IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN XXXX-XXXX, Vol. 0, No. 0, pp. 1–23. DOI:XXXXXXXX

Adaptive Template Attacks on the Kyber Binomial Sampler

Eric Chun-Yu Peng and Markus G. Kuhn

University of Cambridge, Cambridge, UK, {cyp24,mgk25}@cl.cam.ac.uk

Abstract. Template attacks build a Gaussian multivariate model of the side-channel leakage signal generated by each value of a targeted intermediate variable. Combined with additional steps, such as dimensionality reduction, such models can help to infer a value with nearly 100% accuracy from just a single attack trace. We demonstrate this here by reconstructing the output of the binomial sampler of a Cortex-M4 implementation of the Kyber768 post-quantum key-encapsulation mechanism. However, this performance is usually significantly diminished if the device, or even just the address space, used for profiling differs from the attacked one. Here we introduce a new technique for adapting templates generated from profiling devices in order to attack another device where we are also able to record many traces, but without knowledge of the random value held by the targeted variable. We interpret the model from the profiling devices as a Gaussian mixture and use the Expectation–Maximization (EM) algorithm to adapt its means and covariances to better match the unlabelled leakage distribution observed from the attacked setting. The Kyber binomial sampler turned out to be a particularly suitable target, for two reasons. Firstly, it generates a long sequence of values drawn from a small set, limiting the number of Gaussian components that need to be adjusted. Secondly, the length of this sequence requires particularly well-adapted templates to achieve a high key-recovery success rate from a single trace. We also introduce an extended point-of-interest selection method to improve linear discriminant analysis (LDA).

Keywords: ML-KEM · CRYSTALS-Kyber · Binomial Sampler · Single-Trace Attack · Adaptive Template Attack · Point of Interest Extension · EM Algorithm

1 Introduction

CRYSTALS-Kyber is the first Post-Quantum Cryptography (PQC) standard for key encapsulation, having been adopted in Federal Information Processing Standard (FIPS) 203 as the Module-Lattice-Based Key-Encapsulation Mechanism (ML-KEM) [Nat24]. A recent survey by Ravi *et al.* [RCDB24] reviews published physical attacks on Kyber, categorized into whether the KEM protocol is used with static (longer term) or ephemeral (single use) key pairs. The latter can create a false sense of security, as attackers would have to succeed in extracting a secret from observing a single execution of the algorithm. This may encourage developers to deploy ephemeral-key Kyber-KEM without countermeasures. However, single-trace attacks on real devices have been demonstrated against several Kyber components, including the NTT [PPM17, PP19], Keccak [KPP20, YK20], and message encoding [SKL+20, XPR+22] and decoding [RBRC22] modules. This leaves the binomial sampler, which generates nonces in Kyber, to be explored as a side-channel target: the only attack on that listed in [RCDB24] is a fault-injection-induced nonce-reuse attack [RRB+19]. Here, we evaluate the potential of single-trace template attacks on Kyber's binomial sampler. In particular, we demonstrate the practicality of such attacks with a new technique for porting templates across different devices and operations.

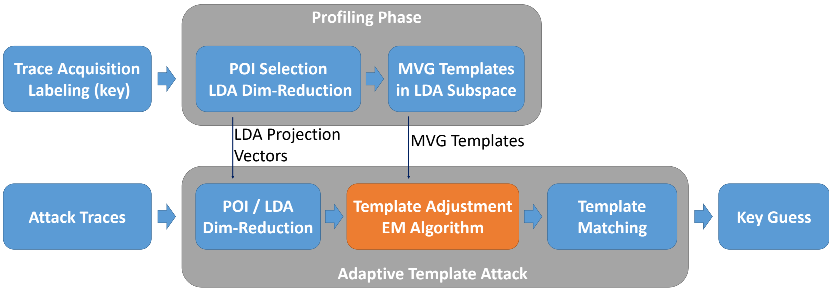


Figure 1: The adaptive template attack introduces an EM-based adjustment phase to improve cross-device template accuracy.

The binomial sampler module is one of the most security-sensitive components in Kyber, because it is the sole provider of the secret "noise" polynomials. It also appears in all three operations of the KEM: `KeyGen`, `Encaps`, and `Decaps`. Due to the large storage size of Kyber's key pairs, NIST allows storing the random seed used to generate a key pair, for re-generating the secret key whenever needed [Nat24]. This storage-space optimization further exposes the binomial sampler as an attack surface. Besides the secret-key sampling operation, this module also generates other secret random polynomials in Kyber's `Encaps` and `Decaps` operations. The exposure of those nonces could easily be used to recover the encapsulated shared secret from the ciphertext.

But if a nonce is generated only once, this requires the attacker to recover it from a single trace. Kyber's three parameter sets denote the number $c \in \{512, 768, 1024\}$ of coefficients in a secret key (or nonce). In this paper, we use the term *accuracy* (and blue shading of table cells) for the probability of correctly recovering a single number, e.g. one coefficient of a secret polynomial, and the term *success rate* (and red shading) for the probability of correctly reconstructing the entire secret, consisting of c polynomial coefficients. Recovering all these c coefficients with an expected success rate of SR requires that the coefficient accuracy a is very close to 100%, such that $a^c \geq \text{SR}$:

	SR = 50 %	SR = 90 %	SR = 99 %
Kyber512	99.86 %	99.979 %	99.9980 %
Kyber768	99.91 %	99.986 %	99.9986 %
Kyber1024	99.93 %	99.990 %	99.9999 %

With such high accuracy requirements, the practicality of such attacks may at first seem questionable, especially if the attacked device differs from the profiling devices used for building the model. This is essentially a template portability problem, which requires methods to maintain accuracy across devices. Previous work suggests either to apply data normalization [EG12, MBTL13] or to employ multi-device training [CK18, BCH+20] to improve the template portability and generalization. Profiling techniques based on machine learning (ML) and deep learning (DL) have also been proposed to improve model portability. Cao *et al.* [CZLG21] introduce a DL-based Cross-Device Profiled Attack technique, which adds a fine-tuning step, to let their neural-network classifier learn the leakage model of the attacked device. In this work, we also explore model adjustments, to enhance the portability of our Gaussian multivariate templates.

We introduce here the *adaptive template attack*, to improve cross-device accuracy with an additional template-adjustment phase that modifies the outcome of the profiling phase to better match the leakage of the attacked device. We propose this as a new way of improving template portability by combining supervised learning (template profiling) with an unsupervised learning technique (template adjustment). As illustrated in Figure 1, our processing pipeline starts conventionally, with point selection and LDA-based dimensionality reduction, to build the initial templates with profiling devices (Section 2.1). We then reuse the same dimensionality-reducing LDA projection in the attack phase and follow it with the Expectation–Maximization (EM) algorithm (Section 2.4) for template adjustment (Section 5.1). The EM algorithm is initialized with the template parameters from the profiling device and then fed with unlabelled traces from the attacked device. This should tailor the resulting model to better fit the leakage of the attacked device, resulting in higher template accuracy.

Contributions

- **First passive power-analysis attack on Kyber's binomial sampler.** We identify a Kyber implementation vulnerability that affects all three operations of Kyber-KEM (Section 2.3 and 4.2).
- **Adaptive template attack.** We show how to improve the attack's portability using a variant of the EM algorithm (Section 5.1), resulting in successful single-trace attacks on different targets (Section 5.3).
- **Address-space portability.** We adapt profiled templates not only to attack different devices, but also to attack the same subroutine when the address location of the processed data differs, e.g. because the call stack or memory allocation in the profiled and attacked setting are not the same (Section 2.3 and Table 3).
- **Point-of-interest extension (POIe) for LDA.** We also propose a new paradigm for selecting points of interest for improving LDA templates (Section 4.3). Since LDA exploits information about *both* the signal and noise, we include some nearby *low*-SNR samples to help the LDA better identify correlated noise (e.g., low-frequency waveforms) and project away from it.

2 Background

2.1 LDA-based Template Attack

Template attacks [CRR02] infer sensitive data directly from side-channel traces, which is essential for single-trace attacks. They typically use multivariate Gaussian (MVG) models as templates [MOP07], with the traces being treated as vectors in a high-dimensional space. The traces of different intermediate values are modelled as different MVG distributions. There are two phases in a template attack: a profiling phase, which builds the templates from traces labelled with known intermediate values, and an attack phase, which infers intermediate values from unlabelled traces.

In the profiling phase, the attacker collects traces from a profiling device and builds for each intermediate value $k \in K$ a template consisting of a mean vector $\boldsymbol{\mu}_k$ and a covariance matrix $\boldsymbol{\Sigma}_k$. In the attack phase, the attacker infers the processed sensitive data through template matching. Given a side-channel trace $\boldsymbol{t}$ and a multivariate Gaussian template $(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ with d dimensions, a basic matching process can estimate the likelihood of value k with the probability density function

$$p(\boldsymbol{t} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = \frac{1}{\sqrt{(2\pi)^d |\boldsymbol{\Sigma}_k|}} \exp\left(-\frac{1}{2}(\boldsymbol{t}-\boldsymbol{\mu}_k)^\top {\boldsymbol{\Sigma}_k}^{-1}(\boldsymbol{t}-\boldsymbol{\mu}_k)\right)$$

needed for the posterior probability

$$\mathrm{P}(k \mid \boldsymbol{t}) = \frac{p(\boldsymbol{t} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)\,\mathrm{P}(k)}{\sum_{k'} p(\boldsymbol{t} \mid \boldsymbol{\mu}_{k'}, \boldsymbol{\Sigma}_{k'})\,\mathrm{P}(k')}.$$

To decrease the computational complexity of the aforementioned likelihood estimation, we can apply two dimensionality reduction techniques. Points-of-interest (POI) selection simply eliminates from traces $\boldsymbol{t}$ samples where some signal-to-noise measure (e.g. NICV [BDGN14]) falls below a threshold. Linear discriminant analysis (LDA) [SA08] is useful not just to compress the trace length but also to boost the template accuracy. LDA projects data from a higher-dimensional space into a lower-dimensional subspace spanned by base vectors $\boldsymbol{w}$ while maximizing the inter-class and intra-class variance ratio

$$J(\boldsymbol{w}) = \frac{\boldsymbol{w}^\top \boldsymbol{B}\boldsymbol{w}}{\boldsymbol{w}^\top \boldsymbol{W}\boldsymbol{w}}$$

where $\boldsymbol{B}$ and $\boldsymbol{W}$ denote the inter-class and intra-class scatter matrices, respectively. The LDA projection vectors can be efficiently computed with the following steps [CK13]. First, compute

$$\boldsymbol{B} = \sum_{k\in K} N_k(\overline{\boldsymbol{t}_k} - \overline{\boldsymbol{t}})(\overline{\boldsymbol{t}_k} - \overline{\boldsymbol{t}})^\top$$
$$\boldsymbol{W} = \sum_{k\in K}\sum_{i=1}^{N_k}(\boldsymbol{t}_{k,i} - \overline{\boldsymbol{t}_k})(\boldsymbol{t}_{k,i} - \overline{\boldsymbol{t}_k})^\top$$

where N_k denotes the number of traces $\boldsymbol{t}_{k,i}$ available for value label k, and $\overline{\boldsymbol{t}_k} = N_k^{-1}\sum_{i=1}^{N_k}\boldsymbol{t}_{k,i}$ is their mean trace. Then, apply eigendecomposition on $\boldsymbol{W}^{-1}\boldsymbol{B}$ to find the eigenvalues and eigenvectors. The eigenvectors $\boldsymbol{w}$ with the d largest eigenvalues then form the LDA projection vectors.

