

Knuth–Yao masked (Gaussian) sampling

Calvin Abou Haidar¹ and Clément Hoffmann¹

NTT Social Informatics Laboratories, Tokyo, Japan

[{calvin.haidar, clement.hoffmann}@ntt.com](mailto:calvin.haidar, clement.hoffmann@ntt.com)

Abstract. Discrete Gaussian sampling remains one of the most delicate operations to protect against side-channel attacks in lattice-based cryptography. In this work, we present the first masked evaluation of the Knuth–Yao sampler, a generic building block for lattice-based schemes. Its random-walk formulation might seem fundamentally at odds with masking, since the walk’s control flow depends precisely on the secret sample being produced. We show instead that its underlying tree structure is remarkably suited to it. Evaluated as a bitsliced Boolean circuit, the public tree prescribes an evaluation order requiring exactly $L - 2$ masked AND gadgets, where L is the total Hamming weight of the probability expansions of the target distribution. This makes the approach particularly effective for the narrow distributions used in lattice-based schemes. We provide a generic framework that compiles any finite-precision distribution into a masked C implementation with optimized linear-gate count. We use FrodoKEM and HAWK as case studies, and also compile the base samplers of Falcon and HAETAE. On ARM Cortex-M4, our sampler outperforms the state-of-the-art masked CDT approach for every distribution at masking orders 1 to 5, running up to 2.3 times as fast at order 1, even for distributions where it evaluates more non-linear gates. Finally, since the Fujisaki–Okamoto transform ties FrodoKEM to the exact randomness consumption of its specified sampler, we discuss the consequences of this constraint. We benchmarked an unmasked implementation and argue that the Knuth–Yao approach is competitive even without masking, while using identical input randomness.

Keywords: Side-channel analysis · Masking · Discrete Gaussian sampling · Knuth–Yao · Lattice-based cryptography

1 Introduction

As post-quantum schemes are progressively rolled into production, the side-channel security of their implementations has become a pressing concern. Since the main contenders for post-quantum standardization of KEMs and signatures are based on lattices, their protection raises new challenges, absent from classical cryptography. A prominent one is the sampling of narrow discrete distributions (often discrete or rounded Gaussians) which play two roles in lattice-based constructions: as the error term hiding the secret in LWE-style encryption, and as the distribution of the signatures themselves in hash-and-sign schemes, where it guarantees that signatures leak nothing about the signing key. In both cases, the sampler is a natural entry point for an attacker, as any information on the sampled values substantially degrades the security of the scheme. This is not solely a theoretical concern: Gaussian samplers have been the target of numerous side-channel and fault attacks over the years [BHL⁺16, PBY⁺17, GMRR⁺22, HPT⁺25], including some [Pre⁺23, SD⁺25] against provably protected implementations [EFG⁺22].

The timing attacks are, by now, largely under control: a long line of work [MW⁺17, BBE⁺19, HPRR⁺20] has made isochronous Gaussian sampling well-understood. Protecting

against stronger attacks such as power analysis is another matter. The generic countermeasure is masking. It applies to any sampler, but its cost is dominated by non-linear operations, whose overhead grows quadratically with the masking order. Since a single scheme operation draws hundreds to thousands of samples, any per-sample overhead directly translates into a massive slowdown of the masked implementation. Efficient masked Gaussian sampling is therefore a prerequisite for deploying protected implementations of the schemes that rely on it.

Our sampler is a generic building block for current and future lattice-based schemes. It applies to any fixed distribution given at finite precision. We use two schemes as main case studies. **FrodoKEM** [NAB⁺20] is a key encapsulation mechanism based on standard, unstructured LWE. It is a deliberately conservative design on two counts: it forgoes the algebraic structure of ring or module lattices, whose extra structure could conceivably be exploited by future cryptanalysis, and it relies on a genuine (rounded) Gaussian error distribution, for which the security reductions of LWE hold, rather than on a heuristic substitute chosen for efficiency. This caution has made it part of the recommendations by agencies such as the German BSI [BSI26], the French ANSSI [ANS26] and ENISA [EUC25]. It was also recently standardized by ISO as part of the first ISO standard covering post-quantum schemes¹. Its error distribution is sampled thousands of times per operation. **HAWK** [BBD⁺24] is a hash-and-sign signature scheme selected for the second round of the NIST call for additional signatures. It has since been withdrawn, after an attack [Ant26] that exploits a non-trivial automorphism of its lattice and lowers the cost of key recovery on HAWK-256 from 2^{64} to 2^{38} . This attack remains exponential, but HAWK would need roughly twice its parameters. The scheme remains a representative case: discrete Gaussian sampling is not a parameter choice but a structural ingredient at the heart of its signing procedure. Together, these two schemes cover both roles of Gaussian sampling in lattice-based cryptography, with fixed, public parameters, which are the setting in which our contributions apply. Beyond these case studies, we also compile the base samplers of **Falcon** [PFH⁺20] (FN-DSA) and **HAETAE** [CCD⁺24]. The full **Falcon** sampler remains out of reach, as its center is a secret real number rather than one of a few fixed values.

1.1 State of the art

One of the first masked implementations of a CDT sampler was introduced by [GR19] for **QTesla**. They directly translate the CDT sampler to a masked version, providing an arbitrary-order masked implementation. In a similar fashion, Gérard and Guerreau [GG26] implemented the same CDT sampler but for **FrodoKEM**. This baseline was then improved along two independent directions. Eid et al. [EGRZ26] target the number of comparisons: for a table of n entries, they replace the n sequential masked comparisons by the exploration of a perfect binary tree, reducing the count to $\log n$. While binary search is a classical idea, their contribution is to show that the exploration can be masked efficiently without visiting all nodes, by maintaining a pivot value updated after each masked comparison. Independently, Krausz et al. [KLS⁺25] accelerate the linear scan itself through bitslicing, amortizing each masked comparison across a full register of samples. Their public code also covers the **HAWK** and **HAETAE** distributions. Finally, [AEHT26b] combines the two optimizations: they adapt the tree-based operations of [EGRZ26] to make them bitslicing-compatible, yielding the current state-of-the-art masked CDT sampler.

Another recent line of work [AEHT26a, YMZ⁺26] introduces a new way of sampling masked Gaussians by using rejection sampling over discrete Laplace distributions. The work of [YMZ⁺26] can be used for large standard deviations, but is inefficient for the small standard deviation considered in schemes such as **HAWK**. On the contrary, [AEHT26a]

¹<https://www.iso.org/standard/86890.html>

seems to be particularly adapted for these settings, and significantly reduces the number of non-linear (and linear) operations compared to the bitsliced sampler of [AEHT26b]. However, they do not provide an implementation to measure concrete efficiency. Both works only sample discrete Gaussians, and thus do not apply to the rounded Gaussian of FrodoKEM.

1.2 Reviving Knuth–Yao

The Knuth–Yao algorithm [KY76] is an old and elegant answer to the question of exact sampling: it represents the target distribution as a binary tree whose leaves encode the binary expansions of the probabilities, and samples by walking down the tree, one fresh random bit per edge, stopping at the first leaf reached. It has been shown optimal in terms of usage of randomness. This frugality rests on two pillars: the tree structure, and early termination.

Adapting this algorithm to masked implementations, however, has shown tricky. Early termination is a timing channel by definition, and even a carefully engineered partial shortcut was shown to leak in practice [KRR⁺18]. A constant-time evaluation must instead pay for the full depth of the tree on every sample, and a masked evaluation must additionally hide every intermediate bit of the walk, whose control flow depends precisely on the secret sample being produced. Half of what made Knuth–Yao attractive is thus lost in a protected setting, and the algorithm has consequently played no role in masked Gaussian sampling to date.

In this work we show that the other half is worth far more than previously understood. The tree is a public object, entirely determined by the specification of the distribution; only the bits driving the walk are secret. Evaluated as a Boolean circuit over these input bits, the tree exhibits a remarkable property: it prescribes an evaluation order in which each internal node costs exactly one non-linear gate, so that the whole sampler is computed with $L - 2$ SecAnd gadgets, where L is the total number of set bits in the probability expansions. Since non-linear gadgets dominate the cost of masking, the sparsity of the tree translates directly into masking efficiency. And the distributions of lattice-based cryptography are precisely those for which the tree is sparse: narrow Gaussians whose probability expansions carry few set bits. The very structure that once made Knuth–Yao entropy-optimal makes it, fifty years later, a natural masking-friendly sampler.

1.3 Our contributions

Our contributions follow the organization of the paper.

Masking the Knuth–Yao sampler (Section 3). We present the first masked evaluation of the Knuth–Yao sampler. We recast the public DDG tree as a Boolean multiplexer selecting among mutually exclusive leaf events, and show an evaluation order in which every internal node is resolved with a single SecAnd. We prove that the resulting circuit evaluates any distribution of total probability Hamming weight L using exactly $L - 2$ SecAnd gadgets, a count fixed by the distribution alone. The construction uses only standard PINI gadgets and public control flow, so its t -probing security at arbitrary order follows by composition.

