Generic Accelerators for Costly-to-Mask PQC Components

Markus Krausz*[1], Georg Land*[1], Florian Stolz*[1], Dennis Naujoks†[2], Jan Richter-Brockmann[1], Tim Güneysu[1,3] and Lucie Kogelheide†[4]

[1] Ruhr-University, Bochum, Germany, firstname.lastname@rub.de, mail@georg.land
[2] ETAS GmbH, Bochum, Germany dennis.naujoks@etas.com
[3] DFKI GmbH, Bremen, Germany
[4] BWI GmbH, Bonn, Germany, lucie.kogelheide@bwi.de

Abstract. In this work, we examine widespread components of various Post-Quantum Cryptography (PQC) schemes that exhibit disproportionately high overhead when implemented in software in a side-channel secure manner: fixed-weight polynomial sampling, Cumulative Distribution Table (CDT) sampling, and rotation of polynomials by a secret offset. These components are deployed in a range of lattice-based and code-based Key Encapsulation Mechanisms (KEMs) and signature schemes from NIST's fourth round of PQC standardization and the signature on-ramp. Masking – to defend against power Side-Channel Analysis (SCA) – on top of required constant-time methods, leads in some of these cases to impractical runtimes. To solve this issue, we start by identifying a small set of core operations, which are crucial for the performance of all three components. We accelerate these operations with an Instruction Set Extension (ISE) featuring masked instructions, which are generic and low-level and can be used in a wide range of cryptographic applications and thereby tackle performance, microarchitectural power leakage, and cryptographic agility, simultaneously. We implement dedicated masked instructions for our core operations as an add-on to the RISC-V core by Gao et *al.* [GGM+21] which features masked instructions for Boolean and arithmetic operations and evaluate several algorithmic approaches in standard and bitsliced implementations on different ISE constellations. Our instructions allow some masked components to run more than one order of magnitude faster and are first-order power side-channel secure, which our practical evaluation confirms.

Keywords: PQC, Fixed Weight Polynomial Sampling, SCA, Masking, RISC-V, BIKE, Frodo, HQC

1 Introduction

Nowadays, embedded devices are interconnected via a broad variety of communication channels. In case that sensitive data is transmitted via these channels, the communication should be encrypted. However – although the chosen algorithms are considered secure – *implementations* of cryptographic algorithms are generally vulnerable to side-channel attacks on any platform. For example, an adversary can exploit secret-dependent runtime differences of the algorithms caused by memory accesses, branching or instructions with data-dependent runtime. Moreover, having physical access to the target device allows the adversary to measure the device's power consumption directly or indirectly via its electromagnetic emission, thereby gaining information on secret key material. Especially

*These authors contributed equally to this work.
†The respective work has been conducted at TUV Informationstechnik GmbH.

power Side-Channel Analysis (SCA) recently became more and more accessible due to simple and inexpensive measurement setups [Inc].

Researchers from academia and industry proposed many approaches to counteract these attack vectors. In order to prevent timing attacks, the so-called constant-time programming paradigm [Por] emerged as a generic defense mechanism. The most promising countermeasure against SCA is masking [PR13], splitting sensitive data into secret shares. Consequently, underlying operations have to be transformed in order to be able to perform these operations on shares.

However, applying masking techniques to implementations of novel cryptographic schemes can raise new challenges. This includes novel schemes that are designed to be secure against attacks mounted on quantum computers.

A fact that rarely expressed explicitly is that a correctly masked implementation is necessarily also secure against timing side-channels, because secret-dependent branches and memory accesses cannot be masked and thus must be secured using constant-time methods. Additionally, instructions with operand-dependent runtime can usually not be performed with masked values. Protection against timing attacks with constant-time implementations is often achieved by always executing with worst-case runtime and applying dummy memory accesses to all possible locations instead of only accessing the secret-dependent value. Secret-dependent memory accesses are relevant for timing side-channels because the state of the cache can influence the runtime of the cryptographic implementation or the runtime of an adversarial program which accesses a shared cache. In order to achieve protection against power SCA, these extra operations need to be masked as well introducing additional overhead.

For example, as shown by Krausz et *al.* [KLRBG23], protecting fixed-weight polynomial samplers of common Post-Quantum Cryptography (PQC) schemes like BIKE, HQC, NTRU and many others with masking introduces a huge overhead with respect to the runtime and randomness requirements of the design. One reason for this huge overhead lies in the simultaneous protection against timing and power side channels.

A second example can be found in the implementation techniques for the samplers of FrodoKEM [BCD+16] and the recently presented schemes Hawk [DPPvW22] and Haetae [CCD+23]. Here, the common approach to achieve constant-time implementations is to compare a uniformly random bit string with entries in a Cumulative Distribution Table (CDT). Again, adding protection mechanisms based on masking techniques on top introduces a huge overhead.

Another scenario where protection is costly can be observed in the portable [DGK] and optimized implementations [CCK21] of BIKE [ABB+22]. The decoding algorithm of BIKE requires to rotate the syndrome based on the non-zero indices of the private key. Without protection against timing and power side-channel attacks, an adversary can easily extract information about the private key.

In this work we identify a small set of performance-critical core operations, relevant for all three previously mentioned use-cases. These operations are the conditional move (cmov) and integer comparisons (cmp), to be more precise: comparison for equality (cmpeq) and greater-than comparison (cmpgt).

We accelerate these core operations in hardware to reach practical performance for masked post-quantum cryptography in software. Note that these core operations are not scheme specific, but relevant to a wide range of post-quantum digital signature procedures and Key Encapsulation Mechanisms (KEMs). Our accelerated core operations are generic enough to be found in many more applications within and outside the domain of post-quantum cryptography. For example, the message-to-polynomial conversion in Kyber, which has also been targeted by a SCA [WBD23] could also make use of our accelerators. Therefore, in contrast to prior masked accelerators [FBR+22], our solution allows for more crypto-agility.

Besides acceleration, masked hardware instructions also tackle the issue of microarchitectural power leakage [MPW22], a problem of masked software implementations, which can be secure from a source code perspective, but still exhibit side-channel leakage due

to unforeseeable combinations of shares induced by microarchitectural behavior. In this context, Gao et *al.* [GGM+21] presented a RISC-V Instruction Set Extension (ISE) with masked Boolean and arithmetic instructions that tackles microarchitectural leakage and performance overheads induced by masking. Furthermore, they evaluated its performance impact on symmetric cryptography.

Our identified core operations can be performed efficiently with short instruction sequences from that core. Additionally, because our selected core operations are very low level, we decided to integrate them as additional masked instructions into their core to allow for a direct comparison.

Contribution. To summarize the contributions of this work, we achieve for the first time practical performance for constant-time and power side-channel resistant PQC components by relying on a masked ISE for RISC-V. To motivate the necessity of a masked syndrome rotation, we present a power SCA on BIKE, developed independently of the concurrent work [CARG23]. Moreover, we identify core operations (cmov, cmp) required in multiple critical PQC components across a wide range of PQC schemes. Additionally, we extend the RISC-V core presented in [GGM+21] with masked instructions for these operations. Besides the performance issue, we also address microarchitectural power leakage and crypto-agility with our approach.

To evaluate the impact of accelerating the core operations, we extend the bitsliced fixed-weight polynomial samplers from [KLRBG23] with non-bitsliced variants to allow a fair and in-depth comparison. Additionally, we provide the first masked software implementation for CDT samplers and syndrome rotation in BIKE. We confirm side-channel security of our implementations by performing t-test evaluations on our additional ISE and an exemplary sampling algorithm. We perform extensive benchmarks with multiple parameters to demonstrate the performance impact of the accelerated core operations and the newly introduced masked instructions. Our source code for software and hardware will be publicly available at https://github.com/Chair-for-Security-Engineering/pqc-masking-ise.

Related Work. Following [GGM+21], our masked accelerators so far only implement first-order masking. Marshall et al. [MP21] investigate how higher masking orders can be realized, utilizing RISC-Vs vector ISE. Note that the concepts for our masked instructions are also suitable for higher-order masking. Cheng et al. [CP23] also work with an ISE to address power side-channel leakage, but instead of accelerating instructions by computing on multiple shares, they only focus on the microarchitectural leakage aspect.