2.2 CRYSTALS-Kyber and Binomial Sampler

CRYSTALS-Kyber (or Kyber) uses a variant of the Fujisaki-Okamoto (FO) transform [FO99] to turn the Kyber public key encryption (PKE) scheme into an IND-CCA2-secure KEM. Simply put, it re-encrypts and performs a ciphertext-integrity check during decapsulation. Kyber768 is the NIST-recommended default setting with a "sound" security margin against possible quantum-algorithm breakthroughs [Nat24]. The three main algorithms of Kyber-KEM (KEM.`KeyGen`, `Encaps`, `Decaps`) are built on top of the Kyber-PKE scheme (PKE.`KeyGen`, `Encrypt`, `Decrypt`), simplified versions of which are shown below. For a more detailed description of Kyber, see FIPS 203 [Nat24] or the original specification [ABD+21].

Algorithm 1 KEM.`KeyGen`

Output: (pk, sk)
1: $d, z \xleftarrow{\$} \mathbb{B}^{32}$
2: $(pk, sk') = \text{PKE.}\mathtt{KeyGen}(d)$
3: $sk = (sk', pk, \mathtt{H}(pk), z)$

Algorithm 2 KEM.`Encaps`

Input: pk
Output: (K, ct)
1: $m \xleftarrow{\$} \mathbb{B}^{32}$
2: $(K, r) = \mathtt{G}\big(m \,\|\, \mathtt{H}(pk)\big)$
3: $ct = \text{PKE.}\mathtt{Encrypt}(pk, m, r)$

Algorithm 3 KEM.`Decaps`

Input: (sk, ct)
Output: K'
1: $(sk', pk, h, z) = sk$
2: $m' = \text{PKE.}\mathtt{Decrypt}(sk', ct)$
3: $(K', r') = \mathtt{G}(m' \,\|\, h)$
4: $ct' = \text{PKE.}\mathtt{Encrypt}(pk, m', r')$
5: $\bar{K} = \mathtt{J}(z \,\|\, ct)$
6: **if** $ct \neq ct'$ **then**
7: $K' = \bar{K}$

Algorithm 4 PKE.`KeyGen`

Input: *seed*
Output: (pk, sk')
1: $\mathbf{A} \xleftarrow{\$} R_q^{k\times k}$
2: $\mathbf{s}, \mathbf{e} \xleftarrow{\$} \mathcal{B}_\eta(seed) \in R_q^k$
3: $\mathbf{t} = \mathbf{A} \circ \mathbf{s} + \mathbf{e}$
4: $pk = (\mathbf{t}, \mathbf{A})$
5: $sk' = \mathbf{s}$

Algorithm 5 PKE.`Encrypt`

Input: (pk, m, r)
Output: ct
1: $\mathbf{r}, \mathbf{e}_1 \xleftarrow{\$} \mathcal{B}_\eta(r) \in R_q^k$
2: $e_2 \xleftarrow{\$} \mathcal{B}_\eta \in R_q$
3: $\mathbf{u} = \mathbf{A}^\top \circ \mathbf{r} + \mathbf{e}_1$
4: $v = \mathbf{t}^\top \circ \mathbf{r} + e_2 + \mathtt{Encode}(m)$
5: $ct = (\mathbf{u}, v)$

Algorithm 6 PKE.`Decrypt`

Input: (sk', ct)
Output: m'
1: $m' = \mathtt{Decode}(v - \mathbf{s}^\top \circ \mathbf{u})$

The *centered binomial sampler* module ($\mathcal{B}_\eta$) generates all the secret small polynomials in Kyber, such as the secret key $\mathbf{s}$ and random nonces ($\mathbf{r}$,$\mathbf{e}$,$\mathbf{e}_1$,e_2). The sampling procedure

Algorithm 7 Implementation of KYBER's Centered Binomial Sampler

Input: *Bufs* ▷ buffer containing 128 uniformly distributed pseudo-random bytes
Output: $\boldsymbol{s}_{\text{poly}}$ ▷ a noise polynomial with coefficients $s_0, \ldots, s_{255} \in \{-2, -1, 0, 1, 2\}$

1: **for** i **from** 0 **to** 31 **do**
2: $\quad t = \text{load32bits}(\mathit{Bufs}[4 \times i : 4 \times i + 3])$
3: $\quad d = t$ `&` `0x55555555`
4: $\quad d = d + ((t$ `>>` $1)$ `&` `0x55555555`$)$ ▷ computing 16 Hamming weights in parallel
5: $\quad$ **for** j **from** 0 **to** 7 **do**
6: $\quad\quad a = (d$ `>>` $(4 \times j))$ `&` `0x3` ▷ $\text{HW}(x)$
7: $\quad\quad b = (d$ `>>` $(4 \times j + 2))$ `&` `0x3` ▷ $\text{HW}(y)$
8: $\quad\quad s_{8 \times i + j} = a - b$ ▷ sample drawn from centered binomial distribution

first expands a 32-byte random seed, using the SHAKE-256 extendable output function, to obtain a uniformly distributed pseudo-random byte stream. Then, sampling a secret coefficient s from a binomial distribution is done by subtracting the Hamming weights (HW) of two η-bit values x and y taken from this pseudo-random byte stream:

$$s = \text{HW}(x) - \text{HW}(y)$$

In KYBER768, $\eta = 2$, i.e. each sample $s \in \{-2, -1, 0, 1, 2\}$ has one of five possible values.

While a constant-time implementation, such as Algorithm 7, is inherently secure against timing attacks, it is still potentially vulnerable to other side-channel attacks, such as power analysis. Both the ANSI C reference implementation [BDK+24] and the optimized pqm4 implementation [KPR+] adopt a similar technique, which utilizes 32-bit integers to compute 16 2-bit Hamming weights in parallel, as shown in line 4 of Algorithm 7. This design is not only efficient, but also reduces the Hamming-weight leakage signal of the random inputs x and y during the computation. However, such a design may not thwart profiling attacks from recovering the secret polynomial.

2.3 Attack Scenarios

All three KEM operations are susceptible to attacks on binomial sampling, each facing different threats:

	`KeyGen`	`Encaps`	`Decaps`
targeted secret	secret key sk'	nonce $\mathbf{r}$	nonce $\mathbf{r}$
attack trace count	single/multiple	single	multiple
profiling device	`Decaps`/cloned device	attacked device	attacked device

Attacking the binomial sampler called by `KeyGen` allows extracting the secret key sk'. This operation is typically executed only once per key pair, but multiple traces may be available if a memory-constrained device regenerates sk' from a stored random seed d for each decapsulation request.

Both `Encaps` and `Decaps` invoke the PKE.`Encrypt` subroutine, which uses the binomial sampler to generate a random nonce $\mathbf{r}$ from a seed r. If $\mathbf{r}$ is exposed, the attacker can recover the shared secret m as

$$\begin{aligned} \texttt{Decode}(v - \mathbf{t}^\top \circ \mathbf{r}) &= \texttt{Decode}(\mathbf{t}^\top \circ \mathbf{r} + e_2 + \texttt{Encode}(m) - \mathbf{t}^\top \circ \mathbf{r}) \\ &= \texttt{Decode}(e_2 + \texttt{Encode}(m)) = m \end{aligned}$$

given the public key $pk = (\mathbf{t}, \mathbf{A})$ and ciphertext $ct = (\mathbf{u}, v)$. `Encaps` also provides only a single trace for each $\mathbf{r}$ if, as required, each invocation gets a fresh seed r from a TRNG.

In contrast, `Decaps` may provide multiple traces for the same $\mathbf{r}$ when repeatedly invoked to decapsulate the same ciphertext ct.

Both `Decaps` and `Encaps` may allow direct profiling on the attacked device, without the need for porting templates.

In `Decaps`, the attacker can collect the profiling dataset by using the target's public key pk to generate ciphertexts $ct = \text{PKE.}\mathtt{Encrypt}(pk, m, r)$ with chosen messages m and seeds r. Profiling traces $\boldsymbol{T}_\text{p}$ can then be recorded when the attacked device decapsulates these ct and generates the nonces $\mathbf{r} \xleftarrow{\$} \mathcal{B}_\eta(r)$ from the attacker-chosen seeds r.

For `Encaps`, the attacker can first generate a key pair, pk and $sk = (sk', pk, h, z)$, and then obtain the profiling traces $\boldsymbol{T}_\text{p}$ and ciphertexts ct directly from the attacked device, by querying for encapsulations with pk. The attacker can then derive the nonce labels $\mathbf{r}$ used during encapsulation from ct via $m = \text{PKE.}\mathtt{Decrypt}(sk', ct)$, $(K, r) = \mathtt{G}(m \parallel h)$, and $\mathbf{r} \xleftarrow{\$} \mathcal{B}_\eta(r)$.

For `KeyGen`, direct profiling on a target device is infeasible, assuming the TRNG-generated seed d is inaccessible. Instead, attackers could perform profiling of $\mathbf{r}$ through invoking `Decaps` on the attacked device, and then port these templates to sk'. Alternatively, they may invoke `KeyGen` on a compromised cloned device, where d is accessible, and port those templates onto the attacked device. Both options require template porting while retaining high template accuracy for successful attacks.

2.4 Gaussian Mixture Model and EM Algorithm

Gaussian mixture models (GMM) can be used to represent the distribution of datasets with latent variables, which in our case are the side-channel recordings from an attacked device where we do *not* know a-priori the values of the targeted intermediate variables. Central to a GMM is the assumption that each data point in the observed dataset is drawn from one of several Gaussian distributions. The value of the latent variable determines which of these Gaussian distributions the data point was drawn from.

A Gaussian mixture model Θ consists of $|K_v|$ Gaussian models $\theta_k = (\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$, and also has for each a mixture weight π_k that sums up to $\sum_{k \in K_v} \pi_k = 1$ and models the probability distribution $\mathrm{P}(k)$ of the values $k \in K_v$ of the latent variable v. The probability density function of a Gaussian mixture model is formulated as a weighted sum of these Gaussian distributions:

$$f(\boldsymbol{t} \mid \Theta) = \sum_{k \in K_v} \pi_k p(\boldsymbol{t} \mid \theta_k)$$

During the profiling phase of a template attack, we use a multivariate Gaussian model to characterize the traces of each of the different values $k \in K_v$ of an intermediate variable v on a profiling device. We can construct a Gaussian mixture model for the leakage of v by combining all the MVG distributions for values $k \in K_v$. This models each trace from the recording as a sample drawn from an MVG distribution (template) selected by the processed intermediate value k. In the attack phase, however, an attacked device's intermediate values k are hidden from the attacker. As such, v becomes a latent variable, and the trace set from an attacked device can be modelled as a mixture of Gaussian distributions.