A generic circuit-generation framework (Section 4). We provide an open tool that compiles any finite-precision distribution into a masked bitsliced C implementation. Beyond the exact non-linear count above, it reduces the number of linear gates, a problem we cast as shortest linear program computation and address with the Paar algorithm. We instantiate the framework on all FrodoKEM and HAWK parameter sets, and on the base samplers of Falcon and HAETAE. Compared to the state-of-the-art masked CDT approach, our circuits use fewer SecAnd gates for FrodoKEM and HAWK-256, but up to 63% more

Table 1: Performance overview against the masked CDT sampler of [AEHT26b]. Full results in Tables 3 and 4.

		Cycles per sample		Randomness per sample	
		order 1	order 5	order 1	order 5
FrodoKEM-640	CDT	2094	7802	2.13	31.88
	KY	964 (−54%)	5078 (−35%)	1.75 (−18%)	26.25 (−18%)
HAWK-512 (T_1)	CDT	10121	37187	10.00	150.00
	KY	5339 (−47%)	27661 (−26%)	10.28 (+3%)	154.22 (+3%)
Falcon base	CDT	14310	47181	11.59	173.91
	KY	7514 (−47%)	38178 (−19%)	14.31 (+23%)	214.69 (+23%)

on the other distributions. They use 51 to 64% fewer SecXor gates, and 38 to 54% fewer gates in total.

Implementation and evaluation (Section 5). We benchmark our sampler on an ARM Cortex-M4 against the state-of-the-art masked CDT implementation of [AEHT26b], which we extend to the HAWK, Falcon and HAETAE distributions to obtain a baseline. Our sampler is faster for every distribution and at every masking order from 1 to 5, with the largest gains at the low orders most relevant in practice, and we validate the absence of leakage through TVLA experiments. Table 1 illustrates the order of magnitude of the gains. Remarkably, at low order our sampler remains substantially faster even on distributions where it consumes *more* internal randomness, i.e. evaluates more non-linear gates, than CDT. This comes from its much smaller number of linear gates.

Beyond masking (Section 6). Since FrodoKEM relies on the Fujisaki–Okamoto transform, encapsulation and decapsulation must consume their randomness identically, so replacing its sampler amounts to a (light) specification change. We argue that this change is worth considering: our sampler consumes exactly the same input randomness as the reference RCDT, and our packed unmasked implementation runs in the same order of magnitude, in fact faster in our benchmarks. Given the substantial advantage as soon as masking is required, we propose our sampler as a legitimate replacement for, or a specified alternative to, the reference FrodoKEM sampler, and discuss how specifications could accommodate such implementation diversity.

2 Background & State-of-the-art

2.1 Masking and probing security

2.1.1 Masking

Masking is a widely adopted countermeasure designed to protect sensitive data against side-channel analysis. It operates by splitting a secret variable x into $t + 1$ separate components x_i , referred to as *shares*, such that $\bigoplus_{i=0}^t x_i = x$. Consequently, any subset of at most t shares is statistically independent of the secret x . Maintaining this property throughout the execution for all sensitive intermediate values is formalized by the t -probing model [ISW03], which guarantees that an adversary observing up to t intermediate variables learns nothing about the secrets. We refer to t as the *masking order*.

Masking is naturally described at the level of Boolean circuits: the computation is expressed as a sequence of elementary gates, and each gate is replaced by a masked *gadget* that processes shared inputs and produces shared outputs. Throughout this paper,

we denote by SecOp the gadget equivalent of the operator Op : SecAnd , SecXor , SecNot , SecCopy , and SecUnif , which returns a fresh sharing of a uniform random bit. Linear gates (SecXor , SecNot , SecCopy) are performed share-wise, at a cost linear in the number of shares and without randomness. Non-linear gates (SecAnd) incur a quadratic overhead in the number of shares and consume fresh randomness. Their cost dominates the total execution time as the masking order grows. The number of SecAnd gates is therefore the main cost metric at high order, and this metric drives every design choice in this paper: the circuits of [Sections 3](#) and [4](#) are built to minimize it first, and the number of linear gates second.

In this work, we focus solely on Boolean masking. In practice, Gaussian sampling is mostly used within lattice-based primitives, whose subsequent operations often require arithmetic masking². To the best of our knowledge, the most efficient Boolean-to-Arithmetic (B2A) conversion gadgets rely on bitslicing [BC22] and are therefore highly compatible with our approach.

2.1.2 Bitslicing

Our sampler is described as a Boolean circuit: a sequence of logic gates (Xor , Or , ...) operating on *single-bit* values. Evaluating such a circuit naively on a CPU is wasteful, as each instruction would manipulate 1-bit values inside registers of w bits (typically $w = 32$ or 64). Bitslicing is the classical technique that removes this waste when many independent evaluations of the same circuit are needed. The idea is to run ω instances in parallel: the k -th bit of a register holds the value of one wire in the k -th instance. A single bitwise CPU instruction (eg. an AND between two registers) then evaluates the same gate in all ω instances simultaneously, and the cost of the circuit is amortized by a factor ω . The only requirement is a workload that supplies ω independent instances, which Gaussian sampling meets naturally: lattice-based schemes draw hundreds to thousands of independent samples per operation.

Bitslicing combines seamlessly with masking: each shared bit simply becomes an array of $t + 1$ registers, and gadgets apply their gate sequence to full registers. Two consequences matter for this work. First, the bitsliced program *is* the circuit: one instruction per gate. Then, minimizing the number of gates, non-linear *and* linear, is the same thing as optimizing the implementation. Second, only the total number of gates matters, not the depth of the circuit: all ω instances progress together, one gate at a time. The constructions of [Section 3](#) freely exploit this by favoring gate count over depth.

2.1.3 PINI vs SNI

To prove the security and composability of masked gadgets in the t -probing model, different composition frameworks co-exist. The most prominent are Non-Interference (NI) alongside Strong Non-Interference (SNI) [BBD⁺16], and the more recent Probe Isolating Non-Interference (PINI) [CS20] on which we rely.

The core feature of PINI is its stability under composition: a circuit composed of PINI gadgets is trivially PINI. Crucially, linear operations are inherently PINI, which guarantees that they can be executed without any randomness refresh. This composability comes at the cost of a slightly more expensive elementary gadget (eg. the multiplication), but the overhead is offset by simpler security proofs and the ability to deploy bitslicing globally.

Combining SNI proofs with bitslicing is notoriously challenging. The SNI notion relies on strict isolation of shares, typically enforced by inserting refresh gadgets to break dependencies. In a bitsliced implementation, where data is packed and processed in parallel,

²Sharing a value x in shares x_i such that $x = \sum x_i \pmod{q}$, for a given q . This representation makes modular addition and multiplication by a constant linear, and greatly simplifies modular multiplication between secret values.

enforcing this isolation requires frequent refreshes, at every non-linear gate, which destroys the performance gains of vectorization. Conversely, PINI is structurally compatible with these constraints: since PINI gadgets compose trivially, no inter-gadget refresh is needed. While the cost of a PINI SecAnd is roughly twice that of an SNI SecAnd (primarily due to increased randomness requirements), this local overhead is an upper bound; in a complete circuit, the removal of refresh layers results in a significant net performance gain. All the gadgets used in this work are PINI, so that the security of our sampler follows by composition (Section 3).

2.2 CDT-based masked Gaussian sampling

Reference unmasked implementations use CDT sampling to sample half-Gaussian, and then apply a random sign bit to the output. Concretely, this process consists of storing the Cumulative Distribution Function (CDF) of the target distribution in a table, sampling a uniform value, and comparing this uniform value with the stored table, which is done through inverse sampling: if $U \sim \text{Unif}([0, 1])$, then $F^{-1}(U)$ has CDF F . Implementing this technique in a masked way is relatively straightforward: you have to compare the (masked) uniform value to every entry of the table. Then, your output is the number of successful comparisons.

[EGRZ26] came up with the novel idea of representing the CDT as a binary tree rather than as a table. At each level of the tree, a pivot value (to be compared with the uniformly sampled value) can be computed efficiently. This way, every comparison divides the number of possible outputs by 2, while a linear search through the table reduces the candidates by 1 at every comparison. This optimization enables sampling in $\lceil \log_2 n \rceil$ instead of n comparisons, where n is the size of the support of the output distribution. We refer to [EGRZ26] for more details.

2.3 Knuth–Yao sampling

2.3.1 Knuth–Yao sampling

The Knuth–Yao sampler [KY76] draws a discrete random variable from a stream of uniform independent bits used to traverse a binary tree representing the distribution, coined a *discrete distribution generating* (DDG) tree, whose edges are labelled by the successive input bits and whose leaves carry the output values. Sampling is a single root-to-leaf walk: at depth d , the d -th input bit selects the left (0) or right (1) child, and the walk stops as soon as it reaches a leaf, returning its label.

The shape of the tree is entirely determined by the binary expansions of the probabilities. Let χ be a distribution over a finite support S of size m , specified with n bits of precision, i.e. every probability is an integer multiple of 2^{-n} :

$$p_x := \Pr[X = x] = \sum_{d=1}^n p_{x,d} 2^{-d}, \quad p_{x,d} \in \{0, 1\}, \quad x \in S.$$

A set bit $p_{x,d} = 1$ creates one leaf labelled x at depth d . The number of leaves placed at depth d is therefore the Hamming weight of the d -th binary column, $\sum_{x \in S} p_{x,d}$, and the remaining nodes at that depth stay internal and are subdivided further.