Accelerators for masked PQC implementations have been presented by Fritzmann et al. [FBR+22]. In contrast to our work, they mainly implement higher level functionalities, nevertheless they include a masked adder for Boolean shared values, which is also featured in the work we based on [GGM+21]. While Fritzmann et al. focus on improving the execution time of the PQC schemes Kyber and Saber, their masked Keccak accelerator is also highly relevant for a wide range of PQC schemes. For the signature generation in the reference implementation of Haetae, reportedly around 60 percent of the computation time is spent with Keccak. A comparison of their masked Keccak accelerator versus a masked Keccak based on the low-level instruction that we use would be an interesting target for future work. Currently, the different target processors prohibit a direct comparison.

2 Applications

In this work, we focus on three different PQC components: fixed-weight polynomial sampling, CDT sampling and syndrome rotation. All three components have in common that unprotected implementations are susceptible to SCA as shown in prior work and demonstrated in this section. The next commonality is that masked software implementations for these algorithms unfortunately suffer from extraordinarily high performance

overheads. Fortunately, they can all significantly be accelerated by the same small set of core operations.

2.1 Fixed-Weight Polynomial Sampling

A wide range of PQC schemes requires fixed-weight polynomial samplers to randomly generate a binary or ternary polynomial with a fixed length N and a fixed number of non-zero coefficients (weight) W from a uniform distribution.

Algorithms. The known algorithms for this problem can be divided into three categories. The index rejection method and its bounded variant [ND22] sample the non-zero coefficients with rejection sampling. Another approach aims at generating random bitstrings, this direction is followed by the Comparison method [KLRBG23]. Lastly, two algorithms start with fixed polynomials with the correct length N and weight W and then apply shuffling with a Fisher-Yates variant [Sen21] or sorting [Ber22]. Notably, these algorithms generate polynomials in one of two possible representations: the index representation stores indices to non-zero coefficients whereas the coefficient representations stores the plain coefficients. The index representation allows for a denser representation of polynomials with a low W/N ratio. Ultimately, the implementation of the operations on the polynomial following the sampling determine which representation is required, usually the coefficient representation. A conversion from index to coefficient representation and vice versa is straightforward but costly, and mainly depends on a cmov operation.

Applications. Fixed-weight polynomial sampling is used in BIKE, HQC, McEliece, NTRU, Streamlined NTRU Prime, and NTRU LPRrime well-known from NIST's standardization process and also SMAUG [CCHY23], which was recently submitted to the Korean Post-Quantum Cryptography Competition. Moreover, we expect that several code-based schemes for NIST's signature on-ramp feature fixed-weight-sampling procedures. While requirements and parameters slightly differ between the schemes, the sampling problem at its heart is the same. Timing side-channel attacks on the samplers of BIKE and HQC have already been demonstrated [KAA21,GHJ+22,Sen21] and without any dedicated protection, similar attacks could be performed based on information gained from power side-channel attacks.

Masking. A recent work [KLRBG23] compares the performance of all known fixed-weight polynomial sampling algorithms as masked software implementations. The benchmarks show that the performance of the algorithms clearly varies depending on the parameters N and W, and the preference for different algorithms depends on the individual use case. However, all reported runtimes are far from satisfying. The situation is worst for BIKE and HQC, where the sampling is required during decapsulation. The most efficient algorithms cannot be applied here for a side-channel secure implementation, because they inherently leak the secret seed for the Pseudorandom Number Generator (PRNG) via their timing behavior (cf. [GHJ+22]). For these cases, the most efficient algorithms require millions to hundreds of million cycles on a Cortex-M4, depending on the polynomial representation form. An efficient approach for a masked implementation has been presented earlier [CGTZ23], but as [KLRBG23] already points out it is likely susceptible to a horizontal SCA [BCPZ16].

The most performance critical operation for almost all sampling algorithms is either a masked cmov, a masked cmpeq, or a masked cmpgt. In Section 3.1 we explain, why exactly these operations are so crucial. Sometimes even more than one of these operations is required in a nested loop. Accelerating these operations with dedicated hardware instructions therefore can greatly improve the sampling performance and lead to practical runtimes.

2.2 Sampling from a Discrete Gaussian Distribution

As mentioned above, discrete Gaussian sampling is essential for many lattice-based cryptographic schemes [DDLL13, BCD+16, GPV08]. Hence, efficient and secure implementations of these sampling algorithms are important for reliable modern cryptography. In order to discuss and highlight crucial parts, we denote the discrete Gaussian distribution with variance σ^2 as $\mathcal{N}_{\mathbb{Z}}(\sigma^2)$. Then, $\mathcal{N}^+_{\mathbb{Z}}(\sigma^2)$ is defined to have the same distribution as $\mathcal{N}_{\mathbb{Z}}(\sigma^2)$ for all samples in $\mathbb{N}$, half the probability for sampling zero and probability zero for sampling negative values.

Algorithm. There are multiple well-studied algorithms to perform sampling from discrete Gaussian distribution [DN12, BCG+14, DDLL13]. The most common approach is CDT sampling [Pei10], where a uniform random bit string is compared with the entries in a pre-computed Cumulative Distribution Table. This CDT approximates $\mathcal{N}^+_{\mathbb{Z}}(\sigma^2)$ for a given precision and σ^2. For a random bit string r, the output sample is the index i in the table T, for which $T_i < r \geq T_{i+1}$. The optimal way to achieve this in terms of memory access and number of comparisons is an adapted binary search that takes into account the distribution itself. Unfortunately, this opens a (cache-)timing side channel, because the memory access pattern would depend on the sampled value. Thus, to avoid branches and secret-depending memory accesses, the random bit string must be compared against each entry in the table. Then, the comparison result bits are accumulated to obtain the sample, which is mapped to $\mathcal{N}_{\mathbb{Z}}(\sigma^2)$ by conditional negation with another random bit.

Applications. The most notable application of CDT sampling is FrodoKEM, where it is used for sampling error terms during key generation, encapsulation and decapsulation. In particular, the procedure is performed deterministically during decapsulation in the re-encryption step, because of the Fujisaki-Okamoto transform. Obtaining (side-channel) information about these errors may result in a recovery of the shared secret or the private key. Notably, a practical attack on FrodoKEM was shown recently [MKK+23]. Thus, the sampling operation requires masking to comprehensively secure FrodoKEM.

Another important example is Hawk [DPPvW22], a PQC signature scheme based on the newly proposed lattice isomorphism problem. In contrast to prior work, it does not require floating-point arithmetic which would prohibit a masked implementation. Furthermore, the authors claim that the only missing part to enable a fully masked implementation is sampling discrete Gaussian variates from a CDT.

Finally, the recently presented scheme Haetae [CCD+23], which is similar to Dilithium, samples the signature nonce $\mathbf{y}$ from a Gaussian distribution. Again, CDT sampling is deployed for that purpose. For a high-assurance version of its deterministic variant, this sampler must be protected accordingly using masking countermeasures.

Masking. The basic operation within the sampler is the comparison of a *secret bit string* with a *public constant*. A standard software approach to achieve this is bitslicing the inputs and then using the optimized comparison as described in [KLRBG23]. However, bitslicing transformations also cause an overhead and may lead to additional microarchitectural leakage.

A non-bitsliced implementation with hardened instructions may yield better overall results. For a masked comparison, masked subtraction and shift operations as provided by [GGM+21] are sufficient. However, a dedicated masked greater-than comparison instruction could further improve the performance while inducing only a miniscule area overhead in the core.

2.3 BIKE Syndrome Rotation

For our third example, we investigate the side-channel resistance of BIKE which is one of the three remaining KEM candidates in NISTS's PQC standardization process. In

Algorithm 1 Word-unit rotation used in BIKE's decapsulation.