The Expectation–Maximization (EM) algorithm [DLR77] is a commonly-used method for estimating GMM parameters for a dataset, in scenarios where neither the parameters θ_k of the Gaussian distributions, nor the value k of the latent variable v selecting the distribution is known. In our case, the data points are the traces from an attacked device that have not been labelled with k. While we have Gaussian parameters θ_k from a profiling device, useful as a starting point, these may differ from those matching the attacked device.

The EM algorithm estimates the GMM parameters through an iterative process, consisting of alternating applications of the E-step and M-step, until a convergence

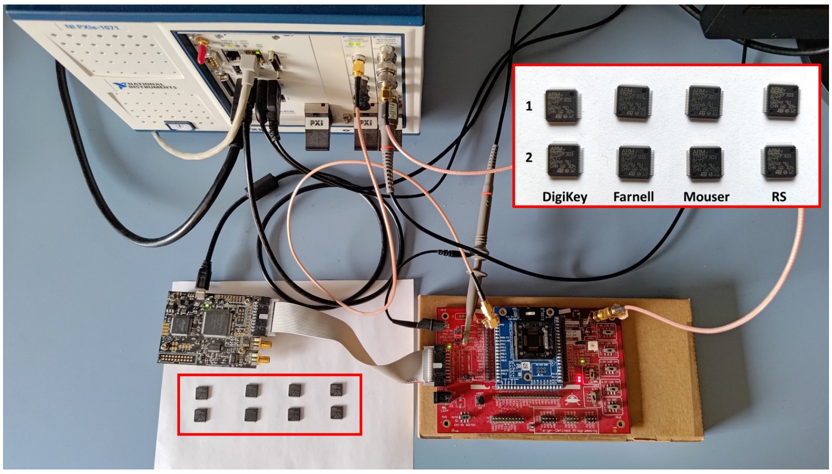


Figure 2: The NI scope and AWG connected to the CW308 UFO and CW308T-STM32F-SOCKET target board, and eight STM32F303RCT7 targets from different distributors.

criterion is met. Given an observed dataset $\boldsymbol{T}$ with a latent variable v, the EM algorithm typically involves the following steps:

Initialization: The algorithm requires an initial guess $\Theta^{(0)}$ for the GMM parameters, such as the Gaussian components θ_k for all $k \in K_v$.

E-step: The algorithm computes in iteration l for each data point $\boldsymbol{t}$ in the dataset $\boldsymbol{T}$, and for each value $k \in K_v$ that the latent variable v can hold, the probability that v held the value k when $\boldsymbol{t}$ was sampled, given the current parameter estimates $\Theta^{(l-1)}$. These probabilities are also called *responsibilities* $r^{(l)}(k, \boldsymbol{t}) = \mathrm{P}(v = k \mid \boldsymbol{t}, \Theta^{(l-1)})$.

M-step: The algorithm then updates the Gaussian components' parameters $\Theta^{(l)}$ based on the current responsibilities $r^{(l)}(k, \boldsymbol{t})$ estimated in the E-step.

Convergence criteria: The EM algorithm iterates between the E-step and M-step until a convergence criterion is met. Common convergence criteria include reaching a maximum number of iterations $l_{\max}$ or the overall likelihood change falling below a predefined threshold.

EM-generated GMMs were used previously in side-channel analysis to model a masked intermediate value [LP07, BCGR22]. Instead, here we model the whole trace dataset $\boldsymbol{T}$ as one GMM, and use the EM algorithm only to adjust the Gaussian components' parameters, i.e. $\theta_k = (\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$, to a specific device, while keeping the mixing weights π_k constant according to the (in our use case) known distributions of the targeted intermediate variables.

In our application of the EM algorithm, we rely on the initial $\Theta^{(0)}$ to have a significant influence on the final Θ_{adj}, as we want to preserve the mapping between k and θ_k. Therefore, our convergence criteria should consider that this mapping can deteriorate if the algorithm runs too long. This differs from other common applications of the EM algorithm, where $\Theta^{(0)}$ is chosen randomly and is not expected to significantly influence the outcome.

3 Measurement Setup

The CRYSTALS-Kyber768 implementation that we targeted in our experiments comes from pqm4 [KPR+], a library for ARM Cortex-M4 microcontrollers (MCUs) that contains implementations of PQC schemes proposed in the NIST standardization project. While pqm4 provides hand-optimized assembler code for many larger modules, such as the NTT and SHA-3, the binomial sampler module that we targeted implements the simple and efficient Algorithm 7 in C. We compiled it in our target firmware using ARM GCC 9.2.1, unless stated otherwise with optimization level `-Os`, the default in the ChipWhisperer environment.

Our power-analysis target is the blue CW308T-STM32F-SOCKET board sitting on top of the red CW308 UFO board shown in Figure 2. The former has a 64-pin TQFP socket that allows us to test various STM32F303RCT7 devices for template portability with the same measurement setup. The STM32F303RCT7 microcontroller has a 32-bit ARM Cortex-M4 core, which is the NIST-recommended embedded-system evaluation platform for their PQC project. We use eight such microcontrollers to test attack portability, bought from four different electronics distributors [DigiKey (DK), Farnell (FN), Mouser (MS), and RS] to obtain different manufacturing dates and batches, in the hope of increasing observed manufacturing variation. The UFO board inserts a $10\,\Omega$ shunt resistor into the VCC power-supply line for current measurements. As a result, this signal comes with an undesirable 3.3 V DC offset. We added an analog 4.5 MHz RC high-pass filter to remove this bias, while keeping the filter's impulse response to within a couple of clock cycles.

Our acquisition instrument is an NI PXIe-5160 10-bit oscilloscope with 2.5 GHz sampling rate and 500 MHz bandwidth. The target devices use an external clock source, a 5 MHz square wave provided by a function generator (AWG) phase locked to the oscilloscope's sampling clock. Clock-synchronous sampling reduces horizontal noise in the traces and thereby helps to increase the success rate of side-channel attacks [OC15]. Each trace is horizontally aligned before being digitally down-sampled with a Lanczos low-pass filter from 2.5 GHz to 100 MHz to reduce the data volume. For each device, we obtained 4,000 recordings of invocations of the binomial-sampling operation for profiling and 1,000 such recordings for testing.

4 Single-Device Single-Trace Attack

We target the secret-key generation of Kyber768, NIST's recommended default setting for Kyber-KEM. In this single-device setting, we perform profiling and attacks on the same device, targeting the same operation, here `KeyGen`. We initially attempted to build templates for the variables s, x, and y and process these in a factor graph with associated constraints using belief propagation (SASCA) [VGS14], inspired by the single-trace framework by Primas *et al.* [PPM17]. However, the parallel design in Algorithm 7 reduces leakage and template accuracy, and this attempt did not achieve single-trace attacks. Furthermore, due to Kyber's relatively large parameter set and high computation cost for verifying each candidate key (compared to symmetric cryptosystems), we decided not to rely on key enumeration. Therefore, we needed to incorporate more information and build better templates to achieve single-trace attacks.

4.1 Leakage Assessment for Binomial Sampler

Figure 3 shows the NICV [BDGN14] leakage assessment result for the binomial sampler. The result in the red frame confirms substantial leakage for the sampled secret s, and relatively little signal for the variables x and y, with NICV values below 0.4. We later discovered an interesting leakage signal from x and y located long before the sampling operation, shown in the yellow frame. This signal appears to originate from the use of byte-wise load/store operations[1] where SHAKE-256 returns outputs in pqm4. We refer to

[1] https://github.com/mupq/pqm4/blob/1eeb74e4106a80e26a9452e4793acd6f191fe413/common/keccakf1600.S#L672-L676

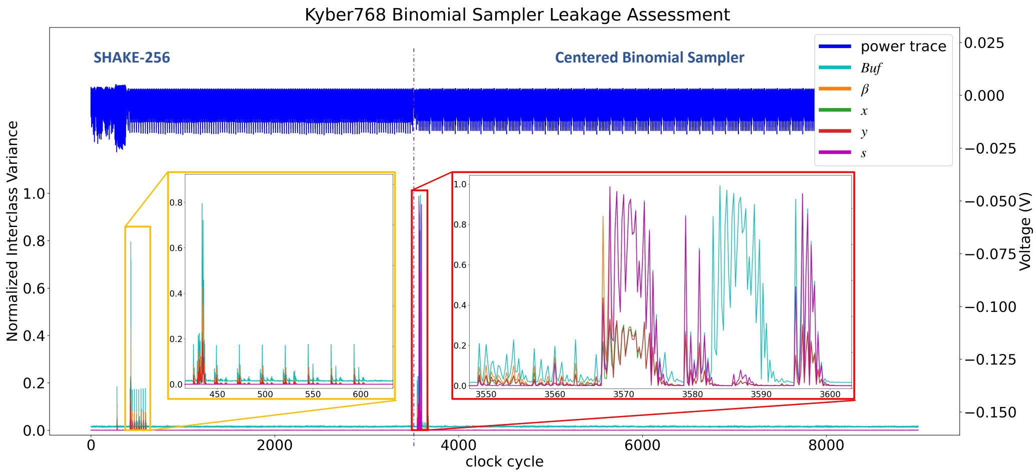


Figure 3: NICV of KYBER768's binomial sampler.

this early leakage as *Buf*, following the corresponding variable name later in the C code.

4.2 Template Attack and Marginalization

Based on the leakage assessment, we added *Buf* to the list of variables (s, x, and y) that we profile. Upon further testing, we found that high single-trace attack success rates can be achieved by merely marginalizing the *Buf* template results, without incorporating the more error-prone, low-SNR templates for x, y and s, which in fact reduced the overall success rate. Each *Buf* byte produces two binomial secrets

$$s_1 = \mathrm{HW}(\mathit{Buf} \,\&\, \mathtt{0x03}) - \mathrm{HW}(\mathit{Buf} \,\&\, \mathtt{0x0C}) = \mathrm{HW}(x_1) - \mathrm{HW}(y_1)$$
$$s_2 = \mathrm{HW}(\mathit{Buf} \,\&\, \mathtt{0x30}) - \mathrm{HW}(\mathit{Buf} \,\&\, \mathtt{0xC0}) = \mathrm{HW}(x_2) - \mathrm{HW}(y_2)$$

and marginalizing from $\mathrm{P}(\mathit{Buf} \mid \boldsymbol{t})$ we obtained their likelihoods

$$\mathrm{P}(s_1 \mid \boldsymbol{t}) = \sum_{\substack{\mathit{Buf} \in \mathbb{Z}_{256}, \\ \mathrm{HW}(\mathit{Buf}\,\&\,\mathtt{0x03}) - \mathrm{HW}(\mathit{Buf}\,\&\,\mathtt{0x0C}) = s_1}} \mathrm{P}(\mathit{Buf} \mid \boldsymbol{t})$$
$$\mathrm{P}(s_2 \mid \boldsymbol{t}) = \sum_{\substack{\mathit{Buf} \in \mathbb{Z}_{256}, \\ \mathrm{HW}(\mathit{Buf}\,\&\,\mathtt{0x30}) - \mathrm{HW}(\mathit{Buf}\,\&\,\mathtt{0xC0}) = s_2}} \mathrm{P}(\mathit{Buf} \mid \boldsymbol{t})$$

The variables x and y are 2-bit fragments of the 8-bit variable *Buf*, which itself is part of the 32-bit t in Algorithm 7. While LDA improves x and y templates by utilizing information from all relevant trace samples, including s and *Buf* and other micro-architectural leakages, the resulting template accuracy was insufficient for single-trace attacks. In contrast, profiling the larger *Buf* variable enhances the accuracy of marginalized $\mathrm{P}(s \mid \boldsymbol{t})$ for two reasons: (1) *Buf* directly leaks through the SHAKE-256 operation, and (2) *Buf* spans two sampling operations, which inherently captures leakage information from x_1, y_1, s_1 and x_2, y_2, s_2. However, a larger variable requires more templates, which might affect the performance of the EM algorithm.