Definition 1 (DDG tree). With the notation above, a *DDG tree* $\mathcal{T}_\chi$ is a binary tree built, for $d = 1, \dots, n$, by giving every internal node at depth $d - 1$ two children at depth d and turning exactly $\sum_{x \in S} p_{x,d}$ of these depth- d nodes into leaves, one labelled by each x such that $p_{x,d} = 1$. A node is identified by the input bits $u_1, \dots, u_d$ followed from the root; sampling returns the label of the unique leaf reached.

Remark. Note that there are multiple ways to build a DDG tree, since every permutation of the leaves of a same level gives a valid DDG tree.

Because χ is a probability distribution given at precision n , the leaf depths satisfy the equality $\sum_{x \in S} \sum_{d=1}^n p_{x,d} 2^{-d} = 1$. Since a depth- d leaf is reached with probability 2^{-d} , value x is output with probability $\sum_{d=1}^n p_{x,d} 2^{-d} = p_x$, as required.

Entropy optimality. Knuth and Yao showed that this procedure is essentially optimal in its use of randomness: the expected number ν of input bits it consumes satisfies $H(\chi) \leq \nu < H(\chi) + 2$, where $H(\chi)$ is the Shannon entropy of χ , and no sampler drawing from uniform independent bits can beat $H(\chi)$ in expectation [KY76]. This optimality is the historical motivation for Knuth–Yao. It concerns the *input randomness* driving the sampling, and is unrelated to the *internal randomness* later required for masking. At finite precision (here n bits), the sampler consumes exactly n bits of input randomness per call, matching the standard CDT method.

(In)finite precision sampling. Applying Knuth–Yao to a distribution with non-terminating probabilities yields an infinite tree, and exact sampling then requires either an unbounded bit stream or some form of rejection. This is not a constraint in our setting: the distributions given by cryptographic schemes specifications are *already normalised* at a fixed precision n by their specifications. Concretely, FrodoKEM specifies its centred distribution at $n = 16$ bits and HAWK its discrete Gaussian at $n = 79$ bits (in both cases one bit encodes the sign). The base samplers of Falcon and HAETAE use $n = 72$ and $n = 16$ bits, without sign. The tree and its depth is thus fixed and known in advance, and no rejection ever occurs.

Running example. We illustrate the construction on the small distribution χ over $S = \{0, 1, 2\}$ with $p_0 = 5/8$, $p_1 = 2/8$ and $p_2 = 1/8$, given at precision $n = 3$. Its binary expansions are

$$\frac{5}{8} = (.101)_2, \quad \frac{2}{8} = (.010)_2, \quad \frac{1}{8} = (.001)_2,$$

so value 0 owns leaves at depths 1 and 3, value 1 a leaf at depth 2, and value 2 a leaf at depth 3. The resulting DDG tree is shown in Figure 1. Reading the literals along each root-to-leaf path gives the event triggering each output:

$$\begin{aligned} X = 0 &: \overline{u_1} \quad (\text{depth 1}) \text{ and } u_1 u_2 u_3 \quad (\text{depth 3}), \\ X = 1 &: u_1 \overline{u_2}, \\ X = 2 &: u_1 u_2 \overline{u_3}, \end{aligned}$$

which recovers $p_0 = \frac{1}{2} + \frac{1}{8} = \frac{5}{8}$, $p_1 = \frac{1}{4}$ and $p_2 = \frac{1}{8}$.

The leaves are *mutually exclusive*: every input string reaches exactly one of them.

2.3.2 Knuth-Yao sampling in cryptography

The use of Knuth–Yao for cryptographic Gaussian sampling was initiated by Dwarakanath and Galbraith [DG14], who first cast the discrete Gaussian as a DDG tree and analysed its storage cost on constrained devices. They introduce a block variant that partitions the support in order to compress the zeros of the probability matrix. Sinha Roy, Vercauteren and Verbauwhede [RVV14] turned this into the first hardware sampler: rather than materialising a level of the tree, they traverse it through the sole relative distance between the visited node and the rightmost internal node, and store the probability matrix column-wise with a one-bit differential encoding of the column lengths. This architecture was

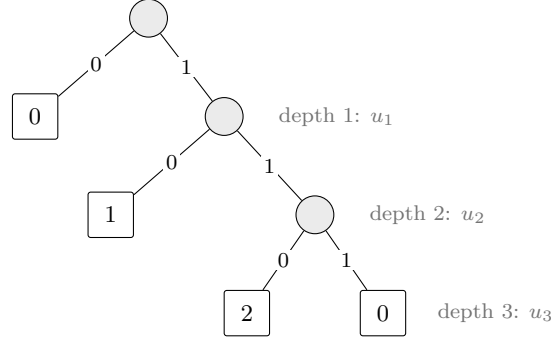


Figure 1: DDG tree of the running-example distribution χ over $\{0, 1, 2\}$ with probabilities $5/8, 2/8, 1/8$, given at precision $n = 3$. Internal nodes are circles, leaves are boxes labelled by the value they output. At depth d the edge is selected by the input bit u_d ($0 = \text{left}$, $1 = \text{right}$); a leaf at depth d is reached with probability 2^{-d} . A branch truncates into a leaf (here on the left) as soon as a value’s binary expansion has a set bit at that depth.

later compacted by Roy et al. [RRVV14], who also gave the first side-channel analysis of a Knuth–Yao sampler: the data-dependent length of the root-to-leaf walk leaks the sampled value through timing and simple power analysis, which they mitigate by randomly shuffling the generated coefficients after sampling. Howe et al. [HKR⁺16] subsequently surveyed the main sampling techniques in hardware and observed that a genuinely constant-time Knuth–Yao walk is prohibitively slow, again resorting to a shuffle to hide the timing.

The closest precursor to our work is the constant-time sampler of Karmakar et al. [KRR⁺18]. They observe that, since the walk is entirely determined by its input bits, each output bit is a fixed Boolean function of these bits. They minimize the number of gates to evaluate these functions using ESPRESSO and evaluate them in constant time, leveraging bitslicing to amortise the cost across many samples. They additionally propose an early-abort variant that trades a measurable timing leakage for speed, which their own evaluation detects. We start from the same input-bits-to-sample mapping, but both our goal and our means differ. All the works above target the timing channel, through shuffling or constant-time evaluation; none considers masking. Furthermore, the Boolean view of [KRR⁺18] is obtained by flattening the mapping into a truth table prior to minimisation, which discards the structure of the tree, whereas we keep the DDG tree as the circuit itself and spend essentially one `SecAnd` per internal node. To the best of our knowledge, this work is the first to evaluate a Knuth–Yao sampler in a masked manner.

3 Masking the Knuth–Yao sampler

We now turn the public DDG tree of Definition 1 into a masked Boolean circuit. Recall from Section 2.1 that, under Boolean masking, linear or affine operations (`SecXor`, `SecNot`, `SecCopy`) are applied share-wise at a cost linear in the number of shares, whereas every non-linear gate (`SecAnd`) incurs a quadratic overhead and consumes fresh randomness. The cost of a `SecAnd` gate is therefore significantly higher than the cost of a linear gate, and minimising that count is the sole objective of this section. Then, the linear gates are dealt with in Section 4.

A central observation is that the tree itself is *public*: the target distribution χ is fixed by the scheme specification, so the shape of $\mathcal{T}_\chi$, the depth of every node and the label of every leaf are all public data. The only sensitive values are the n uniform input bits $u_1, \dots, u_n$ and the intermediate wires derived from them. We draw the input bits in masked form with `SecUnif` and evaluate the public tree as a masked Boolean circuit over these shares.

3.1 The sampler as a masked multiplexer

Evaluating the whole depth- n tree means computing, for each input string $u = (u_1, \dots, u_n)$, the label of the unique leaf it reaches. A leaf ℓ at depth d is reached by a public branch pattern $(c_1, \dots, c_d) \in \{0, 1\}^d$, and we associate to it the *path monomial*

$$m_\ell = \prod_{i=1}^d (u_i \oplus \bar{c}_i), \quad \bar{c}_i = 1 \oplus c_i.$$

Each factor equals u_i when $c_i = 1$ and $\bar{u}_i$ when $c_i = 0$; since c_i is public, it is an **Xor** with a constant and thus a free linear operation. By construction $m_\ell = 1$ exactly when the walk follows the path to ℓ . As the leaves are *mutually exclusive*, at most one path monomial evaluates to 1 for any given u , so the indicator that the sampler outputs a value x is the sum of the monomials of the leaves labelled x :

$$\mathbb{1}_{X=x} = \bigoplus_{\ell : \text{label}(\ell)=x} m_\ell.$$

Mutual exclusivity is what lets us use $\oplus$ in place of a logical **Or**, keeping the whole construction within the $\{\text{SecAnd}, \text{SecXor}, \text{SecNot}\}$ basis. The number of leaves carrying a label x equals the Hamming weight $\text{HW}(p_x)$ of its probability, i.e. the number of set bits in its n -bit expansion.

