Information-Set Decoding," in *Applied Cryptography and Network Security*, May 2023, pp. 725–748.

[13] B.-Y. Sim, J. Kwon, K. Y. Choi, J. Cho, A. Park, and D.-G. Han, "Novel Side-Channel Attacks on Quasi-Cyclic Code-Based Cryptography," *IACR Transactions on Cryptographic Hardware and Embedded Systems, 2019(4)*, pp. 180–212,

[14] L. Demange and M. Rossi, "A provably masked implementation of BIKE key encapsulation mechanism," *IACR Communications in Cryptology*, vol. 1, no. 1, Apr. 9, 2024.

[15] M. Krausz, G. Land, J. Richter-Brockmann, and T. Güneysu, "A holistic approach towards side-channel secure fixed-weight polynomial sampling," in *Public-Key Cryptography – PKC 2023*, May 2023, pp. 94–124.

[16] M. Krausz, G. Land, J. Richter-Brockmann, and T. Güneysu, "Efficiently masking polynomial inversion at arbitrary order," in *Post-Quantum Cryptography*, Sep. 2022, pp. 309–326.

[17] J. Richter-Brockmann, M.-S. Chen, S. Ghosh, and T. Güneysu, "Racing BIKE: Improved Polynomial Multiplication and Inversion in Hardware," *IACR Transactions on Cryptographic Hardware and Embedded Systems, 2022(1)*, pp. 557–588,

[18] N. Aragon, P. S. L. M. Barreto, S. Bettaieb, *et al.*, "BIKE: Bit Flipping Key Encapsulation," [Online]. Available: https://bikesuite.org/.

[19] E. Ferrufino, L. Beckwith, A. Abdulgadir, and J.-P. Kaps, "FOBOS 3: An Open-Source Platform for Side-Channel Analysis and Benchmarking," in *2023 Workshop on Attacks and Solutions in Hardware Security*, Copenhagen Denmark: ACM, Nov. 2023, pp. 5–14.

[20] J. Richter-Brockmann, J. Mono, and T. Guneysu, "Folding BIKE: Scalable Hardware Implementation for Reconfigurable Devices," *IEEE Transactions on Computers*, vol. 71, no. 5, May 2022.