We therefore also evaluated the 4-bit fragment templates [YK21] of *Buf*, which provide an intermediate size between the smaller 2-bit x and y variables and the larger 8-bit *Buf*. This 4-bit *Buf*-nibble offers a moderate number of Gaussian components, making it easy (for our later experiments) to visualize and understand the EM algorithm's behavior across

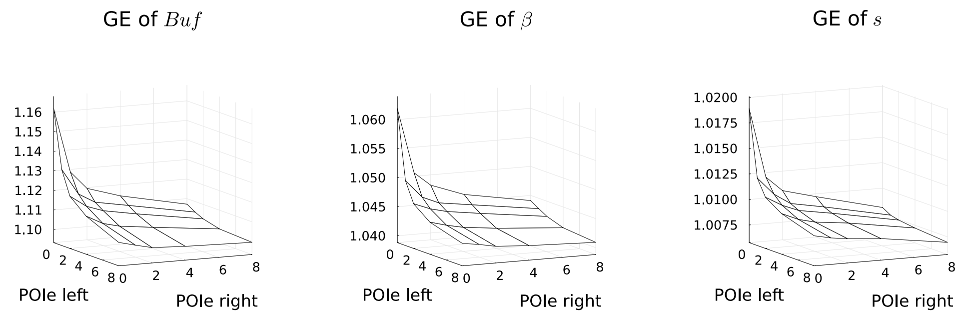


Figure 4: Guessing entropy (GE) drops with regard to POIe by clock cycles.

various initialization states in higher dimensions. Despite their lower SNR, these templates achieve high marginalized accuracy in the `-Os` setting. We refer to this 4-bit nibble as β, marginalized as:

$$\mathrm{P}(s \mid \boldsymbol{t}) = \sum_{\substack{\beta \in \mathbb{Z}_{16}, \\ \mathrm{HW}(\beta\ \texttt{\&}\ \texttt{0x3}) - \mathrm{HW}(\beta\ \texttt{\&}\ \texttt{0xC}) = s}} \mathrm{P}(\beta \mid \boldsymbol{t})$$

4.3 LDA-based Template Attack and Points of Interest Extension

For profiling, we exploit the repetitive nature of the sampling operation and split each recording of a noise-polynomial generation into multiple segments to increase our profiling dataset. In KYBER768, the secret key $\mathbf{s} \in R_q^3$ consists of three degree-255 polynomials generated by Algorithm 7. We observed waveform repetition for every eight *Buf*-storing operations of SHAKE-256, in the yellow frame of Figure 3, which are used by 16 sampling operations, in the red frame. Therefore, we split each recording of 768 coefficients s being sampled into 48 segments, each containing 16 sampling operations. This way, we turned our 4,000 recordings into $4{,}000 \times 48 = 192{,}000$ traces as our profiling dataset.

Since the LDA calculation requires more traces than there are time samples (points) per trace [SA08], and since its time complexity grows with the third power of the number of points per trace [CHH08], we first select which time samples of the recorded traces we include in the LDA as “points of interest” (POIs). We used NICV as the metric to select POIs when shortening our traces. In our experiments, we chose an NICV threshold of 0.004 for *Buf*, due to its larger size, and 0.001 for the smaller variables β and s.

Selecting only POIs that have at least some minimum signal content, e.g. as measured by NICV, makes sense for the calculation of an inter-class scatter matrix $\boldsymbol{B}$, which characterizes in which dimensions the best signal can be found. However, since the LDA also characterizes the noise of the recordings, via the intra-class scatter matrix $\boldsymbol{W}$, we wanted to try out if there are benefits from also deliberately including some time samples as POIs that have a *low* signal content, but due to their proximity might carry information about noise that is highly correlated with noise contained in high-signal POIs. The idea was that including such noise-correlated samples might allow the LDA to better project away from such noise. We call this method *point-of-interest extension* (POIe).

Even though POIe samples contain little signal from the targeted intermediate variable, they may still carry other relevant information, such as low-frequency noise or residual impulse response of previously processed data that interfere with nearby POIs that carry signal. We therefore tried adding a small number of clock cycles worth of additional POIe samples to the left or right of any high-signal POIs.

Figure 4 illustrates the guessing entropy drop that we achieved by extending our POI

Table 1: KYBER768.`KeyGen` template accuracies and attack results.

	`-Os`					`-O3`				
	Buf	β		s		*Buf*	β		s	
byte 1 accuracy	79.08 %	89.19 %	86.88 %	98.49 %	97.63 %	89.03 %	95.18 %	91.03 %	81.49 %	97.50 %
byte 2 accuracy	94.28 %	93.72 %	99.94 %	98.85 %	99.43 %	91.66 %	95.41 %	94.16 %	85.64 %	88.49 %
byte 3 accuracy	81.69 %	91.18 %	93.75 %	99.45 %	99.23 %	77.51 %	87.85 %	89.20 %	83.47 %	85.66 %
byte 4 accuracy	88.61 %	92.18 %	99.95 %	99.33 %	99.85 %	88.14 %	88.84 %	95.61 %	84.57 %	90.77 %
byte 5 accuracy	99.95 %	99.97 %	99.95 %	98.75 %	98.72 %	89.84 %	89.54 %	95.80 %	82.22 %	97.61 %
byte 6 accuracy	99.27 %	99.07 %	99.99 %	99.04 %	99.46 %	91.03 %	91.03 %	96.10 %	85.04 %	89.84 %
byte 7 accuracy	92.50 %	93.75 %	99.60 %	99.48 %	99.32 %	83.53 %	83.38 %	89.08 %	84.12 %	85.64 %
byte 8 accuracy	90.48 %	95.21 %	99.95 %	99.41 %	99.77 %	93.48 %	87.62 %	98.39 %	82.73 %	91.37 %
average	90.73 %	95.89 %		99.14 %		88.03 %	91.76 %		87.26 %	
s accuracy	100.00 %	100.00 %		99.14 %		99.76 %	97.70 %		87.26 %	
single-trace SR	100.00 %	99.30 %		0.20 %		28.90 %	0.00 %		0.00 %	

selection to prior (POIe left) and subsequent (POIe right) samples next to POIs selected by the NICV threshold. The wider the POIe region, the lower the guessing entropy. Our experiment sometimes showed an extra 10 % boost in template accuracy with a POIe of 2 clock cycles prior and 4 clock cycles after the selected POIs. We use in subsequent sections these POIe parameters as they provided a good balance between profiling efficiency and template accuracy.

4.4 Single-Device Single-Trace Attack Result

Recording and preprocessing 4,000 key-generation profiling traces took us around three hours. After segmenting the traces, we created 40 template sets ($8\times$*Buf*, $16\times\beta$, $16\times s$), with accuracies listed in Table 1. Building these templates using our selected NICV and POIe parameters required about an hour, with the LDA computation being the most time-consuming step. During the attack phase, using a 4-core CPU running the computationally efficient marginalization method, attacking the 1,000-recording test set took us only a few minutes for the 8-bit *Buf* variable and under a minute for the smaller β and s templates.

Table 1 shows the template and marginalized s accuracies (blue) and single-trace attack success rates (red) for a single device, DK1. The targeted KYBER768 implementation was compiled with two optimization levels, `-Os` and `-O3`, to evaluate the impact of different leakage levels. The `-Os` version exhibited more extensive leakage (Figure 3), while `-O3` exhibited reduced and shortened leakage due to the compiler-optimized sampling procedure. Each trace segment covers eight bytes of *Buf*, an equivalent of 16 sampling operations. When we use the POIe-improved LDA-based templates to directly profile on the variable s, we achieved an average template accuracy of 99.14 %. Still, the single-trace attack success rate remained near zero (0.2 %) due to the large number of s in a secret key. By marginalizing the *Buf* results (Section 4.2), the s accuracy reached 100 %, achieving a 100 % single-trace success rate. Marginalizing the fragmented β template resulted in a slightly lower s accuracy (99.9991 %) and single-trace success rate ($99.9991^{768} = 99.3$ %) but offered improved efficiency. Under the `-O3` optimization, lower leakage levels reduced template accuracy: the β and s templates failed to achieve sufficient accuracy for single-trace attacks, but marginalising the *Buf* template still yielded an s accuracy of 99.76 %, resulting in 28.9 % of our 1000 single-trace attacks to succeed, even though the average template accuracy of the 8-bit value *Buf* was only 88.03 %.

These results indicate the pqm4 KYBER binomial sampler exhibits sufficient leakage to enable single-trace key (or nonce) extraction attacks when compiled with `-Os`. However, success rates sometimes plummet when applying the templates across different devices (Table 2, top left). We address this by proposing the *adaptive template attack*, where we adjust templates to fit the side-channel characteristics of each target device.

5 Cross-Device Attacks with Adaptive Templates

To enhance cross-device template accuracy, we first analyzed the impact of inter-device variances on side-channel signals. By porting the LDA vectors from profiling to attacked devices, we found that they are still effective in separating traces of different intermediate values in the LDA subspace, even for low-SNR targets like the 2-bit variable x. In most cases, the projected data groups remained very much Gaussian-like. However, the location of the projected data often deviates from the profiled templates, indicating that some form of model adjustment is required to restore the template accuracy.

5.1 EM Algorithm for Adaptive Templates

We address the cross-device portability problem as a Gaussian-mixture-model parameter estimation task. Suppose an attacker has exhausted available profiling resources to optimize LDA projection vectors and build MVG templates. Despite efforts, a model mismatch persists between the attack trace set and the MVG templates in the LDA subspace. To address this, we apply the EM algorithm to fine-tune the MVG templates and align them better with the attack trace set.

During profiling, we obtain a complete (labelled) dataset $D_\mathrm{p} = [(\boldsymbol{t}'_1, k'_1), \ldots, (\boldsymbol{t}'_M, k'_M)]$ from profiling devices, in which each observable trace recording $\boldsymbol{t}'_i$ is paired with the corresponding known intermediate value $k'_i \in K$. Whereas the hypothetical dataset $D_\mathrm{a} = [(\boldsymbol{t}_1, k_1), \ldots, (\boldsymbol{t}_N, k_N)]$ describing the device under attack remains "incomplete" for an attacker who can only observe the side-channel traces $\boldsymbol{T}_\mathrm{a} = [\boldsymbol{t}_1, \ldots, \boldsymbol{t}_N]$, while the intermediate values (labels) k_i have now become the values of the hidden (latent) variable. Such a problem is a typical use case for the EM algorithm.

