MAGNET: MAsked Gaussian Now Efficient and Table-less

Mert Yassi[1], Soundes Marzougui[2], Raymond K. Zhao[3], Muhammed F. Esgin[1], Amin Sakzad[1] and Ron Steinfeld[1]

[1] Monash University, Melbourne, Australia, {firstname.lastname}@monash.edu
[2] Deutsches Elektronen-Synchrotron, Hamburg, Germany, soundes.marzougui@desy.de
[3] ExeQuantum, Melbourne, Australia, raymond@exequantum.com

Abstract. Discrete Gaussian sampling (DGS) is a fundamental method for generating random noise in various post-quantum cryptographic key generation and signature schemes. However, DGS has been shown to be highly susceptible to side-channel analysis, and several countermeasures have been developed. Masking, a robust countermeasure, is widely employed to secure these schemes against side-channel attacks. Due to the non-linear arithmetic operations involved, DGS has traditionally been considered unsuitable for efficient masked implementations. In this work, we propose MAGNET: an efficient masking design for the novel discrete Gaussian sampler based on Boolean circuits introduced by Wei et al. at ACM CCS 2023. With MAGNET, we demonstrate that DGS can be implemented in a masking-friendly manner. Previous masked DGS approaches in the literature have relied on computation-intensive floating point operations or table-lookup-based techniques using Cumulative Distribution Tables (CDT). In contrast, we show that DGS can be efficiently masked for moderate orders without relying on heavy computation or precomputed large lookup tables. In addition to delivering good performance at a small standard deviation σ, the efficiency of MAGNET becomes increasingly significant in large σ settings. MAGNET achieves up to 17× speed-up at $\sigma = 256$, and 56× speed-up at $\sigma = 1024$ over the CDT-based sampler of Gérard and Rossi (2019). We provide an arbitrary-order C implementation and a first-order ARM Cortex-M4 implementation of MAGNET. We validate the practical security of the first-order implementation through Test Vector Leakage Assessment (TVLA) and systematic hardening of gadgets that exhibit side-channel leakage.

Keywords: Lattice-Based Cryptography · Discrete Gaussian Sampling · Masking · Side-Channel Attacks · Implementation

1 Introduction

Recent rapid developments in quantum computing technologies have raised serious concerns about the security of existing cryptographic systems. In response, research has turned to quantum resistant approaches, with lattice-based cryptography emerging as a leading practical solution, because of its versatility across a wide range of applications. Lattice-based constructions require high-quality randomness at multiple stages to support applications such as key exchange, digital signatures, and fully homomorphic encryption. This randomness is sampled from a well-defined error distribution. In this setting, the discrete Gaussian distribution is often preferred for its tunable parameters and strong statistical guarantees. Discrete Gaussian sampling (DGS) is the procedure of drawing samples from a discrete Gaussian distribution defined by standard deviation $\sigma > 0$ and center $c \in \mathbb{R}$. Zero-centered discrete Gaussian samplers are widely used in lattice-based cryptography, offering more

efficient and simpler constructions. For example, [Lyu12, DDLL13] demonstrate that zero-centered discrete Gaussians lead to more compact signatures, and [LSS24] employs them to shorten zero-knowledge proofs. However, DGS routines used in lattice-based constructions can be vulnerable to side-channel attacks (SCA). An example is the discrete Gaussian sampler with zero center in BLISS [DDLL13], which has been shown to be susceptible to SCA in [BHLY16, EFGT17, BDE+18, TW20].

To protect DGS against SCA, one of the most effective approaches is to apply masking techniques. Masking provides resistance to SCA by splitting sensitive data into a fixed number of shares and hiding each share with fresh, independent randomness. It is widely used in lattice-based cryptography to protect cryptographic primitives against SCA [BBE+18, MGTF19, CGMZ22, EEN+24]. When a secret value is separated into multiple shares, the number of executions required by SCA to recover the original value increases exponentially with the number of shares [CJRR99]. However, masking inevitably introduces additional overhead to the protected procedure. In the case of DGS, achieving an efficient masked implementation of the two most common approaches, CDT and rejection sampling, has traditionally been regarded as particularly challenging [BBE+18, EEN+24]. CDT-based DGS uses large lookup tables, which incur substantial overhead in masking. Discrete Gaussian samplers based on rejection sampling, on the other hand, are also typically avoided in masked implementations, as they include difficult-to-mask transcendental functions, which make masking prohibitively costly [BBE+18]. However, the recent rejection-sampling-based discrete Gaussian sampler of [WYF+23] addresses this challenge by efficiently computing the required exponential functions using multiple bitwise Bernoulli samplings, inspired by the method of [DKM+06]. They show that DGS can be performed efficiently using only Boolean circuits. In particular, this sampler significantly reduces the number of `AND` gates required compared to other methods such as CDT or Knuth-Yao, especially when targeting large σ, which makes the sampler of [WYF+23] potentially suitable for masking. In this work, we propose an efficiently masked zero-centered discrete Gaussian sampler that maintains high performance even at large σ.

1.1 Related Work

DGS can be performed over integers (called integer sampling) or over a n-dimensional lattice. Various discrete Gaussian samplers over lattices [GPV08, Pei10, MP12, DLP14] employ integer sampling as a subroutine, which further highlights its critical role in lattice-based cryptography. However, drawing integer samples from a discrete Gaussian distribution that is negligibly close to the ideal continuous Gaussian has always been considered a costly task, and numerous works have been proposed to improve its performance [DN12, BCG+13, AMAR17, ZSS20a]. Despite extensive research, DGS remains the main bottleneck in lattice-based cryptographic schemes. To improve run-time performance, several studies adopt CDT-based discrete Gaussian samplers [BCNS15, ABB+20]. However, this approach has high memory demand due to the size of the lookup table generated in the offline phase and is generally avoided on constrained devices [DG14]. CDT-based samplers perform well for small σ, but their run-time performance still decreases as the target σ increases.