The sampler returns $X = \sum_{x \in S} x \cdot \mathbb{1}_{X=x}$, encoded in binary on $w = \lceil \log_2 |S| \rceil$ output bits. Writing $x = \sum_j x_j 2^j$ with $x_j \in \{0, 1\}$, the j -th output bit collects every value whose j -th bit is set:

$$y_j = \bigoplus_{x : x_j=1} \mathbb{1}_{X=x} = \bigoplus_{\ell : \text{label}(\ell)_j=1} m_\ell. \quad (\star)$$

The value 0 is the all-zero codeword: by $(\star)$ the monomials of its leaves enter no output equation and are never computed. A non-trivial re-encoding could move this free slot to a heavier value, but the gain is marginal and we keep the natural binary encoding rather than complicate the circuit.

Running example. On the distribution of Figure 1, the support $\{0, 1, 2\}$ needs $w = 2$ output bits ($1 = 01_2$, $2 = 10_2$). The two leaves labelled 0 contribute to no output bit, and values 1 and 2 contribute one monomial each:

$$y_0 = u_1 \bar{u}_2, \quad y_1 = u_1 u_2 \bar{u}_3.$$

Expanding a monomial reintroduces cross terms, e.g. $y_1 = u_1 u_2 (1 \oplus u_3) = u_1 u_2 \oplus u_1 u_2 u_3$; we will therefore *not* expand the monomials, and instead exploit the tree structure to share their non-linear gates. We discuss this extensively in the next subsection.

3.2 One SecAnd per internal node

Evaluating each monomial of $(\star)$ as a separate product of its literals would recompute shared prefixes over and over. We instead compute the wires directly along the tree, visiting every node once, so that each internal node contributes a single non-linear gate.

Let ν be an internal node at depth d , carrying the wire m_ν (its path monomial, with $m_{\text{root}} = 1$ at depth 0). Reading the next input bit u_{d+1} splits ν into its two children, the right one following $u_{d+1} = 1$ and the left one $u_{d+1} = 0$:

$$m_r = m_\nu \wedge u_{d+1}, \quad m_l = m_\nu \wedge \bar{u}_{d+1}.$$

A single **SecAnd** produces both:

$$m_l = m_\nu \wedge (1 \oplus u_{d+1}) = m_\nu \oplus (m_\nu \wedge u_{d+1}) = m_\nu \oplus m_r,$$

so the left child is recovered from the right one with a cheap **SecXor**. Computing both descendants of an internal node therefore costs exactly one **SecAnd**.

The root is an exception: there $m_{\text{root}} = 1$, hence $m_r = u_1$ and $m_l = \overline{u_1}$ can be computed without non-linear gates.

Proposition 1 (Non-linear cost of the masked Knuth–Yao sampler). *Let χ be a distribution over a finite support S with $|S| \geq 2$, given at precision n , and let*

$$L = \sum_{x \in S} \text{HW}(p_x)$$

*be the number of leaves of its DDG tree $\mathcal{T}_\chi$. The masked sampler of Section 3.2 evaluates χ with exactly $L - 2$ **SecAnd** gates.*

Proof. By Definition 1, every internal node of $\mathcal{T}_\chi$ has exactly two children, so $\mathcal{T}_\chi$ is a full binary tree. Each leaf corresponds to a set bit of some probability expansion, hence $\mathcal{T}_\chi$ has $L = \sum_{x \in S} \text{HW}(p_x)$ leaves; since $p_x > 0$ for every $x \in S$, we have $L \geq |S| \geq 2$.

Let I denote the number of internal nodes. Every node but the root has exactly one parent, and every internal node has exactly two children, therefore $I + L - 1 = 2I$, hence $I = L - 1$.

By the construction of Section 3.2, each internal node is split into its two children at the cost of a single **SecAnd**, the sole exception being the root. All remaining operations (the left-child recombinations and the output sums of $(\star)$) are linear. The number of **SecAnd** gates is therefore $I - 1 = (L - 1) - 1 = L - 2$. $\square$

Since L depends only on the probabilities of χ , the **SecAnd** count is fixed by the distribution alone: it is unaffected by the output encoding or the choice of default value, which bear only on the linear gates. Proposition 1 gives the exact cost of our evaluation.

3.3 Discussion

Proposition 1 ties the non-linear cost of the sampler directly to L , the number of leaves of the DDG tree, which is the total Hamming weight of the probability expansions. The masked Knuth–Yao sampler is therefore most efficient on *narrow* distributions, whose probabilities concentrate on a small support and admit sparse binary expansions. Indeed, large probabilities significantly reduce the number of possible 1 appearing in the binary expansion matrix.

The worst case for this sampling method would be the uniform distribution over 2^n values: every probability equals 2^{-n} , the tree is the perfect binary tree of depth n , and $L = 2^n$, so the **SecAnd** count grows exponentially in the precision. On the opposite, Gaussian-like distributions used in lattice-based schemes are suitable for our method. Their probabilities decay quickly, the support is small, and the high-precision tails contribute only isolated set bits, so L remains far below the 2^n worst case. This is precisely the regime in which the sampler is designed to operate, and Section 4 shows it can outperform the CDT approach.

The tree should be read as doing two things at once. First, it makes explicit the Boolean function to be evaluated: each output bit y_j of $(\star)$ is a Boolean function of the input bits $u_1, \dots, u_n$, and the sampler is the simultaneous evaluation of these w functions. Once the tree has given this function, its evaluation can be organised freely. In particular, the linear gates can be rearranged and shared at will, as we do in Section 4. Second, and independently, the tree prescribes an evaluation sequence for the non-linear gates that

attains the $L - 2$ count of [Proposition 1](#): following it directly yields the shared **SecAnd** gates, whereas expanding the y_j into sums of monomials over the literals $u_i, \bar{u}_i$ would hide the shared prefixes and inflate the non-linear cost.

4 Generating the gadget

We described in the previous section the function we need to evaluate, as well as its complexity in terms of non-linear gates. In this section, we describe how to concretely generate the code of our sampler. To this end, we use a Python script³ which takes as input the Probability Mass Function (PMF) of the targeted distribution and generates an optimized circuit. This script runs in three steps:

- It first generates the path of **SecAnd** to traverse the tree, ensuring we compute our monomials with the number of non-linear gate set by [Proposition 1](#),
- optimizes the number of **Xor** operations to compute all monomials,
- generates the netlist circuit of our sampler, and translates it into C code.

4.1 Minimizing the number of linear gates

The number of non-linear gates is fixed by [Proposition 1](#). We thus tackle the problem of reducing the number of linear gates. This is especially important for low order masking, since at high order, the computation and randomness cost of non-linear gates will dominate anyway.

Optimizing the linear part of the circuit is an instance of the *Shortest Linear Program* (SLP) problem: given a set of linear forms over $\mathbb{F}_2$ (here, each output bit as an **Xor** of monomials), find the shortest sequence of two-operand **Xor** instructions computing all of them. The problem is NP-hard [[BMP08](#), Theorem 1], and several heuristics have been developed for circuit synthesis [[Paa97](#), [BMP13](#)]. We rely on Paar’s heuristic [[Paa97](#)]: sort the pairs of operands by their number of occurrences across the remaining equations, emit an instruction for the most frequent pair, substitute it everywhere, and repeat. For instance, if a pair (a, b) appears in many output equations, the instruction $u = a \oplus b$ is emitted early and u replaces every occurrence of the pair. Notice that Paar’s heuristic only allows to produce cancellation-free programs. In our case, this means that for every operation $u \oplus v$, the variables appearing in u will differ from the ones appearing in v . While efficient, this algorithm doesn’t produce the best optimizations in theory. In [[BMP08](#), Theorem 5], Boyar, Matthews and Peralta show that any such program using only cancellation-free operations can achieve a number of gates at most 1.5 times the optimal.

However in [[BMP13](#)], Boyar and Peralta introduce a new heuristic, that does not rely only on cancellation-free operations, thus circumventing the previous lower bound. Their algorithm achieves better optimization in practice, but is too inefficient in practice to run on a large number of input variables with a high number of operations. For this reason, we opted to go with Paar’s heuristic.

Paar’s heuristic alone reduces the number of **Xor** operations of our circuits by roughly 50%. The heuristic of Boyar and Peralta [[BMP13](#)] is known to produce smaller circuits, as it is not restricted to cancellation-free solutions, but its computational cost makes it impractical at the dimensions of our systems. Sun, Yang and Li [[SYL25](#)] recently proposed an improved variant that scales to larger instances and report **Xor** counts 10 to 20% below Paar’s on comparable circuits; we have not implemented it, and leave this additional optimization as a possible refinement.

³given with our code, it can be used to generate the circuit from any finite precision distribution

Unmasked implementation. We underline that our optimization process makes sense only for the masked version of the sampler. In an unmasked variant, one would want to optimize the number of total gates since both **Xor** and **And** operations have roughly the same cost.

4.2 Generating the code

We provide a Python script that takes as input the specification tables from FrodoKEM, HAWK, Falcon and HAETAE. It first transforms these specifications into a PMF. FrodoKEM gives the probability of $X = \pm x$, HAWK and Falcon give RCDT tables for the non-zero probabilities of their half-Gaussians, and HAETAE gives a CDT table. The script then builds the Knuth–Yao tree from the PMF and generates the circuit from the tree. The Python script generating the circuits is available at https://github.com/hoffmanncl/knuth_yao_masked_sampler. It can be easily adapted to produce the C code for the masked circuit of any distribution, provided its PMF.