Our GMM Θ for the underlying distribution of D_a contains known mixture weights π_k and unknown Gaussian parameters $\theta_k = (\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ for all $k \in K$. For instance, our binomially sampled secret variable s is modelled as

$$\Theta_s = [(0.0625, \theta_{-2}), (0.25, \theta_{-1}), (0.375, \theta_0), (0.25, \theta_1), (0.0625, \theta_2)].$$

The EM algorithm then estimates a set of optimized Gaussian mixture parameters to fit the observed trace set $\boldsymbol{T}_\mathrm{a}$. Afterwards, we should have obtained a set of refined template models customized for the attacked device.

Algorithm 8 outlines the procedure of our adaptive template attack in three key steps (see Appendix A for the detailed subroutines):

GMM Initialization. We combine the MVG templates θ_k from the profiling device into an initial GMM $\Theta^{(0)}$. The distribution of mixture weights π_k is determined by the targeted algorithm. Intermediate variables in cryptographic algorithms often are uniformly distributed. (However, if the intermediate variable held, for example, a random value's Hamming weight, the corresponding weights $\pi_{\mathrm{HW}(v)}$ would have a binomial distribution.)

To initialize the Gaussian components, we shift for each LDA dimension and each component θ_k the mean μ_k to a new mean

$$\mu'_k = (\mu_k - \overline{\mu})\sigma_\mathrm{a}/\sigma + \overline{t_\mathrm{a}}$$

such that the overall mean and standard deviation $(\overline{\mu}, \sigma)$ across all GMM components θ_k match the corresponding values $(\overline{t_\mathrm{a}}, \sigma_\mathrm{a})$ estimated from the attack traces (Algorithm 9). This minimizes systematic bias and places the Gaussian templates closer to their data group, similar to trace normalization, but in the LDA subspace.

We then identify pairs of Gaussian components θ_k that are indistinguishable, by pairwise application of Hotelling's T^2-test, the multivariate analogue of the univariate t-test. Afterwards, we merge such indistinguishable components into one, adding their π_k

Algorithm 8 EM-based Template Adjustment

Input: $\Theta = \left[(\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \mid k \in K\right]$ ▷ Profiled templates as GMM
Input: $\boldsymbol{T}_{\mathrm{a,LDA}} = [\boldsymbol{t}_1, \boldsymbol{t}_2, \ldots, \boldsymbol{t}_N] \in \mathbb{R}^{d\times N}$ ▷ Same-LDA-projected attack-device trace set
Input: $l_{\max},\ \delta,\ scale_{\boldsymbol{\Sigma}}$
Output: $\Theta_{\mathrm{adj}} = \left[(\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \mid k \in K\right]$

 # GMM Initialization
$\left[(\pi_k, \boldsymbol{\mu}'_k, \boldsymbol{\Sigma}_k) \mid k \in K\right] = \mathrm{GMMNormalize}(\Theta, \boldsymbol{T}_{\mathrm{a,LDA}})$ ▷ unbias Θ
for all $k \in K$ **do**
  $\boldsymbol{\Sigma}'_k = \boldsymbol{\Sigma}_k \times scale_{\boldsymbol{\Sigma}}$ ▷ cover more distant data groups
$\Theta^{(0)}, labels_{\mathrm{idx}} = \mathrm{MergeComponents}\left(\left[(\pi_k, \boldsymbol{\mu}'_k, \boldsymbol{\Sigma}'_k) \mid k \in K\right]\right)$
 # EM Process
for l **from** 1 **to** $l_{\max}$ **do**
  $\Theta^{(l)} = \mathrm{EMprocess}(\Theta^{(l-1)}, \boldsymbol{T}_{\mathrm{a,LDA}})$
  # Termination Criteria
  $L^{(l)} = \mathcal{L}(\Theta^{(l)} \mid \boldsymbol{T}_{\mathrm{a,LDA}})$ ▷ GMM likelihood
  if $(L^{(l)} - L^{(l-1)})/(L^{(2)} - L^{(1)}) < \delta$ **then**
    break
$\Theta_{\mathrm{adj}} = \mathrm{UnmergeComponents}(\Theta^{(l)}, labels_{\mathrm{idx}})$

(Algorithm 10). This can reduce computation and lead to more variety among the mixture weights π_k, which can help to improve the EM process.

EM Process. At each iteration l, the EM algorithm alternates between the E-step and M-step to maximize the likelihood function (Algorithm 11). In the E-step, we calculate the posterior distribution of the intermediate variable v, with the given $\boldsymbol{T}_{\mathrm{a}}$ and $\Theta^{(l-1)}$. We estimate the responsibility

$$r_{k,i}^{(l)} = \frac{\pi_k p(\boldsymbol{t}_i \mid \theta_k^{(l-1)})}{\sum_{k' \in K_v} \pi_{k'} p(\boldsymbol{t}_i \mid \theta_{k'}^{(l-1)})}$$

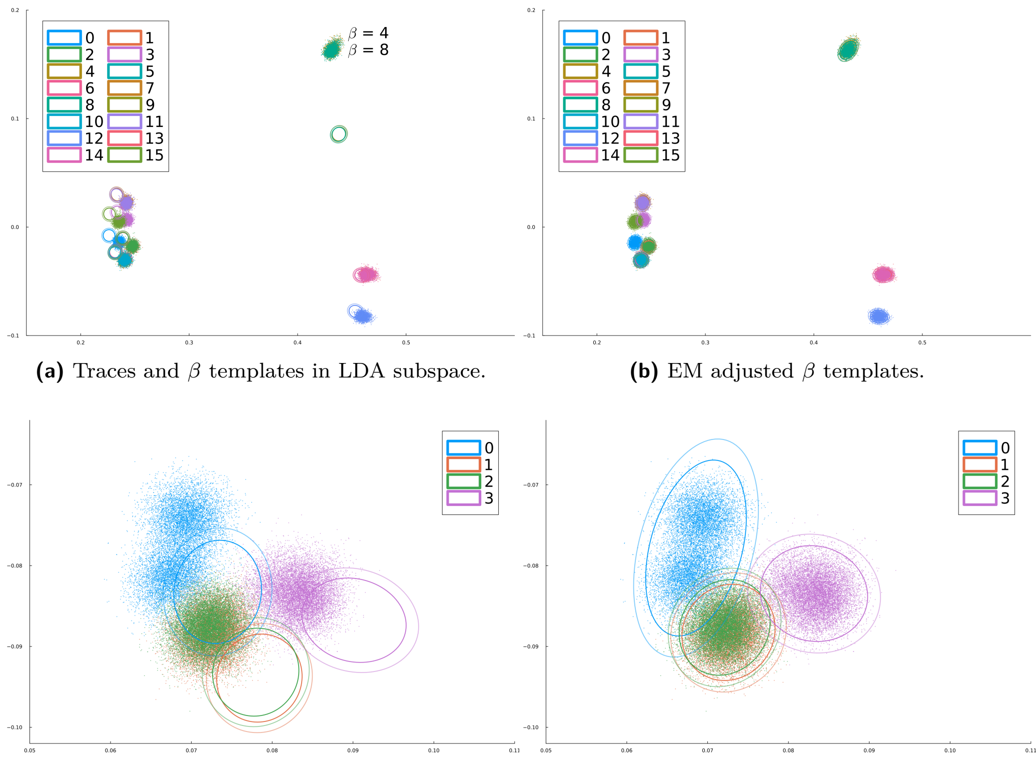
for each trace $\boldsymbol{t}_i$ and each Gaussian component $\theta_k^{(l-1)}$.

In the M-step, we estimate $\Theta^{(l)}$ by maximizing the likelihood expectation with the given responsibilities $r_{k,i}^{(l)}$ from the E-step. Simply put, we update the GMM parameters $\theta_k^{(l)} = (\boldsymbol{\mu}_k^{(l)}, \boldsymbol{\Sigma}_k^{(l)})$ given the trace set $\boldsymbol{T}_{\mathrm{a}}$ and the estimated responsibilities $r_{k,i}^{(l)}$:

$$\begin{aligned}
\boldsymbol{\mu}_k^{(l)} &= \frac{\sum_{i=1}^N r_{k,i}^{(l)} \boldsymbol{t}_i}{\sum_{i=1}^N r_{k,i}^{(l)}} \\
\boldsymbol{\Sigma}_k^{(l)} &= \frac{\sum_{i=1}^N r_{k,i}^{(l)} (\boldsymbol{t}_i - \boldsymbol{\mu}_k^{(l)})(\boldsymbol{t}_i - \boldsymbol{\mu}_k^{(l)})^\top}{\sum_{i=1}^N r_{k,i}^{(l)}}
\end{aligned}$$

Again, we know the mixture weights π_k of an intermediate variable v from the design of the cryptosystem, and therefore do not update them.

Errors may occur if one of the components dominates multiple data groups, leaving other components without any assigned data after the responsibility estimation. In such cases, we scale up the initial covariance with a $scale_{\boldsymbol{\Sigma}}$ factor to expand the initial coverage of each template, allowing the Gaussian components to explore more distant data groups, and restart the EM adjustment in Algorithm 8. Note that this covariance expansion may further reduce the number of Gaussian components and the resulting template accuracy.



(a) Traces and β templates in LDA subspace. **(b)** EM adjusted β templates.

(c) Traces and x templates in LDA subspace. **(d)** EM adjusted x templates.

Figure 5: EM-based adjustments with DK2 profiling templates and the MS2 target device.

Termination Criteria. We terminate the EM algorithm after a maximum number of iterations or when the relative likelihood increment ratio falls below a chosen threshold $\delta < 10^{-9}$. Typically, the algorithm converges within a few iterations. Note that templates merged during the initialization end up having the same parameters $(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ after EM adjustment (Algorithm 12), trading off some separability for adaptability.

5.2 Adaptive Template Attack Evaluation

Figure 5 illustrates the effect of EM-based template adjustment on the first two LDA dimensions for the 4-bit nibble β and 2-bit variable x. We apply the LDA projection vectors from the DK2 profiling device to the MS2 attack trace set $\boldsymbol{T}_{\mathrm{a,MS2}}$. In Figure 5a, the scattered dots represent the attack traces in the LDA subspace, color-coded by their latent intermediate value $\beta_i \in K_\beta$. The original multivariate Gaussian templates θ_β for DK2 are the ellipses with matching colors. Although there are 16 ellipses, only nine distinct clusters are formed in this two-dimensional subspace, corresponding to the nine Hamming weight combinations of the two 2-bit variables x and y. The remaining LDA dimensions further separate these overlapping clusters. However, if using only these two dimensions, the GMM initialization in Algorithm 8 would merge the ellipses into nine Gaussian components. In addition, this example also demonstrates the importance of GMM initialization based on a set of known template models θ_k. It would be challenging to correctly assign components to their respective latent values with a randomly initialized 16-component GMM, even though they may lead to identical parameter sets with the EM algorithm.

Before EM adjustment (Figure 5a), the data clusters show misalignment with the profiling templates, including systematic shifts and relative positional drift. Notably, the

Table 2: Single-trace attack success rate of KYBER768.`KeyGen` from `KeyGen`-*Buf* templates.