Securing DGS against SCAs further reduces its efficiency, especially given the inherent difficulty in implementing DGS efficiently. However, such protection is essential; without proper countermeasures, the sampler can leak sensitive information and compromise the security of the entire scheme. For example, [BHLY16] showed a cache-based timing attack that exploited BLISS sampler leakage, enabling recovery of the secret key. To protect DGS against timing attacks, numerous studies have proposed constant-time sampling techniques. Performing constant-time operations without sacrificing efficiency introduces additional challenges, which have been explored in [MW17, KRR+18, KRVV19, ZSS20b]. Although a constant-time implementation makes DGS resistant to timing attacks, the procedure remains vulnerable to more advanced SCA such as power analysis and electromagnetic

emission attacks. For example, [MWG+22] performed a power analysis attack on the discrete Gaussian sampler of GALACTICS [BBE+19], accurately predicting sensitive internal information.

Effective protection against power analysis and electromagnetic emission attacks can be achieved by applying masking techniques. Despite providing security against these attacks, masking also degrades the efficiency of DGS and is, therefore, rarely employed in practice. For example, Dilithium [DKL+18] opted to use the uniform distribution for ease of side-channel-resistant implementation in its design, at the expense of a larger signature size [MGTF19, CGTZ23].

Several studies have explored efficient masking of DGS. [BBE+19] introduced GALACTICS, a constant-time masked implementation of BLISS. The construction follows a CDT lookup approach for masked discrete Gaussian sampling and provides a security proof in the d-probing model. [GR19] proposed a provably secure masked implementation of qTESLA [ABB+20] and masked the DGS using the CDT-based method. However, as with any approach based on table lookup, the performance of this masked implementation decreases rapidly as σ increases. [EFG+22] proposed MITAKA, an easily maskable variant of FALCON, which replaced FALCON's discrete Gaussian sampler (SamplerZ) with the hybrid sampler of [Pre15]. Subsequently, [Pre23] presented a key recovery attack demonstrating that MITAKA is insecure in the d-probing model. In addition, [SD25] demonstrated that the MITAKA masked share-by-share discrete Gaussian sampler is vulnerable to first-order SCA. Recent studies have examined masking of SamplerZ. [BPT+25a] investigated the masking of the floor function in SamplerZ and provided a masked implementation of FALCON. [BPT+25b] improved the masking of the division in SamplerZ, thus increasing the performance of the masked sampler. However, even after optimization, the overhead incurred by masking SamplerZ remains prohibitively high. All these efforts demonstrate that the literature does not offer an efficient masked large σ discrete Gaussian sampler.

1.2 Our Contribution

We introduce the first masked discrete Gaussian sampler over integers that relies on neither floating-point arithmetic nor lookup tables. Building on the Boolean circuit sampler of [WYF+23], we develop an efficient masking strategy applicable to any security order. Our construction integrates established masking gadgets with new gadgets and algorithms designed for efficient masking. We provide a plain C implementation that supports arbitrary masking orders, as well as a first-order ARM Cortex-M4 implementation of MAGNET, both of which are publicly available at https://github.com/mert-yassi/MAGNET.

We propose new masking gadgets `SecNEG`, `SecINC`, and `SecSQU` to increase the performance of MAGNET. We show that our specialized `SecINC` gadget outperforms `SecADD` for increment operations (see Table 3). We also provide the security proofs of the new gadgets in the d-probing model.

To evaluate the performance of MAGNET, we assess the overhead introduced by the masking countermeasure. We compare MAGNET with the CDT-based masked sampler of [GR19] across multiple standard deviations and masking orders (see Table 1), demonstrating the higher efficiency of our design.

MAGNET remains particularly efficient in the large σ setting, which makes it suitable for various (advanced) lattice-based cryptographic constructions such as zero-knowledge proofs [ESLR23, LSS24], private payment protocols [ESZ22b, ESZ22a], and masking-friendly signature schemes [EEN+24]. To further demonstrate its applicability, we integrate MAGNET into MatRiCT+ [ESZ22b], a lattice-based post-quantum blockchain payment protocol that employs discrete Gaussian sampling as a core component. This integration serves as a concrete case study demonstrating that MAGNET can be deployed in practical post-quantum cryptographic schemes while providing excellent performance and maintaining security. In particular, with first-order masking, the core 'spend' procedure

Table 1: Masked DGS run-times (in seconds) for MAGNET versus the CDT-based sampler of [GR19], with security parameter $\lambda = 64$ and number of samples $n = 10{,}000$, across varying values of standard deviation σ and masking orders. Each entry indicates the median of 100 independent timing measurements on an ARM Cortex-A76 processor.

Order	$\sigma = 64$			$\sigma = 256$			$\sigma = 1024$		
	[GR19]	MAGNET	Speedup	[GR19]	MAGNET	Speedup	[GR19]	MAGNET	Speedup
1	1.44	0.28	5.14×	5.65	0.33	17.12×	22.14	0.39	56.77×
2	3.23	1.01	3.20×	12.65	1.20	10.54×	49.67	1.40	35.48×
3	6.26	1.67	3.75×	24.50	1.99	12.31×	96.09	2.31	41.59×

of MatRiCT$^+$ requires 2724.44 seconds when using the sampler of [GR19], whereas the same procedure completes in 46.93 seconds with our MAGNET sampler, corresponding to a speedup of about 58×. With second-order masking, the spend operation takes 154.2 seconds with MAGNET, whereas the implementation using [GR19] does not terminate even after two hours. We report the corresponding runtime results in Table 6.

We prove that MAGNET is secure in the high-order probing model through simulation-based proofs built upon the non-interference framework of [BBD$^+$16]. Our proofs are constructed using the techniques of [CC24].

We provide a practical security evaluation of MAGNET through a leakage assessment of its first-order ARM Cortex-M4 implementation. We further systematically harden all underlying gadgets and sampling algorithms to eliminate observable leakage. In addition, we provide first-order attack-resistant implementations of the masked gadgets and sampling procedures used within MAGNET. To the best of our knowledge, this work presents the first hardened first-order masked implementations of Bernoulli, Geometric, and Laplace sampling. With appropriate parameter selection, the proposed samplers are ready for standalone use in first-order masked settings.