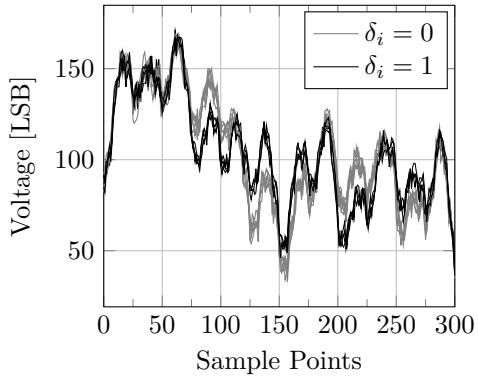
Data: Vector $v = \{s, s, s\}$ holding three consecutive copies of s, secret index δ
Result: Rotated syndrome s

function WORD_UNIT_ROTATION(v, δ)
  $i \leftarrow \log(r/32/2)$
  while $i \geq 1$ **do**
    if $\delta > i$ **then**
      $m \leftarrow$ `0xFFFFFFFF`
    else
      $m \leftarrow$ `0x00000000`
    end if
    $\delta \leftarrow \delta - (i \,\&\, m)$
    for $j = 0$ **to** $r/32 + i$ **do**
      $v[j] \leftarrow (v[j] \,\&\, \sim m) \mid (v[j + i] \,\&\, m)$
    end for
    $i \leftarrow i >> 1$
  end while
  return Lower r bits of v
end function

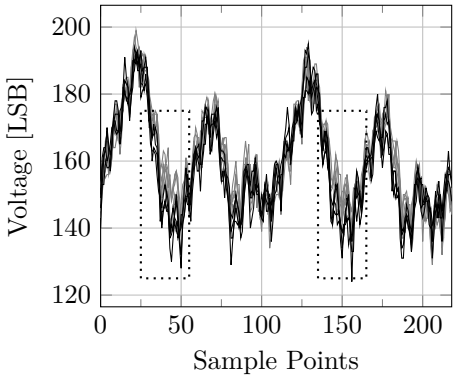
contrast to the numerous lattice-based schemes, BIKE's security foundation relies on linear error-correcting codes. Independently of concurrent work [CARG23], we developed a power side-channel attack on the syndrome rotation of BIKE. In this section, we show that the conditional move operations in BIKE's portable C implementation [DGK] allow to perform a Simple Power Analysis (SPA) leading to a partial secret key recovery even for ephemeral keys. Hence, this attack highlights the requirement for efficient masking techniques for conditional move instructions (cmov).

Algorithm. In the decapsulation of BIKE, the error polynomials, which are intentionally added to the encrypted message in the encapsulation are iteratively decoded by the Black-Grey-Flip (BGF) decoder [DGK20]. In this decoding step, the syndrome s is rotated by the secret indices of the private key (similar to the multiplication presented in [Cho16]). More precisely, these indices are represented by $\lceil \log r \rceil$ bits where r denotes the size of the polynomials used in BIKE. On a 32-bit architecture, the vector storing the syndrome s is divided into $\lceil r/32 \rceil$ chunks. Hence, applying a rotation by k bits (i.e., a secret index of the private key), can be accomplished by performing a *word-unit rotation* by the upper $\lceil \log r \rceil - 5$ bits of k (denoted by δ in the following) and a subsequent *bit-unit rotation* by the lower five bits of k. However, in the presented attack, we only measure the power consumption of the word-unit rotation (i.e., rotation by δ) as differences in the power consumption depending on the applied private key can already be distinguished by using just one single trace.

An implementation of the word-unit rotation, that is not timing side-channel secure, could simply change the pointer to the syndrome, but then subsequent operations on the syndrome would leak the rotation value via their memory access pattern. To this end, we show the constant-time implementation of the word-unit rotation in Algorithm 1. For each bit δ_i in δ, the algorithm computes a mask m which is set to `0xFFFFFFFF` in case the bit is one or to `0x00000000` in case the bit is zero. Afterwards, the implementation loops over the whole syndrome, loads the shifted and the unshifted value and selects on of them based on m. This procedure ensures that both – the rotated and unrotated values are loaded from the memory – and no timing differences with respect to the secret index (i.e., δ) can be observed.



(a) Calculation of the mask m (cf. Algorithm 1, Lines 7-11). Frequent visual differences over the period of the calculation.



(b) Use of the mask m (cd. Algorithm 1, Line 14) in two subsequent iterations. Visual differences are highlighted.

Figure 1: Ten raw power traces (500MS/s) of the word-unit rotation from different secret key indices δ at bit position $i = 13$. The power traces are measured on an ARM Cortex-M4 processor of the FRDM-K22F development board (rev.D) and clocked with 120 MHz. The C code is compiled with `arm-none-eabi-gcc` compiler (v9.2.1) and optimization level `-O3`.

Power Consumption Leakage. While the constant-time implementation protects the rotation against timing attacks, it unfortunately introduces two attack surfaces for side-channel adversaries exploiting the power consumption of the device. First, the computation of the mask m in Lines 6-10 in Algorithm 1 causes variations in the power consumption due to the huge difference of the Hamming weight which is directly connected to the i-th bit of the secret index δ. Second, m is used again in Line 13 to decide whether the rotated or unrotated value of the syndrome is used to update it. Again, due to the huge difference of the Hamming weight of m, it is expected to observe corresponding variations in the power consumption of the target device.

Attacker Model. Since we perform a power side-channel attack on the word-unit rotation, we assume an attacker with physical access to the device. Furthermore, we assume an instantiation of BIKE that works with ephemeral keys as suggested in the specification [ABB+22]. Hence, a side-channel adversary can only acquire the power consumption of one secret key used in the decoding process.

Side-Channel Attack. In our side-channel attack, we first record a power trace of the decoding step of BIKE's decapsulation. Since the BGF decoder always performs seven iterations, each secret index of the private key is used exactly seven times to execute the rotation described in Algorithm 1. In order to achieve cleaner traces, we compute the mean of these seven sub-traces. Now, we select Points of Interest (PoI) by visually inspecting our preprocessed power traces, i.e., the points in the power traces where the mask m is computed or used. The selection of the PoIs is done for each bit position i separately since minor differences can occur in the power traces (e.g., in the horizontal appearance). To recover all W secret indices of one private key at the same time, we apply a *k-means* clustering of all power traces processing the i-th bit of the indices.

Practical Results. We perform the described attack on BIKE's security level $\lambda = 1$ parameter set utilizing only $W = 142$ rotations. As target device, we use an ARM Cortex-M4 processor supplied with a 120 MHz clock. The measurement results for the most significant bit for ten different secret indices δ are shown in Figure 1. The differences between a one and a zero can clearly be distinguished without any further postprocessing of the traces. Repeating the same experiment for the remaining bits of δ would result in

Table 1: Algorithms implemented in this work. The adapted implementations are based on [KLRBG23].

Application	Algorithm	Representation	
		standard	bitsliced
Fixed-Weight Poly. Sampling	Fisher-Yates	this work	adapted
	Sorting	this work	adapted
	Rejection	this work	adapted
	Comparison	this work	adapted
	I2C Conversion	this work	adapted
FrodoKEM Gaussian Sampling	CDT Sampling	this work	this work
BIKE Rotation	Barrel-Shifter	this work	*not applicable*

recovering the upper bits of a secret index k of BIKE's private key. The remaining lower five bits could be recovered by solving linear equation systems.

To demonstrate that our attack works reliably, we measure the power consumption of the word-unit rotation of 1 000 different secret keys. We are able to recover 98.40 % of the partial secret keys without erroneous bits. In general, 99.99 % of all upper secret bits $(8 \cdot 142 \cdot 10^3)$ could be recovered correctly.

Moreover, we want to stress the fact that improvements to the clustering algorithm and feasible template attacks can increase the success rate further, as we only presented a straightforward approach in this section (cf. [CARG23]). Optimized versions [CCK21, CGKT22] of BIKE significantly reduce the Hamming weight, but the attack is still possible in a similar fashion as recently demonstrated in [CARG23].

Masking. Based on a constant-time implementation, the rotation is straightforward to mask, the crucial operation, which generates the leakage is the cmov. First-order masking of the syndrome doubles its memory footprint and thus doubles the cost for loads and store for this already memory instruction intense operation. A masked implementation should therefore be even more focused to be efficient in this regard.

3 Software Implementation

We developed efficient C code for all three applications to evaluate our hardware accelerators. Table 1 provides an overview of the algorithms implemented in this work. We implement four different fixed-weight polynomial sampling algorithms and one conversion algorithm, adapt existing bitsliced variants, and develop a secure CDT sampler for FrodoKEM and a rotation function for BIKE. Our code works for arbitrary masking orders, the masked instructions on the core, however, only support first-order masking.

Bitslicing. For highspeed symmetric cryptographic software implementations, bitslicing [Bih97] has been an important technique for many years. The general idea is to reach better resource utilization by useing the full register width although operating on values bounded by fewer bits – in particular when operating on single bits. Therefore, data has to be transformed to the bitsliced representation. This operation corresponds to a matrix transposition. For example, given a register width of 32 bit, in a bitsliced implementation 32 10-bit values are not stored in 32 registers, instead the i-th bit of each value is aggregated in one register. Therefore, the values are stored in ten registers in total. Operations can then be done with 32 values in parallel, always using the full register width. In theory, a speedup of a factor equal to the register width can be achieved for operations on single-digit values, whenever parallelization is possible. However, transformations from and to the bitsliced representation and sometimes padding introduce overhead.