Target / Profiling	DK1	DK2	FN1	FN2	MS1	MS2	RS1	RS2	DK1	DK2	FN1	FN2	MS1	MS2	RS1	RS2
	Template Attack								Template Attack + EM-Adjusted Templates							
DK1	100.0 %	99.8 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	80.2 %	100.0 %	100.0 %
DK2	94.1 %	100.0 %	0.0 %	0.0 %	0.4 %	0.0 %	25.1 %	70.7 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	2.0 %	44.7 %	100.0 %
FN1	0.0 %	0.0 %	100.0 %	84.6 %	12.6 %	0.0 %	0.0 %	0.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	28.3 %	100.0 %	100.0 %
FN2	0.0 %	0.0 %	87.6 %	100.0 %	74.1 %	53.4 %	0.0 %	0.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %
MS1	0.0 %	0.2 %	0.4 %	86.1 %	100.0 %	92.2 %	0.5 %	14.2 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	99.9 %	100.0 %	100.0 %
MS2	0.0 %	0.0 %	0.0 %	41.2 %	26.7 %	100.0 %	0.0 %	0.4 %	100.0 %	97.2 %	100.0 %	100.0 %	100.0 %	100.0 %	99.9 %	100.0 %
RS1	1.1 %	29.0 %	0.0 %	0.0 %	0.0 %	18.7 %	100.0 %	99.9 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %
RS2	0.0 %	22.3 %	0.0 %	0.0 %	0.0 %	1.5 %	99.9 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %
	Traces Normalized								Traces Normalized + EM-Adjusted Templates							
DK1	100.0 %	100.0 %	100.0 %	99.9 %	100.0 %	52.6 %	97.5 %	99.9 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	78.2 %	100.0 %	100.0 %
DK2	100.0 %	100.0 %	100.0 %	94.6 %	95.7 %	12.0 %	54.6 %	95.6 %	100.0 %	100.0 %	100.0 %	100.0 %	99.9 %	77.7 %	78.4 %	100.0 %
FN1	100.0 %	100.0 %	100.0 %	99.5 %	99.8 %	32.7 %	90.7 %	99.9 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	86.7 %	98.8 %	100.0 %
FN2	99.7 %	91.7 %	99.7 %	100.0 %	100.0 %	92.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	99.9 %	100.0 %	100.0 %
MS1	99.5 %	93.5 %	99.0 %	100.0 %	100.0 %	76.3 %	99.9 %	100.0 %	100.0 %	99.9 %	100.0 %	100.0 %	100.0 %	97.5 %	100.0 %	100.0 %
MS2	45.4 %	2.3 %	8.8 %	93.6 %	67.3 %	100.0 %	99.4 %	91.6 %	100.0 %	75.0 %	79.3 %	99.9 %	99.9 %	100.0 %	100.0 %	100.0 %
RS1	94.1 %	36.9 %	87.9 %	100.0 %	99.6 %	95.7 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	99.9 %	100.0 %	100.0 %
RS2	100.0 %	94.2 %	99.2 %	100.0 %	100.0 %	84.2 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	97.9 %	100.0 %	100.0 %
	Multi-Device Training								Multi-Device Training + EM-Adjusted Templates							
w/o RS2	100.0 %	99.9 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	98.3 %	100.0 %	99.9 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %
w/o RS1	100.0 %	99.9 %	100.0 %	100.0 %	100.0 %	100.0 %	99.8 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %
w/o MS2	100.0 %	99.9 %	100.0 %	100.0 %	100.0 %	99.5 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %
w/o MS1	100.0 %	99.9 %	100.0 %	100.0 %	94.8 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %
w/o FN2	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %
w/o FN1	100.0 %	100.0 %	99.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %
w/o DK2	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %
w/o DK1	98.8 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %

data groups $\beta = 4$ and $\beta = 8$ deviate upwards, and the three right-hand-side clusters exhibit skewness, possibly caused by cross-device LDA vector mismatch or unexpected noise contaminating the recordings. After applying EM-based adjustment, templates realign with attack traces (Figure 5b), improving accuracy from 54.88 % to 84.15 %. While this is lower than the single-device accuracy of 95.89 % (Table 1), it represents the best model achievable under the given LDA vectors from DK2. Note that these figures illustrate only the first two dimensions of the LDA space: the EM adjustment is performed in the full 15-dimensional LDA space, where the 16 clusters are better separated than Figure 5a suggests, but such higher-dimensional spaces are impractical to visualize.

Figure 5c shows as a lower-SNR case (NICV < 0.4) the 2-bit variable x. The LDA vectors remain effective in separating these noisier data groups, and the EM adjustment improves template accuracy from 51.93 % to 89.26 %. However, as shown in Figure 5d, the overlapping template region increases, leading to a drop in their accuracy. The blue data group exhibits two Gaussian-like clusters instead of one, corresponding to the groups of $\beta = 4$ and $\beta = 8$ in the previous Figure 5a, whereas the orange and green groups mostly overlap. Such a split or merger of Gaussian-like clusters could mislead an EM algorithm into allocating GMM components incorrectly, e.g. by allocating two different Gaussian components to the blue group. However, as Figure 5d shows, this didn't happen here. The combination of having the right relative position of components in the initial estimate, along with fixing the number of GMM components and, importantly, fixing the weights π_k, improves the robustness of this EM application. Without such constraints, a more free-roaming EM algorithm would have incorrectly allocated the orange component to half of the blue data.

5.3 Cross-Device Single-Trace Attack Result

The computational complexity of the EM algorithm depends on the size and dimensionality of the GMM model, which is determined by the number $|K_v|$ of intermediate-variable values and the number of dimensions d_v used in the LDA space. For the 8-bit variable *Buf*, with $|K_{Buf}| = 256$ and our chosen dimensionality $d_{Buf} = 16$, each EM iteration takes a few seconds to compute. Initially, we often encountered numerical errors during the EM process and the model often started degrading after a few iterations. To mitigate this problem, we increased the initial coverage of each template model θ_k by setting $scale_{\mathbf{\Sigma}} = 4$

Table 3: Single-trace attack success rate of KYBER768.`Encaps` from `KeyGen`-*Buf* templates.

Target / Profiling	DK1	DK2	FN1	FN2	MS1	MS2	RS1	RS2	DK1	DK2	FN1	FN2	MS1	MS2	RS1	RS2
	Template Attack								Template Attack + EM-Adjusted Templates							
DK1	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	97.0 %	86.3 %	65.1 %	56.9 %	61.8 %	0.6 %	63.0 %	62.6 %
DK2	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	92.9 %	93.4 %	71.6 %	25.3 %	45.6 %	0.0 %	5.7 %	40.2 %
FN1	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	94.6 %	95.9 %	80.5 %	42.1 %	61.9 %	0.1 %	52.7 %	47.6 %
FN2	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	92.4 %	83.8 %	84.8 %	89.6 %	79.1 %	41.8 %	94.5 %	98.7 %
MS1	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	87.4 %	86.7 %	74.9 %	66.8 %	68.4 %	51.2 %	83.7 %	55.8 %
MS2	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	82.6 %	62.1 %	65.6 %	76.0 %	83.0 %	78.7 %	91.6 %	79.8 %
RS1	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	85.9 %	66.7 %	77.5 %	88.7 %	73.1 %	73.5 %	97.9 %	87.9 %
RS2	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	90.3 %	82.9 %	85.5 %	76.1 %	85.5 %	50.2 %	94.6 %	96.8 %
	Traces Normalized								Traces Normalized + EM-Adjusted Templates							
DK1	23.2 %	15.6 %	10.3 %	6.5 %	4.8 %	0.4 %	2.9 %	8.1 %	97.9 %	84.3 %	82.2 %	71.9 %	59.7 %	9.7 %	60.5 %	66.6 %
DK2	10.4 %	14.2 %	5.0 %	1.4 %	1.2 %	0.0 %	0.4 %	2.0 %	94.3 %	93.0 %	71.0 %	48.9 %	49.5 %	3.1 %	19.0 %	44.4 %
FN1	19.1 %	14.1 %	9.8 %	4.9 %	3.7 %	0.0 %	1.3 %	3.9 %	99.5 %	95.8 %	93.1 %	55.6 %	70.7 %	6.2 %	62.7 %	57.1 %
FN2	15.2 %	3.0 %	7.5 %	15.0 %	5.6 %	1.4 %	11.7 %	15.1 %	96.9 %	83.8 %	93.4 %	95.5 %	85.9 %	50.4 %	93.4 %	96.3 %
MS1	9.3 %	3.2 %	4.9 %	4.7 %	9.9 %	1.0 %	7.5 %	11.3 %	95.0 %	88.9 %	80.4 %	85.9 %	84.1 %	64.9 %	84.1 %	79.5 %
MS2	0.0 %	0.0 %	0.0 %	3.1 %	0.3 %	6.4 %	8.8 %	3.5 %	69.1 %	31.3 %	49.1 %	83.3 %	83.6 %	90.1 %	89.3 %	80.7 %
RS1	3.8 %	0.1 %	1.3 %	11.1 %	6.2 %	4.4 %	26.2 %	19.9 %	89.9 %	75.1 %	84.1 %	94.0 %	79.5 %	87.9 %	99.7 %	91.3 %
RS2	10.8 %	2.1 %	3.5 %	8.0 %	7.2 %	2.2 %	11.7 %	14.2 %	95.1 %	86.5 %	95.6 %	90.1 %	94.7 %	63.4 %	87.7 %	92.0 %
	Multi-Device Training								Multi-Device Training + EM-Adjusted Templates							
w/o RS2	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	89.0 %	93.5 %	87.4 %	68.5 %	62.4 %	68.4 %	92.7 %	64.7 %
w/o RS1	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	86.8 %	83.2 %	69.8 %	62.8 %	53.1 %	57.1 %	79.9 %	69.1 %
w/o MS2	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	91.8 %	90.7 %	68.5 %	56.5 %	60.0 %	43.0 %	78.6 %	63.9 %
w/o MS1	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	85.3 %	72.1 %	68.4 %	55.4 %	52.1 %	49.3 %	79.3 %	62.6 %
w/o FN2	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	93.8 %	81.6 %	75.9 %	57.5 %	68.1 %	50.4 %	85.3 %	68.9 %
w/o FN1	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	89.2 %	84.5 %	86.9 %	81.2 %	67.3 %	78.6 %	91.6 %	82.6 %
w/o DK2	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	94.5 %	90.6 %	84.4 %	71.1 %	66.8 %	61.1 %	87.9 %	74.2 %
w/o DK1	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	87.9 %	75.7 %	74.7 %	68.3 %	66.4 %	54.4 %	84.4 %	65.8 %

during GMM initialization in Algorithm 8. We also capped the iteration limit at $l_{\max} = 20$ to prevent prolonged optimization. As a result, completing an EM-based adaptive template attack (including evaluation) for a 1,000-trace set $\boldsymbol{T}_\mathrm{a}$ takes approximately 15 minutes.

We tested the adaptive templates across multiple devices to evaluate their effectiveness in cross-device attack scenarios and compare them to other popular portability techniques. The single-trace attack success rates for these tests appear in Table 2. We used all eight devices for both profiling and attacks. For the multi-device training configuration (bottom blocks), we pooled traces from seven profiling datasets to form a combined training dataset of 192,000 traces, matching the size of the single-device dataset. Each attack trace set $\boldsymbol{T}_\mathrm{a}$ consists of 1,000 `KeyGen` recordings, separate from the profiling set. Note that we employed the entire trace set $\boldsymbol{T}_\mathrm{a}$ for the EM-based template adjustment; however, each attack utilized only a single trace to recover all 768 secret values s, achieving single-trace attacks. This setup reflects a realistic attack scenario for a nonce target, where an attacker collects multiple traces from a device, but each secret appears only once.