2 Preliminaries

2.1 Notation

In this work, bitwise `AND`, `OR` and `XOR` are denoted with $\wedge$, $\vee$ and $\oplus$, respectively. The left and right shifts are written as $\ll$ and $\gg$, respectively. The word size for a randomized variable is w. To represent the bitwise `NOT` of a Boolean x, $\neg x$ is used. An array of size n is given in bold as $\mathbf{a}_n$. The masking order is denoted by d and the number of secret shares by $d+1$. The masked $d+1$ shares of a secret x are denoted by $(x_i)_{0\le i\le d}$, or more succinctly by (x_i). Lower and upper case letters denote Boolean and arithmetic shares, respectively. An array of length n with masked shares is written as $[arr_i]_{0\le i\le d} = ((x_i^0)_{0\le i\le d}, \ldots, (x_i^{n-1})_{0\le i\le d})$. The j-th element of an array of shares is then (x_i^j). A special array of shares is represented by its identifier as $[p_i^{lap}]_{0\le i\le d}$. Finally, an array composed of arrays of shares is denoted by $[\mathbf{arr}_i]_{0\le i\le d}$.

2.2 Discrete Gaussian Sampling over Integers

DGS over integers involves selecting integer values according to a discrete Gaussian distribution defined by a given center $c \in \mathbb{R}$ and standard deviation $\sigma > 0$ [Pei10]. We define the discrete Gaussian distribution over the integers as follows:

Definition 1 (Discrete Gaussian Distribution)**.** Let $\sigma > 0, c \in \mathbb{R}$. Define the Gaussian weight function as $\rho_{\sigma,c}(x) = \exp(-|x-c|^2/2\sigma^2)$ for $x \in \mathbb{R}$. In the zero-centered case, c

is generally omitted. The zero-centered discrete Gaussian distribution over the integers, denoted by $D_{\mathbb{Z},\sigma}$, is defined as: $D_{\mathbb{Z},\sigma}(x) = \frac{\rho_\sigma(x)}{\sum_{z\in\mathbb{Z}}\rho_\sigma(z)}$, $x \in \mathbb{Z}$.

The precision of a discrete Gaussian distribution is determined by its proximity to the ideal continuous Gaussian distribution. We measure this approximation via statistical distance, a standard metric for comparing distributions. In cryptographic schemes, the security parameter λ sets the target security level and the approximation error must not exceed $2^{-\lambda}$. Statistical distance is used to evaluate the gap between the proposal and the target distributions in this work.

Definition 2 (Statistical Distance)**.** The statistical distance between two distributions $\mathcal{P}$ and $\mathcal{Q}$ on a given support S is defined as: $\Delta(\mathcal{P},\mathcal{Q}) = \frac{1}{2}\sum_{x\in S}|\mathcal{P}(x) - \mathcal{Q}(x)|$.

2.3 Masking

Masking is one of the most reliable and effective countermeasures against side-channel attacks. The fundamental principle of masking is to split a sensitive variable into multiple shares and keep operations secure by randomizing each share regularly. A complete masked algorithm is built from small, reusable masked subcircuits, known as *gadgets*. These gadgets must operate in different masking domains to meet different functional requirements.

Masking domains. A sensitive value can be masked in several different ways. Masking is typically performed using Boolean or arithmetic masks [MOP08]. Since the circuits we construct are almost exclusively Boolean, we primarily focus on Boolean masking. In Boolean masking, a secret variable of order d is split into $d+1$ shares as: $x = x_0 \oplus x_1 \oplus \cdots \oplus x_d$, where each $x_i \in \{0,1\}^w$ for $i = 0,\ldots,d$, and w denotes the word size. Boolean shares enable efficient bitwise operations. Our construction gains its efficiency because, except for the `SecSQU` gadget, all components use Boolean masking. `SecSQU` employs arithmetic masking, in which a secret variable of order d is split into $d+1$ shares as: $x = A_0 + A_1 + \cdots + A_d \mod 2^w$, where each $A_i \in \mathbb{Z}_{2^w}$ for $i = 0,\ldots,d$, and w denotes the word size. Note that ensuring probing security within these domains requires a formal security model.

Probing security model. The threshold probing model of [ISW03], later improved in [BBD+15], assumes an adversary who can observe at most d internal wires per execution to recover secret data from the collected probes. Consequently, a masked algorithm or gadget must achieve d-probing security to resist such an adversary.

Definition 3 (d-probing security)**.** A randomized circuit is d-probing secure if and only if the joint distribution of any subset of at most d internal wires is statistically independent of any secret.

[BBD+16] introduced the security concepts of d-Non-Interference (d-NI) and d-Strong-Non-Interference (d-SNI), allowing the individual security evaluation of each gadget that builds a masked scheme. The probing security of a masked circuit can be proved with the correct use of d-NI or d-SNI proven gadgets. As these concepts are crucial to our construction, it will be helpful to provide their definitions.

Definition 4 (d-Non-Interference (d-NI) security [BBD+16])**.** A gadget is d-Non-Interference (d-NI) secure if and only if every set of d intermediate values can be perfectly simulated by no more than d shares of each of its inputs.

Definition 5 (d-Strong-Non-Interference (d-SNI) security [BBD+16])**.** A gadget is d-Strong-Non-Interference (d-SNI) secure if and only if for every set of d_I internal intermediate

values and $d_{\mathcal{O}}$ of its output shares satisfying $d_I + d_{\mathcal{O}} \leq d$, they can be perfectly simulated by no more than d_I shares of each of its inputs.

In some scenarios, a masked procedure may intentionally reveal selected intermediate variables. To formalize security in this setting, [BBE+18] introduced the notion of d-Non-Interference for gadgets with public outputs (d-NIo) security.

Definition 6 (d-Non-Interference with public outputs (d-NIo) security [BBE+18])**.** A gadget with public outputs X is d-non-interfering with public outputs (d-NIo) if and only if every set of at most d intermediate variables can be perfectly simulated using the public outputs and no more than d shares of each input.

We note for convenience that, if a gadget is d-SNI secure, it is also d-NI secure by its definition. As the name suggests, d-SNI is a stronger concept than d-NI, and is also used to enhance the probing security of masked procedures composed of d-NI gadgets. Additionally, we provide the following property of affine (linear) gadgets.

Definition 7 (Affine gadget [BBD+16])**.** A gadget is affine if and only if it processes each of its input shares individually.

An affine gadget is a masked subcircuit built only using share-wise linear operations. It is trivially d-NI secure, and acts as a straightforward component in the d-NI/SNI composition proofs.