Bitslicing is very efficient for *masked* software implementations, in particular because it can reduce the number of costly nonlinear operations [BC22, KLRBG23]. However, for hardware accelerators that execute a masked instruction in a single or only few cycles, the overhead induced by bitslicing can be unprofitable in some cases. To allow a fair comparison, we provide standard (non-bitsliced) and bitsliced implementations for all algorithms if applicable.

Masked Instructions. We implement wrapper functions with inline assembly utilizing the masked instructions. Via preprocessor directives, we are able to determine whether our code is compiled without any masked instructions, with the Boolean and arithmetic, masked instructions developed by Gao et *al.* [GGM+21] or additionally with our masked cmov and cmp instructions. From the masked instructions by Gao et *al.*, only a subset is used in our code, always operating on Boolean shares: mask, unmask, not, and, or, xor, slli, srli, add and sub. The rest of the instructions – e.g., instructions to switch from Boolean to arithmetic masking domain and vice versa – do not apply. Without masked instructions, all masked operations are realized by secure gadgets presented by [BC22, KLRBG23], which rely on unmasked instructions. When evaluating the masked instructions by Gao et *al.*, most operations are realized by their respective instructions, and additionally, these instructions are used for building the higher-level gadgets as the cmov for example, where e.g., a masked and and xor is required.

Furthermore, we implement some inner loops with inline assembly to have more control over the memory and register usage. This allows us to use the benefit of reduced register pressure, which comes with the masked instructions leading to fewer memory loads and stores. Nevertheless, the parametrizability of our high-level C code (arbitrary masking order, parameters N and W and different ISEs) leads to some unnecessary memory accesses, which can be avoided with implementations for a specific parameter set.

cmpeq. For the base Instruction Set Architecture (ISA), we use the operation sequence depicted in Algorithm 2, to express a cmpeq. As can be seen there, the performance will be dominated by the five secure or operations. In contrast to this, we use a dedicated operation sequence whenever masked, Boolean instructions are available. In this case, we compute the first operand minus the second one and vice versa, combining the two results with a secure or, which is shown in Algorithm 3. When our masked cmpeq is enabled, we naturally utilize this instruction.

Algorithm 2 cmpeq with base ISA. All Boolean operations are performed with secure gadgets.

Input: Boolean-shared values a, b
Output: Boolean-shared res, 1 if a and b are equal
$\text{res} := a \oplus b$
$\text{res} := \text{res} \vee (res \gg 16)$
$\text{res} := \text{res} \vee (res \gg 8)$
$\text{res} := \text{res} \vee (res \gg 4)$
$\text{res} := \text{res} \vee (res \gg 2)$
return $(\text{res} \vee (res \gg 1)) \oplus 1$

Algorithm 3 cmpeq with ISE by Gao et *al.* All arithmetic operations are performed with the Boolean-masked instructions.

Input: Boolean-shared values a, b
Output: Boolean-shared res, 1 if a and b are equal
$\text{tmp}_0 := a - b$
$\text{tmp}_1 := b - a$
$\text{res} := \text{tmp}_0 \vee \text{tmp}_1$
return $(\text{res} \gg 31) \oplus 1$

cmpgt. We apply a similar handling for the cmpgt operation. For the base ISA, we adapt the optimized comparison from [KLRBG23] to the standard representation. However, this is very slow and usually yields impractical results, which is why a bitsliced implementation is preferable when no ISE is available. For the evaluation case in which a dedicated masked cmpgt instruction is not available, but masked subtraction and shift instructions are, we

can express the comparison efficiently with a subtraction followed by a shift to the right to get the most significant bit. This is only correct when both operands can be represented with at most 31 bit, or in other words, when the most significant bit of both operands is zero. Due to the possible parameter ranges for N and W, this is always the case in our applications.

3.1 Masked Fixed-Weight Polynomial Sampling

We adapt the bitsliced implementations from [KLRBG23] so that they can be compiled with masked instructions. To enable a fair comparison, we additionally implement non-bitsliced variants of the sampling algorithms. In the following, we briefly summarize the algorithms presented in [KLRBG23] and discuss their key instructions.

Fisher-Yates. The side-channel secure Fisher-Yates method has a quadratic runtime in the weight W. In the inner loop, a masked cmov based on the result of a masked cmpeq is executed. Nevertheless, for the standard implementation, we use bitslicing for another single loop, because it requires a 48-bit transformation from the Boolean to the arithmetic domain and vice versa. Although the core by Gao et *al.* has dedicated instructions for these conversions, they only work for 32-bit operands. In contrast, the gadget implemented by Bronchain and Cassiers [BC22] allows (with minor modifications) efficient bitsliced conversions for this size, and thus the costs are amortized.

Sorting. Constant-time sorting can be done with sorting networks. The bitsliced sorting algorithm in [KLRBG23] is based on Batchers's Bitonic mergesort, which is easy to parallelize. For the standard implementation, we opted for djb-sort [BCLv17] instead. Both sorting algorithms have an asymptotic runtime of $\mathcal{O}(N \log^2(N))$, where N in our case is the polynomial length, but djb-sort is capable of reaching a slightly lower constant factor. The fundamental operation in a sorting network is a cmpgt followed by a conditional swap (i.e., two cmov). Although the values to be swapped during sorting cover the full 32-bit registers, only the upper 30 bits consist of the random values that are compared. By shifting the values to the right before the cmpgt, we can use the optimized operation sequence based on subtraction.

Rejection Sampling. Rejection sampling requires a cmpgt to verify whether a sampled bitstring represents a valid index below N. Additionally, a loop over the already sampled indices checks for collisions using cmpeq. We did not implement and evaluate the bounded rejection sampling method, because the Fisher-Yates approach is strictly superior as already pointed out by Krausz et *al.* [KLRBG23].

Comparison Sampling. Comparison sampling sets each coefficient bit individually with probability $p = W/N$. This is implemented by comparing a random l-bit string with the constant $\lfloor 2^l p \rceil$ and setting the coefficient to one if the random value is below the threshold. This obviously heavily relies on a cmpgt operation.

I2C Conversion. Both the Fisher-Yates shuffle and rejection sampling generate polynomials in the index representations. In most cases, however, the coefficient representation is required. The conversion is straightforward: We start from a polynomial in coefficient representation where all coefficients are set to zero. Then, a public counter i runs from 0 to $N-1$ and in each iteration, we check whether i is equal to each coefficient in the list of indices. Depending on this cmpeq result, we cmov a one into the position i in the result polynomial. Thus, the conversion performs exactly $N \cdot W$ cmpeq and cmov.

3.2 Masked FrodoKEM CDT Sampler

To the best of our knowledge, we are the first to report a masked CDT sampler. The most important masked operations are integer greater-than comparison and bit accumulation. For our bitsliced variant, we implement the optimized comparison with a public constant t from [KLRBG23], which only uses $\lceil \log_2 t \rceil$ secure and operations. Bit accumulation can be performed with half adders, adapting the masked implementation from Bronchain and Cassiers [BC22]. This requires up to $\lceil \log_2 |T| \rceil$ secure and operations, where $|T|$ is the number of entries in the CDT.

Our non-bitsliced implementation varies depending on the masked instructions available. Similar to our fixed-weight polynomial sampling algorithms, the operands for the greater-than comparison are bounded (15 bit in this case), which allows us to use different optimal operation sequences. The comparison results are accumulated with a masked addition, thus both operations are executed $|T|$ times for each sample and dominate the computation time.

3.3 Masked BIKE Rotation

A naïve implementation of the polynomial rotation with word granularity in BIKE could be structured very similarly to Algorithm 1. However, this would lead to many unnecessary loads and stores. For example, when the array is rotated by one, the naïve implementation would begin by loading the first and second entries, conditionally move the second to the first and then store both entries back. Then the same procedure is done with the second and third entries, therefore most entries are loaded and stored two times. Following the optimized, unmasked BIKE implementation for the Cortex-M4 [CCK21], the memory accesses can be reduced to one per entry by operating on chains of values and thus keeping an entry in the registers during the cmov to the next entry and the cmov from the previous entry. The rotation obviously relies heavily on the masked cmov.