4.2.1 Theoretical comparison

Our full sampler is composed of the PMF-generated circuits which evaluates the half-Gaussian, to which we apply the sign bit with a full adder in the same manner as [AEHT26b].

Namely, the sign application uses a conditional two’s complement negation over q -bit words: $\text{out} \leftarrow (\text{out} \oplus (2^q - 1) \cdot s) + s$, where q is the output bit-width ($q = 4$ for FrodoKEM-1344, $q = 5$ for the other FrodoKEM sets and for HAWK) and s is the 1-bit sign share. When $s = 0$ the output is unchanged. When $s = 1$, this evaluates $-\text{out} + 1 = -\text{out}$ in $\{4, 5\}$ -bit two’s complement, embedding the k -bit sampled value into the ring $\mathbb{Z}_{2^q}$. The base samplers of Falcon and HAETAE output a half-Gaussian and leave the sign to the caller, so we count no sign application for them.

We give in Table 2 the number of logic gates of our sampler, compared to the one of the CDT approach. To compute it, we count the gates of our circuit, to which we add $q - 1$ SecAnd and $5q - 3$ SecXor for the sign application (the full adder is performed using [BC22, Algorithm 5]). For the numbers of the CDT method, we follow the method of [AEHT26b, Section 4.4]. Precisely, for a CDT tree of depth ℓ and k bits of precision, the comparisons cost ℓk SecAnd and $3\ell k$ SecXor. The pivot computation costs $2^{\ell-1} - \ell$ SecAnd, plus one SecXor per set bit of each public constant conditioned by a path monomial. The benchmarked CDT code also processes the zero bits, so this count favors CDT. The sign application costs $q - 1$ SecAnd and $5q - 3$ SecXor, as in our sampler.

Knuth–Yao replaces the k -bit masked comparisons of CDT by single-bit tree steps. It thus uses 51 to 64% fewer SecXor gates, which explains most of its gain at low order (Section 5.3).

For reference, Table 2 also reports the expected counts of Maskaglia [AEHT26a] on HAWK. It needs fewer than half the SecAnd gates of Knuth–Yao, and up to 18% more SecXor gates. No implementation exists yet. Maskaglia cannot sample the rounded Gaussian of FrodoKEM, and its rejection consumes a variable amount of input randomness.

t -probing security. The security of our sampler follows directly by composition. The circuit is built exclusively from linear operations (which are inherently PINI [CS20]) and PINI SecAnd gadgets, its sensitive inputs are fresh uniform sharings produced by SecUnif, and its control flow is entirely public: the DDG tree is determined by the (public) target distribution, so no branch, memory access, or gate depends on a secret value. Since a composition of PINI gadgets is itself PINI, the complete sampler is PINI, and therefore secure in the t -probing model at arbitrary order. This argument covers the algorithmic description of the sampler; the fact that the *compiled implementation* preserves these

Table 2: Number of SecAnd and SecXor gates per sample, and their total. In parentheses, the difference of KY relative to CDT. The last two rows give the expected counts of Maskaglia [AEHT26a].

Parameter Set	SecAnd		SecXor		Total	
	CDT	KY	CDT	KY	CDT	KY
FrodoKEM-640	68	56 (−17.6%)	271	124 (−54.2%)	339	180 (−46.9%)
FrodoKEM-976	68	55 (−19.1%)	286	116 (−59.4%)	354	171 (−51.7%)
FrodoKEM-1344	49	44 (−10.2%)	182	83 (−54.4%)	231	127 (−45.0%)
HAWK-256 (T_0)	320	301 (−5.9%)	1358	542 (−60.1%)	1678	843 (−49.8%)
HAWK-256 (T_1)	320	278 (−13.1%)	1318	476 (−63.9%)	1638	754 (−54.0%)
HAWK-512 (T_0)	320	380 (+18.8%)	1397	689 (−50.7%)	1717	1069 (−37.7%)
HAWK-512 (T_1)	320	329 (+2.8%)	1416	599 (−57.7%)	1736	928 (−46.5%)
HAWK-1024 (T_0)	320	360 (+12.5%)	1406	665 (−52.7%)	1726	1025 (−40.6%)
HAWK-1024 (T_1)	320	361 (+12.8%)	1392	675 (−51.5%)	1712	1036 (−39.5%)
Falcon base	371	458 (+23.5%)	1994	854 (−57.2%)	2365	1312 (−44.5%)
HAETAE base	169	276 (+63.3%)	1358	592 (−56.4%)	1527	868 (−43.2%)
[AEHT26a], HAWK-512/1024 (T_0)	≈ 157		≈ 704		≈ 861	
[AEHT26a], HAWK-512/1024 (T_1)	≈ 128		≈ 704		≈ 832	

guarantees, i.e. that neither the compiler nor the microarchitecture introduces share-recombining leakage, is assessed empirically in Section 5.4.

5 Implementation and benchmarks

We evaluate our masked Knuth–Yao sampler on an ARM Cortex-M4 (STM32F415) microcontroller, and compare it against the state-of-the-art CDT sampler of [AEHT26b]. We first detail our benchmarking methodology in Section 5.1. We describe in Section 5.2 how we extended the CDT implementation of [AEHT26b] to the HAWK, Falcon and HAETAE distributions, so as to obtain a baseline to compare against. We then report our cycle and randomness measurements in Section 5.3, and assess the security of our implementation through leakage assessment in Section 5.4.

The take-away of this section is the following. At high masking order, the cost of both samplers is dominated by the SecAnd gadgets and their randomness, so their relative performance converges to the ratio of non-linear gates predicted by Table 2. At low order, however, this prediction is pessimistic for our approach. Knuth–Yao evaluates far fewer linear gates, and needs 39 to 56% fewer cycles than CDT at order 1 on every distribution. Notably, this holds even where our sampler evaluates *more* non-linear gates than CDT.

Our code is public: https://github.com/hoffmanncl/knuth_yao_masked_sampler.

5.1 Benchmarking methodology

We adopt the benchmarking methodology of [AEHT26b]. A masked sampler consumes randomness of two kinds. The first kind is the *input randomness*: the n uniform bits driving each sample (the tree path for Knuth–Yao, the compared uniform value for CDT), drawn in shared form. The second kind is the *internal randomness* consumed by non-linear gadgets to refresh masks. The input randomness is identical for both methods, so its generation cost is a constant offset that carries no comparative information; besides, it is not necessarily produced by a TRNG. In FrodoKEM, for instance, it is derived from the message through the Fujisaki–Okamoto transform (see Section 6). We therefore exclude it from our measurements, in both cycle and randomness counts, and report only the cost of the sampling computation itself. The internal randomness, on the other hand, is precisely

where the two approaches differ. On our target platform, drawing randomness is by far the most expensive operation, and it is generated on demand during the computation. We therefore do not treat it as a separate pre-processing cost: we fold its generation into our cycle counts, and additionally report the number of random words consumed, together with the share of the runtime spent drawing them. All randomness is produced by the on-chip TRNG of the STM32F415, which outputs 32 bits every 108 cycles when benchmarked separately.

On the cost of bitslicing. We exclude from our benchmarks the overhead of data-representation conversions, i.e. packing into and unpacking from a bitsliced state. This choice reflects how masked sampling is integrated within a post-quantum scheme. On the input side, when randomness is drawn directly from a hardware TRNG, the random words can be interpreted natively as a bitsliced stream, bypassing any initial transposition. On the output side, the Boolean shares of the sample must undergo a Boolean-to-Arithmetic (B2A) conversion before use in the wider scheme, and the state-of-the-art masked B2A gadget [BC22] requires bitsliced inputs anyway.

Comparison setup. To compare cycles and randomness against the CDT method, we run the code of [AEHT26b] on the same chip as ours. We do not benchmark the other FrodoKEM implementations, as they are strictly outperformed by [AEHT26b]. For HAWK, Falcon and HAETAE, we compare against our own extension of [AEHT26b] (Section 5.2). The pure software HAWK sampler of Krausz et al. [KLS⁺25] costs 11 to 15 kcycles per sample at order 1 on a RISC-V core, input randomness excluded. Our extension costs 10.1 kcycles on the same basis, and thus remains the stronger software baseline. We do not compare against [AEHT26a], since to the best of our knowledge no implementation of their method exists.

Compilation and measurement setup. All samplers are written in portable C. The only exception is the SecAnd gadget, written in assembly to guarantee its PINI security in practice. It brings no measurable speedup over its C equivalent. We compile with `arm-none-eabi-gcc` version 16.1 and flags `-O3 -mcpu=cortex-m4 -mthumb -mfloat-abi=soft`, on top of the ChipWhisperer STM32F4 hardware abstraction layer (HAL), which also drives the on-chip TRNG. Each measurement processes 200 batches of 32 bitsliced samples and reports the per-sample average, read from the DWT cycle counter through the pqm4 HAL. Since the circuits are branch-free and their memory accesses are data-independent, cycle counts are deterministic up to the on-demand availability of the TRNG output. Functional correctness is validated by running the same code under QEMU (`mps2-an386`, a Cortex-M4 machine): after recombining the shares, the masked sampler produces exactly the same outputs as the unmasked circuit on the same input randomness.