2.4 Bitwise Discrete Gaussian Sampling

In 2023, [WYF+23] proposed a new DGS method to achieve a Differential Privacy (DP) mechanism in the Secure Multi-Party Computation (MPC) setting. The objective of [WYF+23] was to build a DP mechanism utilizing MPC within a high-utility distributed model. Achieving this requires addressing the challenge of efficient noise sampling, which their rejection-sampling-based DGS approach effectively handles by enabling fast and accurate discrete Gaussian generation. We now give an overview of the sampler from [WYF+23]. Henceforth, we shall refer to it as `DP-DGS` for ease of reference. For a clearer presentation of the procedure, we point the reader to the algorithms provided in the Appendix.

Let $\mathcal{T}$ denote the target discrete Gaussian distribution and $\mathcal{P}$ the proposal distribution. `DP-DGS` chooses the discrete Laplace distribution as $\mathcal{P}$, given its close acceptance probability relative to the desired $\mathcal{T}$ and its compatibility with the efficient Boolean circuit design [DKM+06]. The discrete Laplace distribution $\mathcal{L}_{\mathbb{Z}}(t)$ is denoted as: $f_{\mathcal{L}}(x;t) = c \cdot e^{-|x|/t}$, for $x \in \mathbb{Z}$, where t is the discrete Laplace bias parameter, and c is the normalization constant. `DP-DGS` employs rejection sampling to transform discrete Laplace samples into discrete Gaussian samples. The rejection sampling fundamentally tries to acquire the smallest M such that for all x, $M \cdot f_{\mathcal{P}}(x) \geq f_{\mathcal{T}}(x)$ holds, where f denotes the probability function of the value x under the specified distribution. Consequently, the acceptance probability of a sample s from $\mathcal{P}$ can be calculated with $f_*(s) = \frac{f_{\mathcal{T}}(s)}{M \cdot f_{\mathcal{P}}(s)}$. The acceptance condition of s is determined by the acceptance bit b sampled from the Bernoulli distribution $\mathcal{B}(f_*(s))$, which is defined as

$$\mathcal{B}(x) = \begin{cases} p, & x = 1, \\ 1 - p, & x = 0, \end{cases} \tag{1}$$

where probability $p \in [0, 1]$ and variable $x \in \{0, 1\}$. However, sampling the acceptance bit b requires performing an expensive exponential computation. This problem is solved by converting Bernoulli sampling with the bias of some exponential function to distinct

bit-level sampling steps and combining the sampled bits into a Bernoulli sample [DDLL13]. `DP-DGS` completes the rejection phase by generating m candidate Laplace samples and their corresponding acceptance bits, thus producing discrete Gaussian samples. Note that since each sample is independent, the first n accepted samples can be selected from the m candidate samples. Next, we outline the components of `DP-DGS`.

Bitwise Bernoulli sampling. Bernoulli sampling forms the core of the design of `DP-DGS`. The method samples a bit by generating a uniform random value $r \in [0, 1]$ and comparing it against a given probability $p \in [0, 1]$, generating 1 if $r < p$ and 0 otherwise. The challenge of representing real numbers is overcome by employing the bit-by-bit binary approximation method proposed in [DKM+06], where p and r are represented in binary form with μ bits. This bitwise formulation allows the Bernoulli sampling procedure to be achieved using simple logical operations, thus facilitating the implementation in constant time while incurring only a statistical distance of at most $2^{-\mu}$ from the ideal target distribution.

Discrete Laplace sampling from geometric distribution. The discrete Laplace sampling method of `DP-DGS` relies on generating a geometric sample, which is subsequently transformed into a two-sided discrete Laplace sample. To avoid the expensive computational cost typically associated with geometric sampling, `DP-DGS` adopts the ODO (Our Data, Ourselves) protocol of [DKM+06]. This protocol allows for the bit-by-bit generation of a geometric sample through independent bitwise Bernoulli sampling procedures. `DP-DGS` performs κ Bernoulli samplings to get a geometric sample from the geometric distribution $\mathcal{G}(p)$ defined as: $f_{\mathcal{G}}(x; p) = (1-p)\cdot p^x$, for $x \in \mathbb{N}$, where p is the distribution parameter. The truncated geometric distribution $\mathcal{G}_{2^\kappa}(p)$ is built by combining the Bernoulli distributions $\mathcal{B}(p^{2^i}/(1 + p^{2^i}))$ for $i = 0, \ldots, \kappa - 1$.

To sample from the discrete Laplace, a geometric sample is transformed into a two-sided discrete Laplace sample. The target discrete Laplace distribution required for DGS has support in the range $(-N, N)$, which can be covered by extending the geometric distribution $\mathcal{G}_N(e^{-1/t})$ to the negative domain. Although this can be accomplished by applying a sign bit, it leads to double counting of zero due to the geometric distribution covering both $(-N, 0]$ and $[0, N)$. This issue is resolved by adjusting the range of the geometric distribution and applying rejection sampling to the resulting discrete Laplace sample. which is achieved by multiplexing. After the rejection phase, finally the sign bit can be applied using multiplexing.

Acceptance bit sampling from bitwise Bernoulli. Following the generation of a discrete Laplace sample, an acceptance bit must be sampled to determine whether the sample should be rejected. As noted above, rejection sampling involves drawing from the Bernoulli distribution $\mathcal{B}(f_*(s))$, where $f_*(s)$ gives the acceptance probability of the sample s. To align the proposal Laplace distribution closely with the target Gaussian distribution, `DP-DGS` computes this function as

$$f_*(x) = \exp\left(-\frac{(|x| - \sigma^2/t)^2}{2\sigma^2}\right), \tag{2}$$

where t is the discrete Laplace parameter. The costly exponential computation can be avoided using the method given in [DDLL13], which applies a series of bitwise Bernoulli samplings with pre-computed biases. Specifically, `DP-DGS` selects the Bernoulli distribution as $\mathcal{B}(\exp(-u/r))$, since

$$\mathcal{B}(\exp(-u/r)) = \mathcal{B}\left(\exp(-\sum_{i=0}^{l-1} u_i 2^i/r)\right) = \mathcal{B}\left(\prod_{i=0}^{l-1} \exp(-u_i 2^i/r)\right), \tag{3}$$