Bitslicing is not applicable here, because the conditional move already operates on the full register width. Our implementation expects a pointer to three consecutive instances of the masked polynomial. Three instances are required because the number of registers for one polynomial, which is the maximum secret rotation amount r, is not a power of two. As the side-channel secure rotation works bitwise, it covers rotation amounts up to $2^{\lceil \log_2 r \rceil} - 1 > r$, and thus, extends over the second copy. Apart from $\lceil \log_2 r \rceil$ masked shift operations that are required to select the desired bit for each round, the rotation mainly consists of $\lceil \log_2 r \rceil r$ masked cmov operations.

4 Hardware Implementation

We base our hardware implementation on the SCARV CPU[†], an embedded class processor implementing the RISC-V 32-bit base ISA, as well as the multiply and compressed instruction set extensions. Internally, it employs a 5-stage pipeline with forwarding paths to improve performance, but does not feature more advanced features such as branch prediction or a cache subsystem. The design has previously been extended by Marshall et *al.* [MNP+21] to prototype scalar cryptography extensions, by Gao et *al.* [GGM+21] enabling first-order masking via a dedicated ISE and by Marshall et *al.* [MP21], who implement arbitrary-order masking using a separate hardware accelerator.

Because we can express our core operations very efficiently with the given masked operations and to enable a direct comparison with dedicated instructions, we chose to extend the core of Gao et *al.*. Their work is tightly integrated into the core without the need to manage separate register banks, but at the cost of additional engineering effort on the hardware level. They provide Boolean (and, or, ...) and arithmetic (add, sub, ...) instructions for operands masked in the Boolean ($a = a_0 \oplus a_1$, where $\oplus$ represents

[†] https://github.com/scarv

Algorithm 4 cmov Implementation

Data: masked 32-bit values dest, new, cond
Result: new is moved into dest if the least significant bit in cond is 1

function CMOV(dest,new,cond)
  $a \leftarrow \mathsf{dest} \oplus \mathsf{new}$ ▷ share-wise
  $b \leftarrow a \wedge \mathsf{cond}$ ▷ using secure and
  return $b \oplus \mathsf{dest}$ ▷ share-wise
end function

the bitwise addition mod2) and arithmetic domain ($a = a_0 + a_1 \bmod 2^{32}$) as well as conversion between the two domains. The Arithmetic Logic Unit (ALU) is extended with a *masked* ALU module, which implements all necessary primitives. The shares are stored in the general-purpose registers, allowing seamless integration into existing RISC-V code. However, to enable the CPU to load both shares during the register fetch phase, they must be stored in adjacent registers. Additional data paths and logic were added to the forwarding and hazard unit to account for the shares. Furthermore, one of the shares is stored in bit-reversed form preventing accidental combination in other pipeline stages. The original representation is only restored inside the masked ALU.

We add three new instructions for the *dedicated* acceleration of our core operations. This requires us to modify the masked ALU as well as the data paths inside the processor. In the following, we detail our changes. Each paragraph focuses on a specific part of the processor.

General Architecture. RISC-V instructions support up to two source operands and one destination operand. In the SCARV CPU, the destination operand only acts as a pointer to a register which is utilized in the writeback stage. However, we require three operands to implement the cmov instruction. More specifically, we have to use the value of the destination register as a third input to be able to hide the condition. Thus, we have to change the register module to allow the transfer of three operands in their shared representation during the decode and register fetch phase. The decoder will fetch the destination register's value whenever a cmov instruction is detected. We modify the forwarding and hazard unit to account for a third operand and extend the data path to the CPU, however, the third operand is only routed into the masked ALU. Accidental share combinations are again prevented by loading and storing a bit-reversed representation.

Masked ALU. Our new instructions cmov and cmpeq are implemented as additional units, while cmpgt is an extension to an existing unit inside the masked ALU. The cmov module receives three input operands in their shared representation: the destination register's contents, a value and a 1-bit condition, which decides whether the value is moved into the destination register or not. Internally, the 1-bit condition is expanded to 32-bits. The result is computed according to Algorithm 4.

Notably, the cmov operation completes within a single clock cycle, as we reuse the Domain-Oriented Masking (DOM) and gadget provided by Gao et *al.*, which latches its input on the positive edge and adds randomness on the negative edge.

The cmpeq instruction generates a 1-bit condition based on the equality of two input operands and is implemented as its own subunit inside the masked ALU. It checks for equality by first xoring both operands and afterwards, the result is propagated through an or-tree, which at each stage computes the inclusive or of adjacent bits within the intermediate result. Thus, each stage compresses the input by half of its bit length. Therefore, a 32-bit input will be compressed to just a single bit after five stages. If the inputs are equal, the input to the or-tree will be zero and the final output will be zero. Vice versa, if they are unequal, the input will at least contain one digit which is not zero

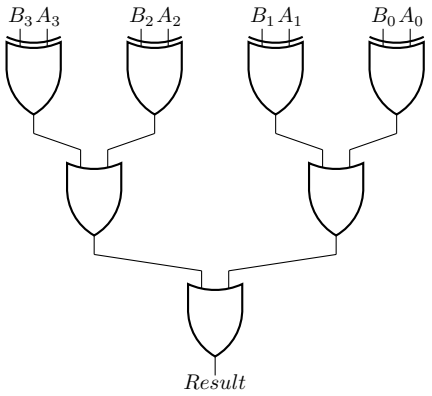


Figure 2: Simplified diagram for a 4-bit cmpeq as included in the extended core.

and the final output will be one. A simplified, non-masked, example of this scheme can be seen in Figure 2. We again reuse the DOM and gadget to build the required inclusive or gate. Because the protected and gate can compute its result in just one cycle, the overall latency of this instruction is five cycles.

Table 2: Encoding for the additional, masked instructions.

31 - 25	24 - 20	19 - 15	14 - 12	11 - 7	6 - 0	Instruction
0x7c	rs2 (condition)	rs1	0x0	rd	0x5b	mask.b.cmov
0x7c	rs2	rs1	0x1	rd	0x5b	mask.b.cmpeq
0x7c	rs2	rs1	0x2	rd	0x5b	mask.b.cmpgt

Lastly, the cmpgt operation compares two input values and sets a 1-bit condition based on which of the two inputs was greater. Unlike the previous instructions, this operation is implemented as a minor extension to the existing bit arithmetic module, which provides masked addition as well as subtraction over Boolean shares. When subtracting a larger value from a smaller value, a carry bit is generated, which is currently unused and not extracted from the Kogge-Stone adder present in the design. Therefore, we modify the adder to output the carry bit in its shared representation, which acts as our 1-bit condition. The adder computes the result of a 32-bit subtraction within 6 cycles, thus cmpgt exhibits the same performance overhead.

Toolchain. We use the modified GCC toolchain from the SCARV project, which adds assembler support for their masking extension. Thus, their instructions can be manually inserted into inline assembly code, but the GCC compiler cannot automatically emit them for high-level code. Our work uses the same opcode space as the original masking extension, however, as it uses all funct3 values available, we choose a new funct7 code in order to create space for our new instructions. cmov, cmpeq and cmpgt are encoded as RISC-V 32-bit R-Type instructions as shown in Table 2. cmpeq and cmpgt perform their comparison on the rs registers and store their result in rd. cmov uses rs2 as the condition whereas rs1 represents the value, which is potentially moved into rd.

5 Evaluation

We evaluate our modified SCARV Core using the ChipWhisperer CW305 FPGA board, which features an Artix xc7a100tftg256 FPGA as well as connections to supply power from a separate power supply and to read out the power consumption. Furthermore, we use the SCARV SoC project to connect the Central Processing Unit (CPU) to an Advanced

extensible Interface (AXI) providing access to memory, peripherals such as serial and General Purpose Input/Output (GPIO), thus allowing us to send inputs to the system as well as setting up precise triggers for our measurements. The CPU has access to 256 kB of shared RAM/ROM and runs at 25 MHz. In the following, we present our results regarding hardware overhead, software performance and side-channel measurements.

Table 3: Field-Programmable Gate Array (FPGA) performance results

		Area			Max. Freq.
		[LUTs]	[FFs]	[Slices]	[MHz]
Masked ISE	Bool.,arith.	7203	2670	2324	29.29
	cmov	128	64	52	
	cmpeq	127	116	81	
	cmpgt	64	8	35	
	other	655	64	137	
	combined	8177	2922	2629	26.93
	Overhead	13.5 %	9.4 %	13.1 %	-8.1 %

5.1 Hardware

Table 3 shows that the hardware overhead of the adapted instructions is moderate. Overall, there is an increase of approximately 13 % in combinatorial logic and 10 % in registers. For the maximum frequencies, we note that we did not perform iterative synthesizing to optimize the frequency, but rather set the target to 25 MHz. The maximum frequency from Table 3 is then computed from the reported worst negative slack.