The top-left block in Table 2 presents attack success rates in the multi-device setting without modifications. Many templates fail to achieve portable single-trace attacks across devices in this configuration. Successful attacks primarily involve devices from the same distributor with matching engraved manufacturing dates. Applying the trace normalization method [EG12] (middle-left block) significantly improves success rates in most cases. However, our EM adjustment method (top-right block) further enhances template portability, except for the MS2 target device. For MS2, a substantial discrepancy between the DK2 templates and the attacked dataset causes Gaussian components to miss their respective data groups even after the EM adjustment, lowering the success rates. Combining both techniques (middle-right block) results in further improvements. With the multi-device-trained templates (bottom-left block), only the anti-diagonal cells are relevant, as they represent the unseen target device for each template. These templates demonstrate excellent cross-device attack capabilities, achieving near-100 % success rates across the anti-diagonal cells. Incorporating our EM adjustment provides a final boost, completing the attacks with 100 % success rates.

We also investigate cross-operation portability, which enables attacks to extend from `Decaps` to the "unprofilable" `KeyGen` operation. This analysis essentially focuses on the address-space portability of templates, as discussed in Section 2.3. We collected 1,000 traces of the `Encaps` operations from all eight devices as the attack trace set $\boldsymbol{T}_{\mathrm{a}}$. Table 3 reports single-trace attack success rates on the random nonce $\mathbf{r}$ in `Encaps` using *Buf* templates from `KeyGen`. Success rates drop across all devices and all portability techniques compared to the attacks on `KeyGen`. This decline shows the effect of additional side-channel variation introduced by the address-space differences.

The commonly used portability techniques listed on the left side of Table 3 demonstrate poor adaptability to the address-space changes. Single-trace attacks using unmodified templates (top-left block) failed entirely, even on the same device, and multi-device-trained templates (bottom-left block) also showed no success. The normalization method (middle-left block) achieved limited success, with an average success rate of 6.94 %. In contrast, the EM adjustment technique (right side) performed significantly better. EM adjustment alone (top-right block) achieved an average success rate of 70.6 %, with some targets exceeding 90 %. Combining EM adjustment with normalization (middle-right block) further increased the average success rate to 75.6 %. Pairing multi-device templates with EM adjustment (bottom-right block) improved success rates from complete failure to an average of 73.7 %. These findings indicate that traditional portability techniques are less effective against address-space variance while incorporating the EM adjustment technique will restore template accuracy and improve attack results.

5.4 Caveats, Limitations and Mitigations

While our results demonstrate the advantages of the EM-based adaptive template attack, the method also has limitations.

Labeling errors. The EM algorithm cannot guarantee a correct mapping between the adjusted templates $\theta_{k,\mathrm{adj}}$ and the latent value k. Clusters may have drifted too far from their original locations. Our GMMNormalize function (Algorithm 9) aims to reduce such errors by initializing templates near their expected locations. The MergeComponents function (Algorithm 10) helps to reduce the risk of GMM components latching onto the wrong data clusters by bundling closely co-located templates, under the assumption that their data also cluster closely on the target. But merging GMM components can also decrease template accuracy, especially when this assumption fails.

GMM assumption. The assumption that the data fits a Gaussian mixture model may not always hold when porting the LDA vectors from a profiling setting to different target environments. Significant discrepancies in the leakage model can distort the data distribution in the LDA space, as can be seen in Figure 5c (blue). Similarly, a noise source affecting only a subset of recordings may split a cluster into multiple groups. In such cases, the EM algorithm may struggle to produce well-adjusted templates. Mitigation strategies include using multi-device and multi-environment profiling data to obtain more robust LDA projection vectors or removing outlier traces to preserve the GMM assumption.

Large attack dataset requirements. For profiling, we need enough traces to satisfy the accuracy requirements of (a) POI selection, (b) LDA, and (c) the accurate estimation of the MVG parameters in LDA space. For the attacked device, we do not have to perform (a) and (b), but we still need enough traces for (c), because the M-step also does that. The larger the number of Gaussian components $|K_v|$, the more accurate this estimate will have to be. One mitigation could be to replace the individual covariance matrices $\mathbf{\Sigma}_k$ with a common one [CK13], $\mathbf{\Sigma}_{\mathrm{pooled}} = \sum_{k \in K_v} \pi_k \mathbf{\Sigma}_k$.

The effectiveness of the EM algorithm depends on trace availability and the validity of the GMM assumption. The accuracy of EM-adjusted templates is constrained by the quality of LDA projections, with better signal-preserving and noise-rejecting vectors yielding better results.

Our attack aims to identify *all* secret polynomial coefficients correctly. However, in case some coefficients are missing or erroneous, lattice-reduction-based cryptanalysis methods using the BKZ algorithm, such as the leaky-LWE-Estimator [DDGR20] or refined two-step estimation [XWW+24], may be applicable to recover those.

5.5 Countermeasures

Our single-trace attack relies on a relatively large leakage signal from the target devices to achieve a high template accuracy. Therefore, hardware implementations of Kyber designed to minimize power signals, or software implementations running on secure microcontrollers designed to have a reduced signal-to-noise ratio, should be less exposed. Our findings regarding the SHAKE-256 implementation in pqm4 (Section 4.1) also highlight the benefits of avoiding byte-wise handling of secret data. Countermeasures such as masking and shuffling could also be added. Shuffling has a relatively small overhead and could increase the attack complexity considerably, given Kyber's large parameter set. Masked binomial samplers have been proposed [SPOG19] and implemented in masked Kyber implementations [OSPG18, BGR+21, HKL+22]. Whether these countermeasures are effective against our single-trace attacks requires further investigation.

6 Conclusion

We demonstrated how targeting the binomial sampler with profiled power-analysis attacks can enable single-trace key, nonce, and message extraction on all Kyber operations. In our target, this leakage was significant enough for our templates to achieve nearly 100 % accuracy in identifying the sampled secrets. We also provide new techniques to improve template accuracy (POIe) and template portability (adaptive template attack), both across different devices and different address spaces. The latter can help port templates between different Kyber operations that call the same implementation of the sampler as a subroutine.

We thank the reviewers for their helpful suggestions.

For the data and Julia code that produced some of the tables and figures in this paper see

https://www.cl.cam.ac.uk/research/security/datasets/kyber/

References

[ABD+21] Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Kyber algorithm specifications and supporting documentation (version 3.02). *NIST PQC Round 3*, pages 1–43, 2021.

[BCGR22] Julien Béguinot, Wei Cheng, Sylvain Guilley, and Olivier Rioul. Side-channel expectation-maximization attacks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(4):774–799, 2022.

[BCH+20] Shivam Bhasin, Anupam Chattopadhyay, Annelie Heuser, Dirmanto Jap, Stjepan Picek, and Ritu Ranjan Shrivastwa. Mind the portability: A warriors guide through realistic profiled side-channel analysis. In *27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23–26, 2020*. The Internet Society, 2020.

[BDGN14] Shivam Bhasin, Jean-Luc Danger, Sylvain Guilley, and Zakaria Najm. NICV: Normalized inter-class variance for detection of side-channel leakage. In *2014 International Symposium on Electromagnetic Compatibility, Tokyo*, pages 310–313. IEEE, 2014.

[BDK+24] Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. The official reference implementation of the Kyber key-encapsulation mechanism, Aug. 2024. https://github.com/pq-crystals/kyber/tree/10b478fc3cc4ff6215eb0b6a11bd758bf0929cbd/ref.

[BGR+21] Joppe W. Bos, Marc Gourjon, Joost Renes, Tobias Schneider, and Christine van Vredendaal. Masking Kyber: First- and higher-order implementations. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(4):173–214, 2021.

[CHH08] Deng Cai, Xiaofei He, and Jiawei Han. Training linear discriminant analysis in linear time. In *Proceedings of the 24th International Conference on Data Engineering, ICDE 2008, April 7–12, 2008, Cancún, Mexico*, pages 209–217. IEEE Computer Society, 2008.

[CK13] Omar Choudary and Markus G. Kuhn. Efficient template attacks. In *Smart Card Research and Advanced Applications – 12th International Conference, CARDIS 2013, Berlin, Germany, November 27–29, 2013. Revised Selected Papers*, volume 8419 of *Lecture Notes in Computer Science*, pages 253–270. Springer, 2013.

[CK18] Marios O. Choudary and Markus G. Kuhn. Efficient, portable template attacks. *IEEE Trans. Inf. Forensics Secur.*, 13(2):490–501, 2018.

[CRR02] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In *Cryptographic Hardware and Embedded Systems – CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13–15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*, pages 13–28. Springer, 2002.

[CZLG21] Pei Cao, Chi Zhang, Xiangjun Lu, and Dawu Gu. Cross-device profiled side-channel attack with unsupervised domain adaptation. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(4):27–56, 2021.

[DDGR20] Dana Dachman-Soled, Léo Ducas, Huijing Gong, and Mélissa Rossi. LWE with side information: Attacks and concrete security estimation. In *Advances in Cryptology – CRYPTO 2020 – 40th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17–21, 2020, Proceedings, Part II*, volume 12171 of *Lecture Notes in Computer Science*, pages 329–358. Springer, 2020.

[DLR77] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 09 1977.

[EG12] M. Abdelaziz Elaabid and Sylvain Guilley. Portability of templates. *J. Cryptogr. Eng.*, 2(1):63–74, 2012.

[FO99] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In *Advances in Cryptology – CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15–19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 537–554. Springer, 1999.

[HKL+22] Daniel Heinz, Matthias J. Kannwischer, Georg Land, Thomas Pöppelmann, Peter Schwabe, and Amber Sprenkels. First-order masked Kyber on ARM Cortex-M4, 2022.

[KPP20] Matthias J. Kannwischer, Peter Pessl, and Robert Primas. Single-trace attacks on Keccak. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(3):243–268, 2020.

[KPR+] Matthias J. Kannwischer, Richard Petri, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. PQM4: Post-quantum crypto library for the ARM Cortex-M4. https://github.com/mupq/pqm4/tree/1eeb74e4106a80e26a9452e4793acd6f191fe413/crypto_kem/kyber768/m4fspeed.

[LP07] Kerstin Lemke-Rust and Christof Paar. Gaussian mixture models for higher-order side channel analysis. In *Cryptographic Hardware and Embedded Systems – CHES 2007, 9th International Workshop, Vienna, Austria, September 10–13, 2007, Proceedings*, volume 4727 of *Lecture Notes in Computer Science*, pages 14–27. Springer, 2007.

[MBTL13] David P. Montminy, Rusty O. Baldwin, Michael A. Temple, and Eric D. Laspe. Improving cross-device attacks using zero-mean unit-variance normalization. *J. Cryptogr. Eng.*, 3(2):99–110, 2013.

[MOP07] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks – revealing the secrets of smart cards*. Springer, 2007.

[Nat24] National Institute of Standards and Technology. Module-lattice-based key-encapsulation mechanism standard. Federal Information Processing Standards Publication FIPS 203, Department of Commerce, Washington, D.C., 2024.