where $u \in \mathbb{N}, r \in \mathbb{R}^+$. Each u_i for $i = 0, \ldots, l-1$ can be extracted from u using bitwise operations. Consequently, the sampling bit b from $\mathcal{B}(\exp(-u/r))$ can be done by first sampling Bernoulli bits b'_i from $\mathcal{B}(\exp(-2^i/r))$, and then computing $b = \bigwedge_{i=0}^{l-1}(\neg u_i \vee b'_i)$. To be able to use this construction, the following condition must hold:

$$f_*(x) = \exp\left(-\frac{(|x| - \sigma^2/t)^2}{2\sigma^2}\right) = \exp\left(-\frac{g_*(x)}{r}\right), \tag{4}$$

where $g_*(x) \geq 0$ is an integer function that satisfies $u = g_*(x) \in \mathbb{N}, r = 2\sigma^2 \in \mathbb{R}^+$. Note that achieving this equivalence requires the precise selection of the discrete Laplace parameter t. Letting s be the discrete Laplace sample, $g_*(s) = (|s| - z_0)^2$, where $z_0 = \sigma^2/t$. To ensure a high acceptance probability for $\sigma \geq 1$, `DP-DGS` sets $t = \sigma^2/\lfloor\sigma\rceil$. More details on the selection of t and the fine-tuning of z_0 to maximize the acceptance probability can be found in Section 4.5 of [WYF+23]. The subtraction and squaring required to compute $g_*(s)$ are performed in the main DGS loop, following the generation of the Laplace sample s. The acceptance bit sampling algorithm requires the binary decomposition of the integer $u = g_*(s)$ as input, which is used to derive the Bernoulli acceptance bit.

Using the methodology presented, the proposal distribution of [WYF+23] achieves a total statistical distance of $2^{-\lambda}$ from the target Gaussian distribution, where λ is the chosen security parameter. The derivation of this bound is given in Theorem 4.4 in [WYF+23].

3 Masking DP-DGS

This section provides the details of the masking countermeasure we apply to the algorithms of `DP-DGS` based on Boolean circuits. To improve the efficiency of the sampling algorithms, we apply a series of optimizations, with particular focus on enhancing masking performance. For each algorithm, we present our modifications to tailor the construction toward a constant-time, efficient, and zero-centered DGS suitable for large σ setting. Alongside the original algorithms of [WYF+23], we provide our modified versions in Appendix A. We also introduce three new gadgets `SecNEG`, `SecINC`, `SecSQU` to support optimized calculations needed in `DP-DGS`. After making the algorithms masking compatible, we demonstrate our approach to achieve a fully masked DGS implementation. Table 2 lists all the gadgets employed in our masking design, together with their functionality, security in the d-probing model, and corresponding references.

3.1 Masked Bitwise Bernoulli Sampling

In bitwise Bernoulli sampling, we complete the multiplexing operation with just one `AND`, two `XOR`, and one `NOT` operations. The sequence of bitwise operations emulates the selection of $\neg r$ when $t = 1$, and x otherwise. The simplicity of the bitwise Bernoulli sampling procedure and its fully Boolean nature make this component straightforward to mask. Since the `NOT` and `XOR` operations are linear, they incur minimal overhead in the masked setting. The only non-linear operation involved is a single `AND` gate, which constitutes the primary performance bottleneck. We now give our approach for masking this procedure.

Each iteration starts with the generation of $d+1$ uniformly random bits $r_0, r_1, \ldots, r_d$ with the `GenUniRandBit` function. Together, these bits form the secret bit r, i.e., $r = r_0 \oplus r_1 \oplus \cdots \oplus r_d$. They are then refreshed using the multiplication-based `Refresh` gadget [ISW03] to produce w-bit secure masked shares. With `Refresh`, each bit is assigned to a separate share and expanded to the desired word size by randomization, resulting in a properly masked representation. We use the `SecAND` gadget [CGTV15, BBE+18] to perform bitwise `AND` of two Boolean shares. The other bitwise operations are linear and are performed share by share. The masked bitwise Bernoulli sampling algorithm is given in Algorithm 1.

Algorithm 1: MaskedBernoulli

Input: An array of Boolean shares $[p_i]_{0\le i\le d} = (b_i^0)_{0\le i\le d}, \ldots, (b_i^{\mu-1})_{0\le i\le d}$ which satisfies $p = 0.b_0 \ldots b_{\mu-1}$, Boolean shares $(\text{one}_i)_{0\le i\le d}$ of 1, word size w.
Output: Boolean shares of the Bernoulli sample bit $(x_i)_{0\le i\le d}$.

$(x_i)_{0\le i\le d} \leftarrow (\text{one}_i)_{0\le i\le d}$
for $j \leftarrow \mu - 1$ **to** 0 **do**
  $\mathbf{r}_{d+1} \leftarrow$ `GenUniRandBit()` `// Generates` $d+1$ `uniformly random bits`
  $(r_i)_{0\le i\le d} \leftarrow$ `Refresh`$(\mathbf{r}_{d+1})$
  $(t_i)_{0\le i\le d} \leftarrow (b_{j_i} \oplus r_i)_{0\le i\le d}$
  $(r'_i)_{0\le i\le d} \leftarrow (\neg r_0, r_1, \ldots, r_d)$
  $(u_i)_{0\le i\le d} \leftarrow (x_i \oplus r'_i)_{0\le i\le d}$
  $(v_i)_{0\le i\le d} \leftarrow$ `SecAND`$((t_i)_{0\le i\le d},\ (u_i)_{0\le i\le d})$
  $(x_i)_{0\le i\le d} \leftarrow (x_i \oplus v_i)_{0\le i\le d}$
return $(x_i)_{0\le i\le d}$

3.2 Masked Geometric Sampling

The geometric sampler used in discrete Laplace sampling relies on repeated calls to the bitwise Bernoulli sampler. In this context, we employ Algorithm 1. The masked bits sampled by this procedure must be composed in the final stage to produce the masked geometric sample. This is achieved by aligning the masked shares using bitwise shift and `XOR` operations. Since shift operations are not performed across distinct shares, they do not require any refresh mechanisms and can be applied directly. When composing the shifted bits, performing a share-by-share `XOR` is sufficient because the bits in the composition do not overlap. This allows us to efficiently perform the binary composition process $x = x + 2^i b_i$ as $x = x \oplus (b_i \ll i)$, for $i = 0, \ldots, \kappa$. Finally, to prevent the accumulation of statistical dependencies, we rerandomize the shares at the end of each iteration using the `Refresh` gadget. We give the masked geometric sampling procedure in Algorithm 2.