We note that SCARV is a very simple core, without a cache subsystem or floating point unit. Gao et *al.* report a frequency drop of 5 MHz as well as a 80 % LUT and a 25 % FF overhead for their masked ISE. Thus, when adding our extension on top of their modified core, the overhead is overall moderate.

The area costs in Table 3 that are not directly associated with one of our three instructions, but rather introduced as *other*, mainly come from the extended data path for a third operand, which is necessary for our cmov instruction.

5.2 Software

Figures 4, 5 and 6 show our software benchmark results for the three different applications. Note that for each use case, the generation of input randomness is not benchmarked, and may take another significant part of the cycle counts. This is especially the case if a masked Keccak is used (e.g., as Extendable-output Function (XOF)).

Fix-weight Sampling. Table 4 shows our benchmark results for all fix-weight sampling algorithms applied to a representative selection of parameter sets. For the BIKE decapsulation, the last parameter set, only Fisher-Yates and sorting are secure algorithms, because their runtime is independent of the randomness, which is extended from a secret value in this case. In general, our results show that bitslicing improves the performance for all algorithms, when no accelerators are available.

Fisher-Yates and rejection sampling generate polynomials in index representation, sorting and the comparison method generate polynomials in coefficient representation. In most applications, subsequent operations on the polynomials require the coefficient representation. For these applications, comparison sampling is superior, when implementing for the base ISA, confirming the results from Krausz et *al.* [KLRBG23]. When index representation is required, rejection sampling is clearly faster. For the fourth use case, where N is 12323, rejection sampling combined with the conversion is even faster than the comparison method.

Table 4: Fixed-Weight polynomial sampling performance with different parameters and ISEs in kilo cycles. For each parameter set, we underline the cycle count – for base ISA and masked ISA separately – of the fastest algorithm with output in coefficient representation, taking the index-to-coefficient (I2C) conversion into account for Fisher-Yates and Rejection sampling. Sorting requires too much memory with the BIKE decapsulation parameters on our evaluation platform.

Parameter set	Algorithm	Repr.	Base ISA	Masked ISE	
				Bool., arith.	all
NTRU key gen. $N = 677$, $W = 254$	Fisher-Yates	std.	44 729	2 059	1 674
		bs.	6 876	6 455	6 455
	Sorting	std.	25 017	944	850
		bs.	13 678	4 569	4 569
	Rejection	std.	4 144	156	138
		bs.	1 214	909	909
	Comparison	std.	73 289	1 137	1 105
		bs.	2 243	1 010	1 010
	I2C Conversion	std.	236 071	9 312	5 871
		bs.	5 605	4 273	4 273
sNTRU Prime key gen. $N = 953$, $W = 396$	Fisher-Yates	std.	108 352	4 490	3 551
		bs.	12 112	11 108	11 108
	Sorting	std.	38 289	1 433	1 288
		bs.	19 687	6 509	6 509
	Rejection	std.	6 741	247	231
		bs.	2 660	1 927	1 926
	Comparison	std.	147 623	2 305	2 260
		bs.	4 791	2 124	2 124
	I2C Conversion	std.	517 980	20 415	12 866
		bs.	11 186	8 348	8 348
McEliece encaps. $N = 6688$, $W = 128$	Fisher-Yates	std.	11 474	680	583
		bs.	3 244	3 082	3 082
	Sorting	std.	446 669	16 191	14 479
		bs.	217 048	66 040	66 040
	Rejection	std.	2 073	77	70
		bs.	402	311	312
	Comparison	std.	661 901	10 426	10 223
		bs.	24 173	9 705	9 732
	I2C Conversion	std.	1 175 949	46 506	29 353
		bs.	27 020	18 765	18 765
BIKE key gen. $N = 12323$, $W = 71$	Fisher-Yates	std.	3 651	351	321
		bs.	1 854	1 762	1 762
	Sorting	std.	941 111	33 854	30 231
		bs.	460 872	136 838	136 838
	Rejection	std.	1 554	58	52
		bs.	187	152	152
	Comparison	std.	899 499	14 164	13 908
		bs.	34 208	13 269	13 269
	I2C Conversion	std.	1 203 042	47 761	30 203
		bs.	29 568	20 529	20 529
BIKE decaps. $N = 49318$, $W = 199$	Fisher-Yates	std.	27 594	1 431	1 195
		bs.	5 990	5 518	5 518
	Sorting	std.	—	—	—
		bs.	—	—	—
	I2C Conversion	std.	13 475 741	532 081	335 510
		bs.	360 683	243 527	243 527

Because sorting requires more than the available 256 kB of RAM, for the BIKE decapsulation use case only Fisher-Yates remains as a suitable algorithm. To get the required coefficient representation, the costs of the index-to-coefficient conversion lead to unpractical cycle counts of over 360 million cycles.

The picture changes distinctly when we utilize masked instructions. With the ISE, the non-bitsliced standard implementations are faster for Fisher-Yates, sorting and rejection sampling, as they get accelerated in many cases by a factor of 30.

Due to the speed-up of more than an order of magnitude, sorting becomes fastest for small N. With greater values for N, comparison sampling is again faster. For the index representation, rejection sampling remains the preferable method. The advantage of the complete set of masked instructions compared to reduced set is in the range of about 10 to 20 percent.

Although Fisher-Yates is more than five times faster with the ISE compared to the bitsliced variant for the base ISA, the combined costs remain very high at more than 240 million cycles, when taking the conversion into account. With more RAM available to be able to run the sorting method, it is expected to be faster than Fisher-Yates plus the I2C conversion. Still, the estimated costs of more than 100 million cycles, would not be practical.

Table 5: FrodoKEM CDT sampler performance in kilo cycles. N is contained in the parameter set name.

Parameter set	Repr.	Base ISA	Masked ISE	
			Bool., arith.	all
FrodoKEM-640	std.	26 787	400	319
	bs.	1 197	244	244
FrodoKEM-976	std.	34 858	530	433
	bs.	1 578	323	323
FrodoKEM-1344	std.	31 499	511	426
	bs.	1 376	272	272

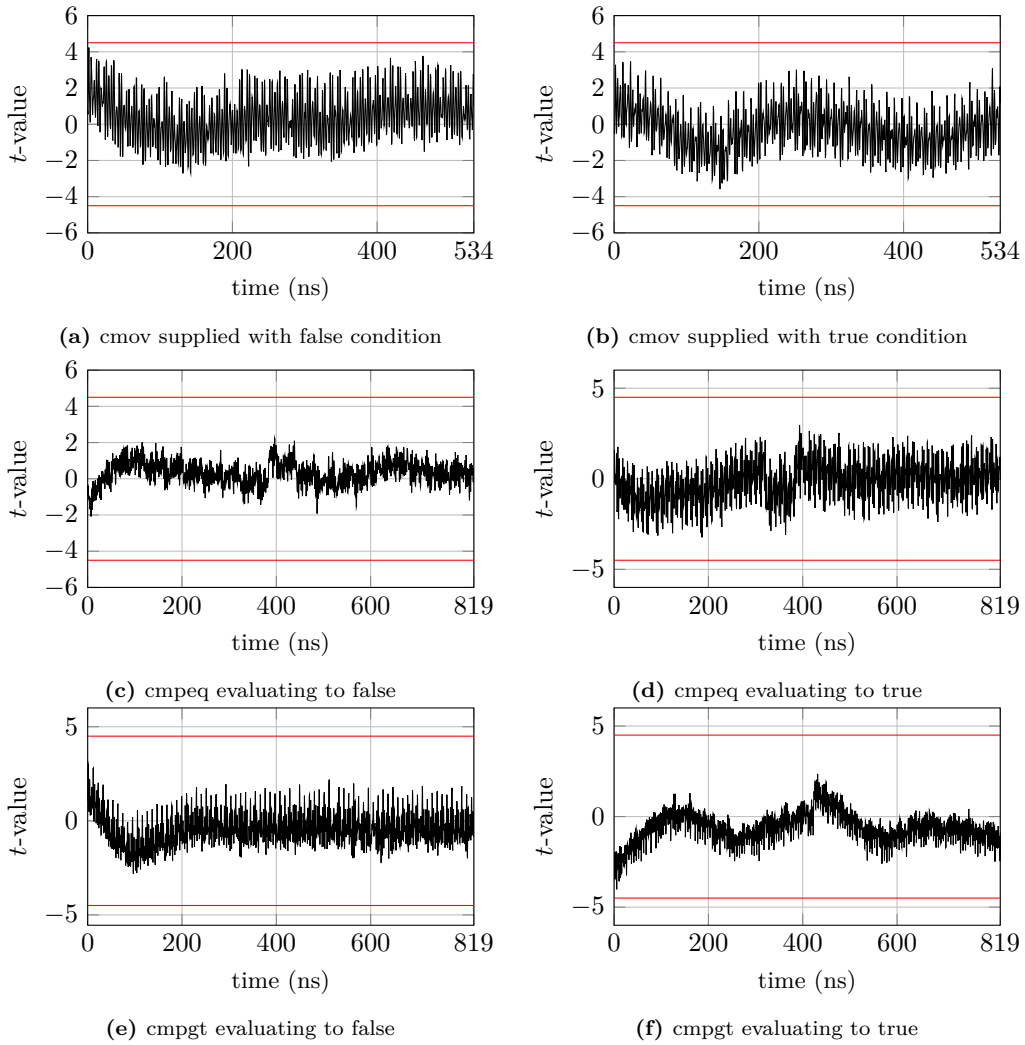
CDT Sampling. Our results for the FrodoKEM CDT sampler can be found in Table 5. Most importantly, the base ISA numbers indicate that side-channel resistant sampling from a CDT is generally feasible, but costly for the FrodoKEM use-case. The bitsliced variants, moreover, are constantly faster than the implementations on standard representation, albeit the featured numbers do not include transformation into and from bitsliced representation. Consequently, this application does not profit from a dedicated acceleration of the core operations, but achieves a significant speed-up with the masked ISE of approximately factor five, lowering the costs from 1197 to 244 thousand cycles for the smallest parameter set.