5.2 Extending the CDT baseline

HAWK comes in three parameter sets, corresponding to security levels HAWK-256, HAWK-512 and HAWK-1024. During signing, each coefficient of the Gaussian is sampled with a center $t \in \{0, 1/2\}$; the vector of centers is derived from the message and the signing key, and is therefore sensitive. A masked implementation consequently cannot branch on the center: every branch must be evaluated. We sample the Gaussian for both centers and select the correct output with a masked bit encoding the center. A minor optimization is to reuse the same input randomness across both samplers, since only one of the two results is ultimately selected.

We do not implement the full HAWK Gaussian, but focus on the two integer Gaussians it is built from, one per center, which we denote T_0 (center 0) and T_1 (center $1/2$) following

Table 3: Cycle counts per sample of the masked CDT and Knuth–Yao (KY) samplers on ARM Cortex-M4, masking orders 1 to 5, excluding input-randomness generation (Section 5.1). The CDT implementation processes every bit of its constants, and thus performs identically across all HAWK parameter sets. In parentheses, the reduction of KY relative to CDT.

Parameter Set		Masking order				
		1	2	3	4	5
FrodoKEM-640	CDT	2094	2986	4177	5889	7802
	KY	964 (−54.0%)	1588 (−46.8%)	2473 (−40.8%)	3643 (−38.1%)	5078 (−34.9%)
FrodoKEM-976	CDT	2094	2986	4177	5889	7802
	KY	931 (−55.5%)	1542 (−48.4%)	2409 (−42.3%)	3554 (−39.7%)	4962 (−36.4%)
FrodoKEM-1344	CDT	1373	1992	2825	4006	5345
	KY	719 (−47.6%)	1204 (−39.6%)	1895 (−32.9%)	2806 (−30.0%)	3929 (−26.5%)
HAWK (all sets)	CDT	10121	14332	20020	28159	37187
HAWK-256 (T_0)	KY	4862 (−52.0%)	8202 (−42.8%)	12918 (−35.5%)	18234 (−35.2%)	25353 (−31.8%)
HAWK-256 (T_1)	KY	4420 (−56.3%)	7497 (−47.7%)	11844 (−40.8%)	16958 (−39.8%)	23397 (−37.1%)
HAWK-512 (T_0)	KY	6165 (−39.1%)	10377 (−27.6%)	15512 (−22.5%)	22874 (−18.8%)	31811 (−14.5%)
HAWK-512 (T_1)	KY	5339 (−47.2%)	8989 (−37.3%)	14145 (−29.3%)	19893 (−29.4%)	27661 (−25.6%)
HAWK-1024 (T_0)	KY	5874 (−42.0%)	9870 (−31.1%)	14759 (−26.3%)	21747 (−22.8%)	30226 (−18.7%)
HAWK-1024 (T_1)	KY	5911 (−41.6%)	9921 (−30.8%)	14820 (−26.0%)	21828 (−22.5%)	30335 (−18.4%)
Falcon base	CDT	14310	19422	26512	36109	47181
	KY	7514 (−47.5%)	11935 (−38.5%)	18672 (−29.6%)	27493 (−23.9%)	38178 (−19.1%)
HAETAE base	CDT	8916	11926	15486	20644	26230
	KY	4738 (−46.9%)	7830 (−34.3%)	12182 (−21.3%)	17051 (−17.4%)	23589 (−10.1%)

the specification. This gives, over the three parameter sets, the six HAWK distributions we benchmark. Recomposing the full Gaussian from these two components, masked center selection and sign application, is identical whether they are sampled through Knuth–Yao or CDT, and therefore does not affect the comparison. We thus benchmark the two integer Gaussian samplers directly, and extend the CDT implementation of [AEHT26b] accordingly.

Deeper trees. The implementation of [AEHT26b] unrolls the pivot computation for CDT trees of depth 3 and 4, which covers FrodoKEM and HAWK. The wider base Gaussians of Falcon and HAETAE need depths 5 and 7. We made the depth ℓ generic. The pivot monomials are indexed by sets of levels and built incrementally, which generalizes the pivot computation of [AEHT26b] to $2^{\ell-1} - \ell$ SecAnd for any depth. In total, $2^\ell - \ell - 1$ of its k -bit public constants are conditioned by a non-constant monomial, which costs many linear gates. This cost grows exponentially with ℓ , hence linearly with the size of the support. It stays small for FrodoKEM and HAWK, but not for the wider supports of Falcon and HAETAE. At order 1, the pivot computation accounts for 40% of the CDT runtime on Falcon and 70% on HAETAE.

5.3 Benchmarks

We report in Table 3 the cycle cost of the different parameter sets of FrodoKEM and HAWK, and of the base samplers of Falcon and HAETAE.

The corresponding randomness consumption is given in Table 4.

As the masking order grows, the relative performance of the two samplers converges to the ratio of non-linear gates reported in Table 2. This is expected: at high order the runtime is dominated by the SecAnd gates and the randomness they consume. Drawing

Table 4: Internal randomness consumption per sample (32-bit words), masking orders 1 to 5, excluding the input randomness (Section 5.1). In parentheses, the share of the cycles of Table 3 spent drawing these words from the TRNG, at 108 cycles per word. The last column reports the overhead of KY relative to CDT, constant across orders since the internal randomness is proportional to the number of SecAnd gates.

Parameter Set		Masking order					Gain
		1	2	3	4	5	
FrodoKEM-640	CDT	2.13 (11%)	6.38 (23%)	12.75 (33%)	21.25 (39%)	31.88 (44%)	
	KY	1.75 (20%)	5.25 (36%)	10.50 (46%)	17.50 (52%)	26.25 (56%)	−17.6%
FrodoKEM-976	CDT	2.13 (11%)	6.38 (23%)	12.75 (33%)	21.25 (39%)	31.88 (44%)	
	KY	1.72 (20%)	5.16 (36%)	10.31 (46%)	17.19 (52%)	25.78 (56%)	−19.1%
FrodoKEM-1344	CDT	1.53 (12%)	4.59 (25%)	9.19 (35%)	15.31 (41%)	22.97 (46%)	
	KY	1.38 (21%)	4.13 (37%)	8.25 (47%)	13.75 (53%)	20.63 (57%)	−10.2%
HAWK (all sets)	CDT	10.00 (11%)	30.00 (23%)	60.00 (32%)	100.00 (38%)	150.00 (44%)	
HAWK-256 (T_0)	KY	9.41 (21%)	28.22 (37%)	56.44 (47%)	94.06 (56%)	141.09 (60%)	−5.9%
HAWK-256 (T_1)	KY	8.69 (21%)	26.06 (38%)	52.13 (48%)	86.88 (55%)	130.31 (60%)	−13.1%
HAWK-512 (T_0)	KY	11.88 (21%)	35.63 (37%)	71.25 (50%)	118.75 (56%)	178.13 (60%)	+18.8%
HAWK-512 (T_1)	KY	10.28 (21%)	30.84 (37%)	61.69 (47%)	102.81 (56%)	154.22 (60%)	+2.8%
HAWK-1024 (T_0)	KY	11.25 (21%)	33.75 (37%)	67.50 (49%)	112.50 (56%)	168.75 (60%)	+12.5%
HAWK-1024 (T_1)	KY	11.28 (21%)	33.84 (37%)	67.69 (49%)	112.81 (56%)	169.22 (60%)	+12.8%
Falcon base	CDT	11.59 (9%)	34.78 (19%)	69.56 (28%)	115.94 (35%)	173.91 (40%)	
	KY	14.31 (21%)	42.94 (39%)	85.88 (50%)	143.13 (56%)	214.69 (61%)	+23.5%
HAETAE base	CDT	5.28 (6%)	15.84 (14%)	31.69 (22%)	52.81 (28%)	79.22 (33%)	
	KY	8.63 (20%)	25.88 (36%)	51.75 (46%)	86.25 (55%)	129.38 (59%)	+63.3%

this randomness from the TRNG takes up to 61% of the Knuth–Yao runtime at order 5 (Table 4). The distributions where Knuth–Yao needs more SecAnd gates than CDT, such as HAWK-512 (T_0) or HAETAE, thus gain less at high orders.

At order 1, however, the SecAnd ratio alone does not explain our gain. The linear gates do. On every distribution, the measured order-1 gain is much closer to the reduction in total gate count of Table 2 than to that in SecAnd gates. For instance, sampling HAWK-512 (T_0) takes 39.1% fewer cycles with our method than with CDT, even though it consumes 18.8% more randomness.

On 64-bit or AVX platforms, bitslicing scales with the register width for both samplers. The platform-independent gate counts of Table 2 thus remain the relevant comparison.

Impact on the full scheme. In the masked FrodoKEM of [GG26], sampling takes one third to two thirds of a decapsulation. The sampler thus matters at the scheme level. We did not port ours to their code. Their implementation is not bitsliced, so a fair integration would require rewriting the full scheme.

5.4 Leakage assessment

To ensure that our implementations do not exhibit trivial first-order leakages, we run a test-vector leakage assessment (TVLA) on the FrodoKEM-640 implementation. We selected this parameter set as it produces the longest circuit among those our setup can assess⁴, making it the most leakage prone.

We follow the methodology of [EGRZ26, AEHT26b], and perform a "fixed vs random" TVLA, where Student's t-test is performed with 2 sets of 100k traces each that are

⁴The HAWK, Falcon and HAETAE circuits are longer still, but exceed what our setup can measure: the ChipWhisperer Pro cannot record enough samples per trace to cover their full execution.

interleavably measured, with the first set using a fixed sampling randomness while the second uses fresh randomness each time.