Algorithm 2: MaskedGeometric

Input: An array of Boolean shares $[\boldsymbol{p_i^{\text{geo}}}]_{0\le i\le d} = ([p_i^0]_{0\le i\le d}, \ldots, [p_i^{\kappa-1}]_{0\le i\le d})$ where each $[p_i^j]_{0\le i\le d} = ((b_i^0)_{0\le i\le d}, \ldots, (b_i^{\mu-1})_{0\le i\le d})$, Boolean shares $(\text{one}_i)_{0\le i\le d}$ of 1.
Output: Boolean shares of the geometric sample $(x_i)_{0\le i\le d}$.

$(x_i)_{0\le i\le d} \leftarrow 0$
for $j \leftarrow 0$ **to** $\kappa - 1$ **do**
  $(b_i^j)_{0\le i\le d} \leftarrow \text{MaskedBernoulli}([p_i^j]_{0\le i\le d}, (\text{one}_i)_{0\le i\le d})$
  $(t_i)_{0\le i\le d} \leftarrow (b_i^j \ll j)_{0\le i\le d}$
  $(x_i)_{0\le i\le d} \leftarrow (x_i \oplus t_i)_{0\le i\le d}$
  $(x_i)_{0\le i\le d} \leftarrow$ `Refresh`$((x_i)_{0\le i\le d})$
return $(x_i)_{0\le i\le d}$

3.3 Masked Discrete Laplace Sampling

In the discrete Laplace sampling algorithm of `DP-DGS`, two separate multiplexers are used: one for the application of the sign bit and the other for the rejection sampling procedure. We postpone the generation and application of the sign bit to avoid the absolute value computation in the rejection sampling phase. Furthermore, we accelerate the rejection sampling process by employing bitwise techniques, performing the multiplexing with one `AND`, one `NEG` and one `NOT` operations. As proposed in the original design, the `MUX` operation

Table 2: Secure masking gadgets used in MAGNET.

Gadget	Functionality	Security	Reference
`SecAND`	Logical AND of two Boolean maskings	d-SNI	[CGTV15], [BBE+18]
`SecOR`	Logical OR of two Boolean maskings	d-SNI	[CC24]
`Refresh`	Multiplicative refreshing of a Boolean masking	d-SNI	[ISW03]
`FullRefresh`	Stronger refreshing of a Boolean masking	d-SNI	[Cor14]
`FullXOR`	Recovering secret by recombining all shares	d-NIo	[CGV14]
`SecADD`	Arithmetic add of two Boolean maskings	d-NI	[CGTV15], [BBE+18]
`SecA2B`	Arithmetic-to-Boolean masking conversion	d-SNI	[Gou01], [BCZ18]
`SecB2A`	Boolean-to-arithmetic masking conversion	d-NI	[Gou01], [CGTZ23]
`SecNEG`	2's-complement negation of a Boolean masking	d-SNI	Algorithm 3
`SecINC`	Arithmetic increment of a Boolean masking	d-SNI	Algorithm 4
`SecSQU`	Arithmetic squaring of an arithmetic masking	d-SNI	Algorithm 7

must return $x+1$ if the Bernoulli bit $b=0$, and 0 otherwise. To achieve efficient masking, we complement b and apply arithmetic negation to produce the bitmask for constant-time selection. Since we work within the domain of unsigned arithmetic, the negation computes the two's complement of the input. For illustration, when $b=0$, we have $\texttt{NOT}(b)=1$, and $\texttt{NEG}(\texttt{NOT}(b))=-1=\texttt{0xFF}\ldots\texttt{FF}$, which produce a full-word bitmask of ones. Consequently, the correct selection between $x+1$ and 0 becomes possible through the application of `AND`. Since the sign bit is not applied within this algorithm, its output does not yet correspond to a valid discrete Laplace sample. We apply the sign bit in the main DGS loop, guaranteeing a proper discrete Laplace sample.

Algorithm 5 outlines the masked discrete Laplace sampling procedure. To efficiently mask the discrete Laplace sampling, we utilize Algorithm 1 and Algorithm 2. Before starting the procedure, we precompute the discrete Laplace bias parameter $p^{lap}=f_{\mathcal{L}_{\mathbb{Z},\mathbb{N}}}(0;t)$ and the geometric distribution parameter $p=e^{-1/t}$ as masked Boolean shares and input them at run-time. Linear `NEG` and `NOT` operations for constant-time selection incur minimal cost. Two's complement negation is performed on the least significant bit of the masked Boolean shares of $\texttt{NOT}(b)$. As we operate on $d+1$ masked shares of $\texttt{NOT}(b)$, extracting and manipulating the least significant bit of each share does not compromise the secret bit since an adversary can probe at most d shares. This enables the required conversion to be performed securely. In addition, we refresh the shares after negation to keep the algorithm d-SNI secure. The gadget of two's complement negation is given in Algorithm 3.

An important aspect to consider is the `ADD` component used to calculate $x+1$. Instead of employing a `SecADD` for this, we introduce a specialized variant tailored for increment operations, referred to as `SecINC` (see Algorithm 4). In `SecINC`, the structure of the `AND` tree in the `SecADD` [BBE+18] based on the Kogge-Stone adder is simplified by halving the number of `SecAND` operations required at each level. The procedure begins by copying the input shares x_i into g_i, which will hold the propagate flags per bit. These flags are