Table 6: BIKE rotation in kilo cycles.

Parameter set	N	Base ISA	Masked ISE	
			Bool., arith.	all
BIKE-1	12323	283	100	72
BIKE-3	24659	620	217	157
BIKE-5	40973	1 146	340	244

Polynomial Rotation. Table 6 shows the speed-ups that can be achieved for rotation of BIKE polynomials. Again, the base ISA numbers indicate that this operation is generally feasible when implemented with side-channel countermeasures, with cycle counts ranging

from 284 to 1146 thousand cycles. Still, the ISE brings a significant speed-up of about factor three with only the Boolean and arithmetic, masked instructions. With the dedicated, masked cmov instruction the rotation is even four times faster.



(a) cmov supplied with false condition

(b) cmov supplied with true condition

(c) cmpeq evaluating to false

(d) cmpeq evaluating to true

(e) cmpgt evaluating to false

(f) cmpgt evaluating to true

Figure 3: Side-channel evaluation of our instruction set extension using a fixed vs. random t-test with 100,000 measurements.

5.3 Side-Channel Security

We verify first-order power side channel resistance by creating microbenchmarks for each of our new instructions and performing a *fixed vs. random* t-test with 100 000 measurements. The setup consists of an external computer, which masks a fixed or random value based on the result of a coin flip. The values are then transmitted to the SCARV core via a serial connection. Before starting a measurement run, the CPU loads a program, which upon receiving a command either stores transmitted values or runs a microbenchmark. Each microbenchmark consists of an inlined assembly function, which first loads values from memory, then sets a trigger to start the oscilloscope and finally executes a single instruction and resets the trigger. Furthermore, some *NOPs* are inserted before and after setting the trigger to clear the internal pipeline of any remnants. All measurements were performed

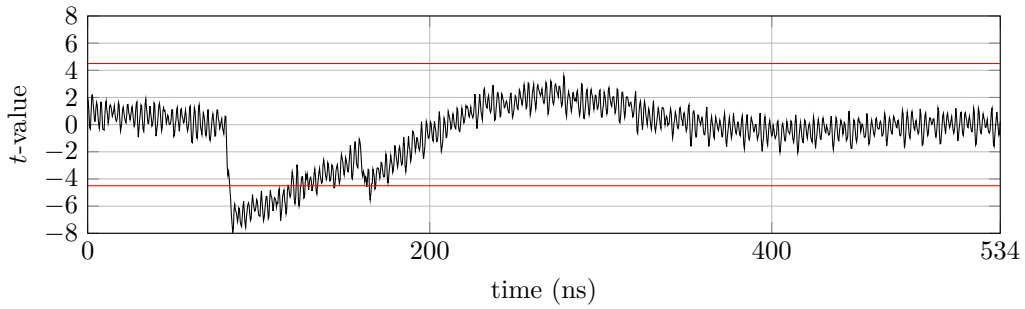


Figure 4: Side-channel measurement setup verification by performing a fixed vs. random t-test with 40,000 measurements of our cmov instruction with the internal randomness engine disabled, causing significant leakage.

using a sample rate of 2.5 GS/s. The results are shown in Figure 3. We performed two t-tests per instruction to ensure that the supplied or generated condition does not leak any information. The results seen in Subfigures (a)-(f) show that our implementation is first-order secure. Additionally, we verified our results by disabling the internal randomness generator for the masked ALU. As seen in Figure 4, without any randomness, the cmov instruction already shows strong signs of leakage after 40 000 measurements.

6 Conclusion

In our work, we show how low-level hardware acceleration for masking can be applied to several subroutines that are highly relevant for a range of PQC schemes. For fixed-weight sampling, this leads to speed-ups of more than an order of magnitude. However, further research is required for sampling in the BIKE and HQC decapsulation, which is inefficient even with the acceleration. Moreover, our work presents – to the best of our knowledge – the first masked CDT sampler operating on Boolean shares and shows its general feasibility. Low-level hardware acceleration decreases the cycle count by a factor of about five. Finally, the polynomial rotation for BIKE benefits significantly from our new instructions. Most of these algorithms we evaluated in this work are very memory intense. The additional memory instructions that are required for masking (at least a factor of two for first-order masking) cannot be accelerated with our ISE and lead to a memory bottleneck.

In addition to more efficient fixed-weight sampling methods that are suitable for BIKE and HQC decapsulation, fully masked implementations of both algorithms would be an important next step for research, especially in the light of a potential standardization of one of the schemes. A fully masked implementation of FrodoKEM, which is still recommended by the French ANSSI and the German BSI, is now feasible with the CDT sampler presented in this work.

Acknowledgements

The work described in this paper has been supported by the German Federal Ministry of Education and Research BMBF through the projects 6GEM (16KISK038) and FlexKI (01IS22086I), by the German Research Foundation (Deutsche Forschungsgemeinschaft, DFG) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972, and by the European Commission under the grant agreement number 101070374.

References

[ABB+22] Nicolas Aragon, Paulo SLM Barreto, Slim Bettaieb, France Worldline, Loïc Bidoux, Olivier Blazy, Philippe Gaborit, Tim Güneysu, Carlos Aguilar Melchor, Rafael Misoczki, et al. BIKE: Bit Flipping Key Encapsulation - Round 4 Submission. 2022. https://bikesuite.org/files/v5.0/BIKE_Spec.2022.10.10.1.pdf.

[BC22] Olivier Bronchain and Gaëtan Cassiers. Bitslicing arithmetic/boolean masking conversions for fun and profit with application to lattice-based KEMs. *IACR TCHES*, 2022(4):553–588, 2022.

[BCD+16] Joppe W. Bos, Craig Costello, Léo Ducas, Ilya Mironov, Michael Naehrig, Valeria Nikolaenko, Ananth Raghunathan, and Douglas Stebila. Frodo: Take off the ring! Practical, quantum-secure key exchange from LWE. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 1006–1018. ACM Press, October 2016.

[BCG+14] Johannes Buchmann, Daniel Cabarcas, Florian Göpfert, Andreas Hülsing, and Patrick Weiden. Discrete ziggurat: A time-memory trade-off for sampling from a Gaussian distribution over the integers. In Tanja Lange, Kristin Lauter, and Petr Lisonek, editors, *SAC 2013*, volume 8282 of *LNCS*, pages 402–417. Springer, Heidelberg, August 2014.

[BCLv17] Daniel J. Bernstein, Chitchanok Chuengsatiansup, Tanja Lange, and Christine van Vredendaal. NTRU prime: Reducing attack surface at low cost. In Carlisle Adams and Jan Camenisch, editors, *SAC 2017*, volume 10719 of *LNCS*, pages 235–260. Springer, Heidelberg, August 2017.