As in [AEHT26b], we use the framework of [DZD⁺17] to establish a reasonable threshold to assess the presence of leakage. Our measurements are performed using a Chipwhisperer Pro, on a STM32F415 target board.

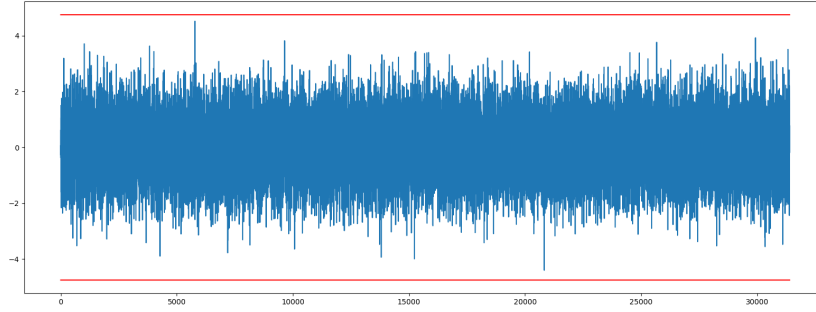


Figure 2: Fixed vs random TVLA for 200k traces for the masked Knuth–Yao FrodoKEM-640 sampler. The leakage threshold is delimited by the red lines.

6 On specification compliance and unmasked sampling

Our sampler evaluates the exact distributions specified by each scheme, but it does not consume its input randomness in the same way as the reference samplers: for a given input stream, the sequence of output samples differs. It therefore fails the Known Answer Tests (KATs) of the reference implementations. This section discusses when this matters, and argues that for FrodoKEM the Knuth–Yao sampler is a legitimate candidate to replace the reference RCDT sampler altogether.

6.1 Randomness usage in KEMs and signatures

How constrained the use of randomness is depends on the type of primitive. For signatures, the Gaussian sampling of the signing procedure is a purely internal operation: no other party re-executes it, not even the verifier. Any sampler producing the correct distribution is interchangeable, and the question of matching the reference KATs is a matter of testing convenience rather than interoperability.

The situation is different for KEMs built from the Fujisaki–Okamoto (FO) transform, such as FrodoKEM. There, CCA security is obtained by re-encryption: the decapsulation re-derives the encryption randomness from the decrypted message and checks that re-encrypting yields the received ciphertext, which forces encapsulation and decapsulation to derive their noise *identically, bit for bit*. The sampler is thus part of the specification itself: two implementations that sample the same distribution but consume the randomness stream differently are not interoperable. Replacing the sampler of FrodoKEM is therefore a specification change: a light one, as it modifies neither the distribution nor the amount of randomness consumed, but a specification change nonetheless. Masking, on the other hand, does not affect interoperability. Masked and unmasked Knuth–Yao evaluate the same Boolean circuit, and return identical outputs on identical input randomness. As long as both use Knuth–Yao, an unprotected device thus interoperates with a masked one.

Table 5: Unmasked cycle counts per sample on ARM Cortex-M4. The reference sampler is the scalar RCDT of the FrodoKEM specification. Scenario A follows the specification’s randomness handling (input and output transpositions included); Scenario B consumes the randomness in bit-plane order (output transposition only).

	Reference RCDT	KY, Scenario A	KY, Scenario B
FrodoKEM-640	66.0	53.8	20.8
FrodoKEM-976	60.0	52.6	19.6
FrodoKEM-1344	52.0	48.0	15.0

6.2 Unmasked performance

We argue that our method is not only more efficient in a side-channel protected setting, but also competitive in an unmasked one. It consumes exactly as much input randomness as the reference RCDT technique, and, as we show below, it is computationally at least as efficient. While Gaussian sampling is not a bottleneck in an unmasked implementation, this matters for the FO argument above: a specification change is only defensible if it does not degrade the unmasked setting that most deployments use.

In the unmasked setting, the cost of converting to and from the bitsliced representation becomes a significant factor (which is a difference from the masked setting, as discussed in Section 5.1). We therefore split our analysis between the cost of evaluating the circuit and the cost of transposing.

On the cost of bitslicing. Without masking, no Boolean-to-Arithmetic conversion is needed, so the output must be de-bitsliced in all cases. For the input, two scenarios arise. In *Scenario A*, we follow the specification’s randomness handling to the letter: the input stream maps each 16-bit chunk to one coefficient, and must be transposed into bitsliced form. In *Scenario B*, the specification is slightly adapted so that the randomness (which is already produced by Shake) is consumed in bit-plane order and arrives naturally bitsliced, removing the input transposition entirely. Note that Scenario B changes only the indexing of the randomness stream, not the amount consumed nor the distribution sampled: it is the same class of light specification change as replacing the sampler itself.

In both scenarios, we amortize the transpositions by packing: our transposition⁵ operates on a 32×32 bit matrix, while a batch of 32 samples only occupies 16 input bit-planes (FrodoKEM) and 4 to 5 output bit-planes. We thus pack two input batches per transposition, and 6 to 8 output batches depending on the parameter set (4 output bits, packed by 8, for FrodoKEM-1344; 5 bits, packed by 6, for FrodoKEM-640/976). Even so, in Scenario B the de-bitslicing alone costs more than the evaluation of the circuit itself.

On the cost of the circuit. Once the representation conversions are set aside, the circuit evaluation is remarkably cheap: about 7 amortized cycles per sample for FrodoKEM-1344. At this scale, further optimization (eg. minimizing the total gate count rather than the number of SecAnd) would bring no measurable benefit, so we simply reuse the circuit generated for the masked setting. This circuit is branch-free, so the unmasked implementation is constant time.

Table 5 compares the reference RCDT sampler with our Knuth–Yao sampler in both scenarios.

⁵which can be achieved with a complexity of $\mathcal{O}(w \log w)$ using the algorithm detailed in [War13], while the naive algorithm runs in $\mathcal{O}(w^2)$

We do not claim that our method outperforms the standard way of sampling the Gaussian. A firm conclusion would require a thorough analysis with equally optimized implementations of both approaches. For instance, the reference sampler could be packed as well, though this would preclude the built-in 32-bit subtraction it relies on for comparisons. We therefore limit our claim to the following: our method consumes the same amount of input randomness as the reference sampler and runs in the same order of magnitude, possibly a few times faster. In other words, adopting it costs nothing in the unmasked setting, while bringing a substantial advantage as soon as masking is required. We are aware that FrodoKEM was recently standardized by ISO, making an outright replacement of the sampler unlikely in the short term; we rather see this discussion as evidence that the choice of a sampling algorithm carries a real cost for protected implementations, and as a case for future revisions and future specifications to accommodate implementation diversity, for instance by providing input–output test data for the sampler alone, decoupling the specified distribution from the strategy used to evaluate it.

6.3 The case of signatures

For signatures such as HAWK, Falcon and HAETAE, a similar discussion could be held, but the stakes are lower. As discussed in Section 6.1, their randomness usage is not constrained by an FO-style re-execution: a natural implementation draws the input randomness directly from a TRNG, which additionally avoids deriving it from a seed through a prohibitively expensive masked Shake. On HAWK, Maskaglia [AEHT26a] also needs fewer SecAnd gates than our sampler (Table 2), although its rejection sampling may induce a non-negligible implementation overhead. We leave a deeper treatment of signature samplers to future work.

Concluding remarks

This work introduced the first masked evaluation of the Knuth–Yao sampler. By viewing the DDG tree as a Boolean circuit, the sampler reduces to $L - 2$ SecAnd gates for a distribution with L set probability bits, which makes it particularly effective for the narrow Gaussians of lattice-based schemes. Our compiler applies to any fixed distribution given at finite precision. On FrodoKEM, HAWK, and the base samplers of Falcon and HAETAE, our bitsliced implementation significantly outperforms the state-of-the-art masked CDT approach, with the largest gains at low masking orders, which is the regime most relevant in practice. Since the sampler is also competitive unmasked while consuming the same randomness budget, we believe it constitutes a sound proposal for the FrodoKEM specification. More generally, it is a building block for the samplers of current and future lattice-based schemes, whenever the target distribution is public.