Algorithm 3: `SecNEG`

Input: Boolean shares $(x_i)_{0\le i\le d}$ of integer x.
Output: Boolean shares $(z_i)_{0\le i\le d}$ of integer z where $z = (2^w - x) \bmod 2^w$.
1: $(z_i)_{0\le i\le d} \leftarrow (-(x_i \wedge \texttt{0x1}))_{0\le i\le d}$
2: $(z_i)_{0\le i\le d} \leftarrow \texttt{Refresh}((z_i)_{0\le i\le d})$
3: **return** $(z_i)_{0\le i\le d}$

used to check whether adding one at that bit position would propagate a carry. For each prefix level $j = 1, \ldots, W$, it computes an offset pow $= 2^{j-1}$ and defines two bitmasks that split the word into upper and lower halves at that offset. To determine whether a carry can cross a 2^j-bit block, the algorithm checks its adjacent 2^{j-1}-bit subblocks. The entire block propagates a carry only when both subblocks propagate a carry. It shifts g_i by pow, then applies `SecAND` to the shifted and original words to generate the full block propagate flags in a'_i. Using `XOR`, the algorithm places the new flags in the upper bits of g_i while preserving the lower bits. `AND`ing the parts with bitmasks ensures that updates occur at the correct bit positions at each stage. A `Refresh` is used to rerandomize the shares, maintaining d-SNI security of the gadget. After all stages, g_i encodes the complete carry-propagate pattern. Finally, computing the `XOR` sum of x_i, the constant one_i, and the left-shifted mask g_i yields the Boolean shares of $x + 1$, which are immediately refreshed to guarantee d-SNI security. Table 3 compares the cycle counts of `SecINC` and `SecADD` to calculate $x + 1$ in different masking orders.

Algorithm 4: `SecINC`

Input: Boolean shares $(x_i)_{0\le i\le d}, (\text{one}_i)_{0\le i\le d}$ of integers $x, 1$; word size w.
Output: Boolean shares $(z_i)_{0\le i\le d}$ of integer z where $z = x + 1$.
1: $(g_i)_{0\le i\le d} \leftarrow (x_i)_{0\le i\le d}$
2: **for** $j \leftarrow 1$ **to** $W := \lceil \log_2(w-1) \rceil$ **do**
3: pow $\leftarrow 2^{j-1}$
4: `high` $\leftarrow (2^w - 1) \ll$ pow
5: `low` $\leftarrow \neg$`high`
6: $(a_i)_{0\le i\le d} \leftarrow (g_i \ll \text{pow})_{0\le i\le d}$
7: $(a'_i)_{0\le i\le d} \leftarrow \texttt{SecAND}((a_i)_{0\le i\le d}, (g_i)_{0\le i\le d})$
8: $(g_i)_{0\le i\le d} \leftarrow ((a'_i \wedge \texttt{high}) \oplus (g_i \wedge \texttt{low}))_{0\le i\le d}$
9: $(g_i)_{0\le i\le d} \leftarrow \texttt{Refresh}((g_i)_{0\le i\le d})$
10: $(z_i)_{0\le i\le d} \leftarrow (x_i \oplus \text{one}_i \oplus (g_i \ll 1))_{0\le i\le d}$
11: $(z_i)_{0\le i\le d} \leftarrow \texttt{Refresh}((z_i)_{0\le i\le d})$
12: **return** $(z_i)_{0\le i\le d}$

Table 3: Comparison of clock cycle counts for `SecADD` and `SecINC` computing $z = x + 1$ across different masking orders. The reported clock cycles correspond to a single invocation of each gadget and represent the median over 100,000 executions, measured on an ARM Cortex-M4 CPU.

Gadget	Order-1	Order-2	Order-3	Order-4	Order-5
`SecADD`	654	3164	6029	9386	13326
`SecINC`	539	2151	4303	6739	9694
Speed-up	1.21×	1.47×	1.40×	1.39×	1.37×

Algorithm 5: MaskedLaplace

Input: Arrays of Boolean shares $[\boldsymbol{p}^{\mathbf{geo}}{}_i]_{0\le i\le d}, [p_i^{lap}]_{0\le i\le d}$, Boolean shares $(\mathrm{one}_i)_{0\le i\le d}$ of 1.

Output: Boolean shares $(v_i)_{0\le i\le d}$ of the discrete Laplace sample v.

1: $(b_i)_{0\le i\le d} \leftarrow \mathrm{MaskedBernoulli}([p_i^{lap}]_{0\le i\le d}, (\mathrm{one}_i)_{0\le i\le d})$
2: $(x_i)_{0\le i\le d} \leftarrow \mathrm{MaskedGeometric}([\boldsymbol{p}^{\mathbf{geo}}{}_i]_{0\le i\le d}, (\mathrm{one}_i)_{0\le i\le d})$
3: $(b'_i)_{0\le i\le d} \leftarrow (\neg b_0, b_1, \ldots, b_d)$
4: $(t_i)_{0\le i\le d} \leftarrow \mathtt{SecNEG}((b'_i)_{0\le i\le d})$
5: $(u_i)_{0\le i\le d} \leftarrow \mathtt{SecINC}((x_i)_{0\le i\le d})$
6: $(v_i)_{0\le i\le d} \leftarrow \mathtt{SecAND}((t_i)_{0\le i\le d}, (u_i)_{0\le i\le d})$
7: **return** $(v_i)_{0\le i\le d}$

3.4 Masked Acceptance Bit Sampling

We do not alter the acceptance bit sampling method of `DP-DGS` as it is a masking-compatible procedure. To mask it, we start by using the masked bitwise Bernoulli sampling routine. After obtaining the bits $\neg u_j$ and b'_j in masked form, the operation $b = \bigwedge_{j=0}^{l-1}(\neg u_j \vee b'_j)$ is performed using the gadgets `SecOR` and `SecAND`. Since these operations require access to the individual bits of the input $(u_i)_{0\le i\le d}$, bit extraction is performed using bitwise right shifts. A right shift is sufficient to bring the target bit u_j to the least significant position for extraction. This approach works because $(b_i)_{0\le i\le d}$ was initialized to 1, allowing the `SecAND` gadget to operate securely only on the least significant bit. Thus, the bit decomposition of $(u_i)_{0\le i\le d}$ can be efficiently obtained in masked form. After the bitwise shift, the shares are refreshed using the `Refresh` gadget to preserve the d-NI property of the sampling procedure. The masked routine for the acceptance bit sampling is presented in Algorithm 6.