[BCPZ16] Alberto Battistello, Jean-Sébastien Coron, Emmanuel Prouff, and Rina Zeitoun. Horizontal side-channel attacks and countermeasures on the ISW masking scheme. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *CHES 2016*, volume 9813 of *LNCS*, pages 23–39. Springer, Heidelberg, August 2016.

[Ber22] Daniel J Bernstein. Divergence bounds for random fixed-weight vectors obtained by sorting, 2022.

[Bih97] Eli Biham. A fast new DES implementation in software. In Eli Biham, editor, *FSE'97*, volume 1267 of *LNCS*, pages 260–272. Springer, Heidelberg, January 1997.

[CARG23] Agathe Cheriere, Nicolas Aragon, Tania Richmond, and Benoît Gérard. BIKE key-recovery: Combining power consumption analysis and information-set decoding. In Mehdi Tibouchi and Xiaofeng Wang, editors, *Applied Cryptography and Network Security - 21st International Conference, ACNS 2023, Kyoto, Japan, June 19-22, 2023, Proceedings, Part I*, volume 13905 of *Lecture Notes in Computer Science*, pages 725–748. Springer, 2023.

[CCD+23] Jung Hee Cheon, Hyeongmin Choe, Julien Devevey, Tim Güneysu, Dongyeon Hong, Markus Krausz, Georg Land, Marc Möller, Damien Stehlé, and Min-June Yi. Haetae: Shorter lattice-based fiat-shamir signatures. Cryptology ePrint Archive, Paper 2023/624, 2023. https://eprint.iacr.org/2023/624.

[CCHY23] Jung Hee Cheon, Hyeongmin Choe, Dongyeon Hong, and MinJune Yi. Smaug: Pushing lattice-based key encapsulation mechanisms to the limits. Cryptology

ePrint Archive, Paper 2023/739, 2023. https://eprint.iacr.org/2023/739.

[CCK21] Ming-Shing Chen, Tung Chou, and Markus Krausz. Optimizing BIKE for the intel haswell and ARM cortex-M4. *IACR TCHES*, 2021(3):97–124, 2021. https://tches.iacr.org/index.php/TCHES/article/view/8969.

[CGKT22] Ming-Shing Chen, Tim Güneysu, Markus Krausz, and Jan Philipp Thoma. Carry-less to BIKE faster. In Giuseppe Ateniese and Daniele Venturi, editors, *ACNS 22*, volume 13269 of *LNCS*, pages 833–852. Springer, Heidelberg, June 2022.

[CGTZ23] Jean-Sébastien Coron, François Gérard, Matthias Trannoy, and Rina Zeitoun. High-order masking of NTRU. *IACR TCHES*, 2023(2):180–211, 2023.

[Cho16] Tung Chou. QcBits: Constant-time small-key code-based cryptography. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *CHES 2016*, volume 9813 of *LNCS*, pages 280–300. Springer, Heidelberg, August 2016.

[CP23] Hao Cheng and Daniel Page. eliminate: a leakage-focused ise for masked implementation. Cryptology ePrint Archive, Paper 2023/966, 2023. https://eprint.iacr.org/2023/966.

[DDLL13] Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. Lattice signatures and bimodal Gaussians. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 40–56. Springer, Heidelberg, August 2013.

[DGK] Drucker, Gueron, and Kostic. Additional Implementation of BIKE (Bit Flipping Key Encapsulation). https://github.com/awslabs/bike-kem. Accessed: 2023-05-20.

[DGK20] Nir Drucker, Shay Gueron, and Dusan Kostic. QC-MDPC decoders with several shades of gray. In Jintai Ding and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020*, pages 35–50. Springer, Heidelberg, 2020.

[DN12] Léo Ducas and Phong Q. Nguyen. Faster Gaussian lattice sampling using lazy floating-point arithmetic. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 415–432. Springer, Heidelberg, December 2012.

[DPPvW22] Léo Ducas, Eamonn W. Postlethwaite, Ludo N. Pulles, and Wessel P. J. van Woerden. Hawk: Module LIP makes lattice signatures fast, compact and simple. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part IV*, volume 13794 of *LNCS*, pages 65–94. Springer, Heidelberg, December 2022.

[FBR+22] Tim Fritzmann, Michiel Van Beirendonck, Debapriya Basu Roy, Patrick Karl, Thomas Schamberger, Ingrid Verbauwhede, and Georg Sigl. Masked accelerators and instruction set extensions for post-quantum cryptography. *IACR TCHES*, 2022(1):414–460, 2022.

[GGM+21] Si Gao, Johann Großschädl, Ben Marshall, Dan Page, Thinh Pham, and Francesco Regazzoni. An instruction set extension to support software-based masking. *IACR TCHES*, 2021(4):283–325, 2021. https://tches.iacr.org/index.php/TCHES/article/view/9067.

[GHJ+22] Qian Guo, Clemens Hlauschek, Thomas Johansson, Norman Lahr, Alexander Nilsson, and Robin Leander Schröder. Don't reject this: Key-recovery timing attacks due to rejection-sampling in HQC and BIKE. *IACR TCHES*, 2022(3):223–263, 2022.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In Richard E. Ladner and Cynthia Dwork, editors, *40th ACM STOC*, pages 197–206. ACM Press, May 2008.

[Inc] NewAE Technology Inc. ChipWhisperer). https://www.newae.com/chipwhisperer. Accessed: 2023-06-25.

[KAA21] Emre Karabulut, Erdem Alkim, and Aydin Aysu. Single-trace side-channel attacks on ω-small polynomial sampling: With applications to ntru, NTRU prime, and CRYSTALS-DILITHIUM. In *IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2021, Tysons Corner, VA, USA, December 12-15, 2021*, pages 35–45. IEEE, 2021.

[KLRBG23] Markus Krausz, Georg Land, Jan Richter-Brockmann, and Tim Güneysu. A holistic approach towards side-channel secure fixed-weight polynomial sampling. In Alexandra Boldyreva and Vladimir Kolesnikov, editors, *PKC 2023, Part II*, volume 13941 of *LNCS*, pages 94–124. Springer, Heidelberg, May 2023.

[MKK+23] Soundes Marzougui, Ievgen Kabin, Juliane Krämer, Thomas Aulbach, and Jean-Pierre Seifert. On the feasibility of single-trace attacks on the gaussian sampler using a CDT. In Elif Bilge Kavun and Michael Pehl, editors, *Constructive Side-Channel Analysis and Secure Design - 14th International Workshop, COSADE 2023, Munich, Germany, April 3-4, 2023, Proceedings*, volume 13979 of *Lecture Notes in Computer Science*, pages 149–169. Springer, 2023.

[MNP+21] Ben Marshall, G. Richard Newell, Dan Page, Markku-Juhani O. Saarinen, and Claire Wolf. The design of scalar AES instruction set extensions for RISC-V. *IACR TCHES*, 2021(1):109–136, 2021. https://tches.iacr.org/index.php/TCHES/article/view/8729.

[MP21] Ben Marshall and Dan Page. SME: Scalable masking extensions. Cryptology ePrint Archive, Report 2021/1416, 2021. https://eprint.iacr.org/2021/1416.

[MPW22] Ben Marshall, Dan Page, and James Webb. MIRACLE: MIcRo-ArChitectural leakage evaluation A study of micro-architectural power leakage across many devices. *IACR TCHES*, 2022(1):175–220, 2022.

[ND22] Dusan Kostic Nir Drucker, Shay Gueron. Isochronous implementation of the errors-vector generation of BIKE. https://github.com/awslabs/bike-kem, 2022. Accessed: 2022-10-25.

[Pei10] Chris Peikert. An efficient and parallel Gaussian sampler for lattices. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 80–97. Springer, Heidelberg, August 2010.

[Por] Thomas Pornin. Why Constant-Time Crypto? https://www.bearssl.org/constanttime.html. Accessed: 2023-06-30.

[PR13] Emmanuel Prouff and Matthieu Rivain. Masking against side-channel attacks: A formal security proof. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 142–159. Springer, Heidelberg, May 2013.

[Sen21] Nicolas Sendrier. Secure sampling of constant-weight words – application to BIKE. Cryptology ePrint Archive, Report 2021/1631, 2021. https://eprint.iacr.org/2021/1631.

[WBD23] Ruize Wang, Martin Brisfors, and Elena Dubrova. A side-channel attack on a bitsliced higher-order masked crystals-kyber implementation. Cryptology ePrint Archive, Paper 2023/1042, 2023. https://eprint.iacr.org/2023/1042.