References

- [AEHT26a] Calvin Abou Haidar, Thomas Espitau, Clément Hoffmann, and Mehdi Tibouchi. Maskaglia: A new, efficient approach to masked discrete gaussian sampling. In Nadia Heninger and Mike Rosulek, editors, *Advances in Cryptology – CRYPTO 2026*, Lecture Notes in Computer Science. Springer, 2026. To appear.
- [AEHT26b] Calvin Abou Haidar, Thomas Espitau, Clément Hoffmann, and Mehdi Tibouchi. Slicing bits and cutting costs in CDT sampling: High-order masking of FrodoKEM’s gaussian sampler, revisited. Cryptology ePrint Archive, Paper 2026/1363, 2026.
- [ANS26] Règles et recommandations concernant le choix et le dimensionnement des mécanismes cryptographiques V3.00. Technical Report PG-083, Agence nationale pour la sécurité des systèmes d’information, March 2026. <https://messervices.cyber.gouv.fr/documents-guides/anssi-guide-mecanismes-crypto-3.00.pdf>.
- [Ant26] Anthropic. An improved key recovery attack on HAWK, July 2026. https://anthropic.com/document/hawk_key_recovery.pdf.
- [BBD⁺16] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. Strong non-interference and type-directed higher-order masking. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 116–129. ACM Press, October 2016.
- [BBD⁺24] Joppe W. Bos, Olivier Bronchain, Léo Ducas, Serge Fehr, Yu-Hsuan Huang, Thomas Pornin, Eamonn W. Postlethwaite, Thomas Prest, Ludo N. Pulles, and Wessel van Woerden. HAWK. Technical report, National Institute of Standards and Technology, 2024. available at <https://csrc.nist.gov/Projects/pqc-dig-sig/round-2-additional-signatures>.
- [BBE⁺19] Gilles Barthe, Sonia Belaïd, Thomas Espitau, Pierre-Alain Fouque, Mélissa Rossi, and Mehdi Tibouchi. GALACTICS: Gaussian sampling for lattice-based constant- time implementation of cryptographic signatures, revisited. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 2147–2164. ACM Press, November 2019.
- [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.
- [BHL⁺16] Leon Groot Bruinderink, Andreas Hülsing, Tanja Lange, and Yuval Yarom. Flush, gauss, and reload - A cache attack on the BLISS lattice-based signature scheme. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *CHES 2016*, volume 9813 of *LNCS*, pages 323–345. Springer, Berlin, Heidelberg, August 2016.
- [BMP08] Joan Boyar, Philip Matthews, and René Peralta. On the shortest linear straight-line program for computing linear forms. In *International Symposium on Mathematical Foundations of Computer Science*, pages 168–179. Springer, 2008.

- [BMP13] Joan Boyar, Philip Matthews, and René Peralta. Logic minimization techniques with applications to cryptology. *Journal of Cryptology*, 26(2):280–312, 2013.
- [BSI26] Cryptographic mechanisms: Recommendations and key lengths. Technical Report TR-02102-1, Bundesamt für Sicherheit in der Informationstechnik, January 2026. <https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TG02102/BSI-TR-02102-1.pdf>.
- [CCD⁺24] Jung Hee Cheon, Hyeongmin Choe, Julien Devevey, Tim Güneysu, Dongyeon Hong, Markus Krausz, Georg Land, Marc Möller, Damien Stehlé, and MinJune Yi. HAETAE: Shorter lattice-based fiat-shamir signatures. *IACR TCHES*, 2024(3):25–75, 2024.
- [CS20] Gaëtan Cassiers and François-Xavier Standaert. Trivially and efficiently composing masked gadgets with probe isolating non-interference. *IEEE Transactions on Information Forensics and Security*, 15:2542–2555, 2020.
- [DG14] Nagarjun C Dwarakanath and Steven D Galbraith. Sampling from discrete gaussians for lattice-based cryptography on a constrained device. *Applicable Algebra in Engineering, Communication and Computing*, 25(3):159–180, 2014.
- [DZD⁺17] A Adam Ding, Liwei Zhang, François Durvaux, François-Xavier Standaert, and Yunsu Fei. Towards sound and optimal leakage detection procedure. In *International Conference on Smart Card Research and Advanced Applications*, pages 105–122. Springer, 2017.
- [EFG⁺22] Thomas Espitau, Pierre-Alain Fouque, François Gérard, Mélissa Rossi, Akira Takahashi, Mehdi Tibouchi, Alexandre Wallet, and Yang Yu. Mitaka: A simpler, parallelizable, maskable variant of falcon. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part III*, volume 13277 of *LNCS*, pages 222–253. Springer, Cham, May / June 2022.
- [EGRZ26] Elie Eid, Aurélien Greuet, Nathan Reboud, and Rina Zeitoun. Efficient high-order masking of FrodoKEMs CDT-based Gaussian sampler. *IACR TCHES*, 2026(3), 2026.
- [EUC25] EUCC agreed cryptographic mechanisms V2. Technical report, European Union Agency for Cybersecurity, May 2025. https://certification.enisa.europa.eu/publications/eucc-guidelines-cryptography_en.
- [GG26] François Gérard and Morgane Guerreau. High-order and Cortex-M4 first-order implementations of masked FrodoKEM. In *CASCADE 2026*, LNCS. Springer, 2026. to appear.
- [GMRR22] Morgane Guerreau, Ange Martinelli, Thomas Ricosset, and Mélissa Rossi. The hidden parallelepiped is back again: Power analysis attacks on falcon. *IACR TCHES*, 2022(3):141–164, 2022.
- [GR19] François Gérard and Mélissa Rossi. An efficient and provable masked implementation of qTESLA. In Sonia Belaïd and Tim Güneysu, editors, *CARDIS 2019*, volume 11833 of *LNCS*, pages 74–91. Springer, 2019.
- [HKR⁺16] James Howe, Ayesha Khalid, Ciara Rafferty, Francesco Regazzoni, and Máire O’Neill. On practical discrete gaussian samplers for lattice-based cryptography. *IEEE Transactions on Computers*, 67(3):322–334, 2016.

- [HPRR20] James Howe, Thomas Prest, Thomas Ricosset, and Mélissa Rossi. Isochronous gaussian sampling: From inception to implementation. In Jintai Ding and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020*, pages 53–71. Springer, Cham, 2020.
- [HPT25] Calvin Abou Haidar, Quentin Payet, and Mehdi Tibouchi. Crowhammer: Full key recovery attack on falcon with a single rowhammer bit flip. In *CRYPTO 2025, Part V*, LNCS, pages 103–135. Springer, Cham, August 2025.
- [ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 463–481. Springer, Berlin, Heidelberg, August 2003.
- [KLS⁺25] Markus Krausz, Georg Land, Florian Stolz, Jan Richter-Brockmann, and Tim Güneysu. To extend or not to extend: Agile masking instructions for PQC. In *CANS 25*, LNCS, pages 70–92. Springer, Singapore, December 2025.
- [KRR⁺18] Angshuman Karmakar, Sujoy Sinha Roy, Oscar Reparaz, Frederik Vercauteren, and Ingrid Verbauwhede. Constant-time discrete gaussian sampling. *IEEE Transactions on Computers*, 67(11):1561–1571, 2018.
- [KY76] D. Knuth and A. Yao. *Algorithms and Complexity: New Directions and Recent Results*, chapter The complexity of nonuniform random number generation. Academic Press, 1976.
- [MW17] Daniele Micciancio and Michael Walter. Gaussian sampling over the integers: Efficient, generic, constant-time. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 455–485. Springer, Cham, August 2017.
- [NAB⁺20] Michael Naehrig, Erdem Alkim, Joppe Bos, Léo Ducas, Karen Easterbrook, Brian LaMacchia, Patrick Longa, Ilya Mironov, Valeria Nikolaenko, Christopher Peikert, Ananth Raghunathan, and Douglas Stebila. FrodoKEM. Technical report, National Institute of Standards and Technology, 2020. available at <https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions>.
- [Paa97] Christof Paar. Optimized arithmetic for reed-solomon encoders. In *Proceedings of IEEE international symposium on information theory*, page 250. IEEE, 1997.
- [PBY17] Peter Pessl, Leon Groot Bruinderink, and Yuval Yarom. To BLISS-B or not to be: Attacking strongSwan’s implementation of post-quantum signatures. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1843–1855. ACM Press, October / November 2017.
- [PFH⁺20] Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. FALCON. Technical report, National Institute of Standards and Technology, 2020. available at <https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions>.

- [Pre23] Thomas Prest. A key-recovery attack against mitaka in the t -probing model. In Alexandra Boldyreva and Vladimir Kolesnikov, editors, *PKC 2023, Part I*, volume 13940 of *LNCS*, pages 205–220. Springer, Cham, May 2023.
- [RRVV14] Sujoy Sinha Roy, Oscar Reparaz, Frederik Vercauteren, and Ingrid Verbauwhede. Compact and side channel secure discrete Gaussian sampling. Cryptology ePrint Archive, Report 2014/591, 2014.
- [RVV14] Sujoy Sinha Roy, Frederik Vercauteren, and Ingrid Verbauwhede. High precision discrete Gaussian sampling on FPGAs. In Tanja Lange, Kristin Lauter, and Petr Lisonek, editors, *SAC 2013*, volume 8282 of *LNCS*, pages 383–401. Springer, Berlin, Heidelberg, August 2014.
- [SD25] Vladimir Sarde and Nicolas Debande. Improvement of side-channel attacks on mitaka. Cryptology ePrint Archive, Report 2025/873, 2025.
- [SYL25] Yao Sun, Runhe Yang, and Ting Li. Revisit the boyar-peralta algorithm to solve the shortest linear program problem. Cryptology ePrint Archive, Paper 2025/1493, 2025.
- [War13] Henry S. Warren, Jr. *Hacker’s Delight*. Addison-Wesley Professional, 2nd edition, 2013.
- [YMZ⁺26] Mert Yassi, Soundes Marzougui, Raymond K. Zhao, Muhammed F. Esgin, Amin Sakzad, and Ron Steinfeld. MAGNET: MAsked gaussian now efficient and table-less. Cryptology ePrint Archive, Paper 2026/728, 2026.