[OC15] Colin O'Flynn and Zhizhang Chen. Synchronous sampling and clock recovery of internal oscillators for side channel analysis and fault injection. *J. Cryptogr. Eng.*, 5(1):53–69, 2015.

[OSPG18] Tobias Oder, Tobias Schneider, Thomas Pöppelmann, and Tim Güneysu. Practical CCA2-secure and masked Ring-LWE implementation. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(1):142–174, 2018.

[PP19] Peter Pessl and Robert Primas. More practical single-trace attacks on the number theoretic transform. In *Progress in Cryptology – LATINCRYPT 2019 – 6th International Conference on Cryptology and Information Security in Latin America, Santiago de Chile, Chile, October 2–4, 2019, Proceedings*, volume 11774 of *Lecture Notes in Computer Science*, pages 130–149. Springer, 2019.

[PPM17] Robert Primas, Peter Pessl, and Stefan Mangard. Single-trace side-channel attacks on masked lattice-based encryption. In *Cryptographic Hardware and Embedded Systems – CHES 2017 – 19th International Conference, Taipei, Taiwan, September 25–28, 2017, Proceedings*, volume 10529 of *Lecture Notes in Computer Science*, pages 513–533. Springer, 2017.

[RBRC22] Prasanna Ravi, Shivam Bhasin, Sujoy Sinha Roy, and Anupam Chattopadhyay. On exploiting message leakage in (few) NIST PQC candidates for practical message recovery attacks. *IEEE Trans. Inf. Forensics Secur.*, 17:684–699, 2022.

[RCDB24] Prasanna Ravi, Anupam Chattopadhyay, Jan-Pieter D'Anvers, and Anubhab Baksi. Side-channel and fault-injection attacks over lattice-based post-quantum schemes (Kyber, Dilithium): Survey and new results. *ACM Trans. Embed. Comput. Syst.*, 23(2):35:1–35:54, 2024.

[RRB+19] Prasanna Ravi, Debapriya Basu Roy, Shivam Bhasin, Anupam Chattopadhyay, and Debdeep Mukhopadhyay. Number "not used" once – practical fault attack on pqm4 implementations of NIST candidates. In *Constructive Side-Channel Analysis and Secure Design – 10th International Workshop, COSADE 2019, Darmstadt, Germany, April 3–5, 2019, Proceedings*, volume 11421 of *Lecture Notes in Computer Science*, pages 232–250. Springer, 2019.

[SA08] François-Xavier Standaert and Cédric Archambeau. Using subspace-based template attacks to compare and combine power and electromagnetic information leakages. In *Cryptographic Hardware and Embedded Systems – CHES 2008, 10th International Workshop, Washington, D.C., USA, August 10–13, 2008. Proceedings*, volume 5154 of *Lecture Notes in Computer Science*, pages 411–425. Springer, 2008.

[SKL+20] Bo-Yeon Sim, Jihoon Kwon, Joohee Lee, Il-Ju Kim, Taeho Lee, Jaeseung Han, Hyo Jin Yoon, Jihoon Cho, and Dong-Guk Han. Single-trace attacks on message encoding in lattice-based KEMs. *IEEE Access*, 8:183175–183191, 2020.

[SPOG19] Tobias Schneider, Clara Paglialonga, Tobias Oder, and Tim Güneysu. Efficiently masking binomial sampling at arbitrary orders for lattice-based crypto. In *Public-Key Cryptography – PKC 2019*, pages 534–564, Cham, 2019. Springer International Publishing.

[VGS14] Nicolas Veyrat-Charvillon, Benoît Gérard, and François-Xavier Standaert. Soft analytical side-channel attacks. In *Advances in Cryptology – ASIACRYPT 2014 – 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7–11, 2014. Proceedings, Part I*, volume 8873 of *Lecture Notes in Computer Science*, pages 282–296. Springer, 2014.

[XPR+22] Zhuang Xu, Owen Pemberton, Sujoy Sinha Roy, David F. Oswald, Wang Yao, and Zhiming Zheng. Magnifying side-channel leakage of lattice-based cryptosystems with chosen ciphertexts: The case study of Kyber. *IEEE Trans. Computers*, 71(9):2163–2176, 2022.

[XWW+24] Wenwen Xia, Leizhang Wang, Geng Wang, Dawu Gu, and Baocang Wang. A refined hardness estimation of LWE in two-step mode. In *Public-Key Cryptography – PKC 2024 – 27th IACR International Conference on Practice and Theory of Public-Key Cryptography, Sydney, NSW, Australia, April 15–17, 2024, Proceedings, Part III*, volume 14603 of *Lecture Notes in Computer Science*, pages 3–35. Springer, 2024.

[YK20] Shih-Chun You and Markus G. Kuhn. A template attack to reconstruct the input of SHA-3 on an 8-bit device. In *Constructive Side-Channel Analysis and Secure Design – 11th International Workshop, COSADE 2020, Lugano, Switzerland, April 1–3, 2020, Revised Selected Papers*, volume 12244 of *Lecture Notes in Computer Science*, pages 25–42. Springer, 2020. https://doi.org/10.1007/978-3-030-68773-1_2.

[YK21] Shih-Chun You and Markus G. Kuhn. Single-trace fragment template attack on a 32-bit implementation of Keccak. In *Smart Card Research and Advanced Applications – 20th International Conference, CARDIS 2021, Lübeck, Germany, November 11–12, 2021, Revised Selected Papers*, volume 13173 of *Lecture Notes in Computer Science*, pages 3–23. Springer, 2021. https://doi.org/10.1007/978-3-030-97348-3_1.

A EM Adjustment Algorithms

Algorithm 9 GMM Normalization (for each LDA dimension)

Input: $\Theta = \left[(\pi_k, \boldsymbol{\mu}_k \in \mathbb{R}^d, \boldsymbol{\Sigma}_k \in \mathbb{R}^{d\times d}) \mid k \in K\right]$
Input: $\boldsymbol{T}_{\mathrm{a,LDA}} = [\boldsymbol{t}_1, \boldsymbol{t}_2, \ldots, \boldsymbol{t}_N] \in \mathbb{R}^{d\times N}$
Output: $\Theta' = \left[(\pi_k, \boldsymbol{\mu}'_k, \boldsymbol{\Sigma}_k) \mid k \in K\right]$

for i **from** 1 **to** d **do**
  $\overline{\mu_i} = \sum_{k\in K} \pi_k \mu_{k,i}$
  $\sigma_i = \left[\sum_{k\in K} \pi_k \left((\mu_{k,i} - \overline{\mu_i})^2 + \boldsymbol{\Sigma}_k[i,i]\right)\right]^{1/2}$
  $\overline{t_{\mathrm{a},i}} = N^{-1} \sum_{j=1}^{N} t_{i,j}$
  $\sigma_{\mathrm{a},i} = \left[(N-1)^{-1} \sum_{j=1}^{N} (t_{i,j} - \overline{t_{\mathrm{a},i}})^2\right]^{1/2}$
  for all $k \in K$ **do**
    $\mu'_{k,i} = (\mu_{k,i} - \overline{\mu_i})\sigma_{\mathrm{a},i}/\sigma_i + \overline{t_{\mathrm{a},i}}$

Algorithm 10 GMM Merge Components

Input: $\Theta = \left[(\pi_k, \boldsymbol{\mu}_k \in \mathbb{R}^d, \boldsymbol{\Sigma}_k \in \mathbb{R}^{d\times d}) \mid k \in K\right]$
Input: N_{GMM}, p_{value}
Output: $\Theta^{(0)}$, $\mathit{labels}_{\mathrm{idx}}$

$\Theta^{(0)} = [\,]$; $\mathit{labels}_{\mathrm{idx}} = [\,]$
for $k \in K$ **do**
  $\mathit{merged} =$ `False`
  $\theta_k = (\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$
  for $(\pi_i, \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) \in \Theta^{(0)}$ **do**
    $\theta_i = (\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$
    if $\mathrm{pvalue}(\mathrm{HotellingT^2Test}(\theta_k, \theta_i, N_{\mathrm{GMM}})) > p_{\mathrm{value}}$ **then**
      $\pi_i = \pi_i + \pi_k$
      $\mathit{labels}_{\mathrm{idx}}[k] = (i, \pi_k)$
      $\mathit{merged} =$ `True`
      break
  if not merged **then**
    $\mathrm{push}(\Theta^{(0)}, (\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k))$
    $\mathit{labels}_{\mathrm{idx}}[k] = (\mathrm{length}(\Theta^{(0)}), \pi_k)$ ▷ last index of $\Theta^{(0)}$

Algorithm 11 EM process (fixed weight)

Input: $\Theta^{(l-1)} = \big[(\pi_k, \boldsymbol{\mu}_k^{(l-1)} \in \mathbb{R}^d, \boldsymbol{\Sigma}_k^{(l-1)} \in \mathbb{R}^{d\times d}) \mid k \in K\big]$
Input: $\boldsymbol{T}_{\text{a,LDA}} = [\boldsymbol{t}_1, \boldsymbol{t}_2, \ldots, \boldsymbol{t}_N] \in \mathbb{R}^{d\times N}$
Output: $\Theta^{(l)} = \big[(\pi_k, \boldsymbol{\mu}_k^{(l)}, \boldsymbol{\Sigma}_k^{(l)}) \mid k \in K\big]$

 # E-step

for i **from** 1 **to** N **do**
  for $k \in K$ **do**
    $r_{k,i}^{(l)} = \dfrac{\pi_k p(\boldsymbol{t}_i \mid \boldsymbol{\mu}_k^{(l-1)}, \boldsymbol{\Sigma}_k^{(l-1)})}{\sum_{k'\in K} \pi_{k'} p(\boldsymbol{t}_i \mid \boldsymbol{\mu}_{k'}^{(l-1)}, \boldsymbol{\Sigma}_{k'}^{(l-1)})}$

 # M-step

for $k \in K$ **do**
  $r_{k,\text{sum}} = \sum_{i=1}^{N} r_{k,i}^{(l)}$
  $\boldsymbol{\mu}_k^{(l)} = \big(\sum_{i=1}^{N} r_{k,i}^{(l)} \boldsymbol{t}_i\big)/r_{k,\text{sum}}$
  $\boldsymbol{\Sigma}_k^{(l)} = \big(\sum_{i=1}^{N} r_{k,i}^{(l)} (\boldsymbol{t}_i - \boldsymbol{\mu}_k^{(l)})(\boldsymbol{t}_i - \boldsymbol{\mu}_k^{(l)})^\top\big)/r_{k,\text{sum}}$

Algorithm 12 GMM Unmerge Components

Input: Θ, $labels_{\text{idx}}$
Output: $\Theta_{\text{adj}} = \big[(\pi_k, \boldsymbol{\mu}_k \in \mathbb{R}^d, \boldsymbol{\Sigma}_k \in \mathbb{R}^{d\times d}) \mid k \in K\big]$

for $k \in K$ **do**
  $(i, \pi_k) = labels_{\text{idx}}[k]$ ▷ restore π_k
  $(\pi_i, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = \Theta[i]$ ▷ ignore π_i
  $\text{push}\big(\Theta_{\text{adj}}, (\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)\big)$