Algorithm 6: MaskedBernoulliExp

Input: Array of Boolean shares $[\boldsymbol{p}^{\mathbf{exp}}{}_i]_{0\le i\le d} = ([p_i^0]_{0\le i\le d}, \ldots, [p_i^{l-1}]_{0\le i\le d})$ where each $[p_i^j]_{0\le i\le d} = ((b_i^0)_{0\le i\le d}, \ldots, (b_i^{\mu-1})_{0\le i\le d})$, Boolean shares $(\mathrm{one}_i)_{0\le i\le d}$ of 1, $(u_i)_{0\le i\le d}$ of an integer u where $u = \sum_{j=0}^{l-1} u_j 2^j$ with $u_j \in \{0,1\}$.

Output: Boolean shares of the acceptance bit $(b_i)_{0\le i\le d}$.

1: $(b_i)_{0\le i\le d} \leftarrow (\mathrm{one}_i)_{0\le i\le d}$
2: **for** $j \leftarrow 0$ **to** $l-1$ **do**
3: $\quad (b'_i)_{0\le i\le d} \leftarrow \mathrm{MaskedBernoulli}([p_i^j]_{0\le i\le d}, (\mathrm{one}_i)_{0\le i\le d})$
4: $\quad (u_i^j)_{0\le i\le d} \leftarrow (u_i \gg j)_{0\le i\le d}$
5: $\quad (u_i^j)_{0\le i\le d} \leftarrow \mathtt{Refresh}((u_i^j)_{0\le i\le d})$
6: $\quad (\tilde{u}_i^j)_{0\le i\le d} \leftarrow (\neg u_0^j, u_1^j, \ldots, u_d^j)$
7: $\quad (v_i)_{0\le i\le d} \leftarrow \mathtt{SecOR}((\tilde{u}_i^j)_{0\le i\le d}, (b'_i)_{0\le i\le d})$
8: $\quad (b_i)_{0\le i\le d} \leftarrow \mathtt{SecAND}((b_i)_{0\le i\le d}, (v_i)_{0\le i\le d})$
9: **return** $(b_i)_{0\le i\le d}$

3.5 Masked Discrete Gaussian Sampling

The DGS algorithm of `DP-DGS` is designed to support both very small and large values of the standard deviation σ. However, since MAGNET targets applications that require large σ samplers, we exclude the case where $\sigma < 1$. This allows us to complete the sample generation without requiring any conditional selection. In addition, precomputations of the parameters t and r are performed only for $\sigma \ge 1$. Moreover, we apply the sign bit at

this stage to convert the output of the Laplace sampler into a two-sided discrete Laplace sample, which we intentionally delayed during the earlier sampling phase. In particular, we generate a uniformly random sign bit, negate it, and `XOR` it with the sample s. Adding the sign bit produces a two-sided discrete Laplace sample, eliminating the absolute value computation in $g_*(s)$. The selection of the n accepted samples from the m candidates begins immediately after generating each candidate s and its acceptance bit b. We complete the selection in constant time by performing one `ADD`, one `AND`, one `NEG`, and two `XOR`s.

We start DGS masking by first sampling the one-sided discrete Laplace sample s in masked form. Next, we securely compute $u = g_*(s)$ to be able to perform the acceptance bit sampling. After sampling the acceptance bit b, we convert the masked sample s into a discrete Gaussian sample. For this, we apply the sign bit to s to extend the distribution over both positive and negative domains. Finally, we select the first n masked discrete Gaussian samples in constant time according to their corresponding acceptance bits. We will now present the details of this procedure.

The algorithm starts by generating the masked discrete Laplace sample s. The next step is to compute $g_*(s) = (|s| - z_0)^2$ with $z_0 = \sigma^2/t$. Here, both the subtraction and the squaring must be performed in masked form. Since the masked Laplace sampler produces only non-negative samples and $t = \sigma^2/\lfloor\sigma\rceil$, it is sufficient to perform the subtraction as $s - \lfloor\sigma\rceil$. We precompute the arithmetic negation of $\lfloor\sigma\rceil$ as Boolean masked shares and complete the computation as $s + (-\lfloor\sigma\rceil)$ using the gadget `SecADD` from [BBE+18] based on Kogge-Stone adder. Following the addition, the subsequent squaring step must be completed in masked form. Although earlier studies [RP10, Cor14] proposed secure multiplication gadgets for Boolean shares, these gadgets are designed to work over characteristic-2 fields and their extensions. Because the input to the squaring step $s - \lfloor\sigma\rceil \in \mathbb{N}$, we need a generic squaring circuit. Unfortunately, even after optimizations, a generic squaring circuit built with full n-bit adders still requires many `AND` gates, making Boolean masking prohibitively expensive. To avoid this overhead, we perform the squaring step under arithmetic masking.

Secure conversion between arithmetic and Boolean masks is a widely studied topic, and the literature offers several optimized conversion algorithms. Since the squaring inputs are Boolean-masked, we first securely convert them to arithmetic shares. For first-order masking, we apply Goubin's method [Gou01]; for higher orders, we use the improved recursion-based Boolean-to-Arithmetic conversion algorithm of [BCZ18]. Once the computation shifts to the arithmetic-masking domain, squaring becomes straightforward. We employ the arithmetic gadget `SecMUL` of [SPOG19] based on `SecAND` and simplify it for secure squaring, as detailed in Algorithm 7. Compared to `SecMUL`, `SecSQU` requires $n(n-1)/2$ fewer multiplications on n shares. After performing the squaring with `SecSQU`, we convert the arithmetic shares back to Boolean shares. For first-order masking, we use Goubin's method [Gou01]; for higher orders, we employ the efficient Arithmetic-to-Boolean conversion algorithm of [CGTZ23], which improves upon the method of [CGMZ22] using the ShiftMod gadget. Although conversion gadgets incur higher overhead than simple Boolean gadgets, we invoke them only once per sample; therefore, the cost of switching between masking domains remains negligible in the overall computation.

Having computed $g_*(s)$ in masked form, we can securely sample the acceptance bit using Algorithm 6. Before selecting a candidate sample based on this bit, the sign bit must be applied to the Laplace sample s to produce a two-sided discrete Gaussian sample. We first uniformly generate the masked sign bit sign and compute its unsigned negation using the gadget `SecNEG`. Next, we apply `XOR` between the masked sample s and the negated masked sign bit, and then add the sign bit back to complete the signed adjustment. Since addition is required only when the sign bit is 1, we apply a conditional increment here. For an integer x, we first compute $x + 1$ with `SecINC`. We then create `diffmask` by computing the bitwise difference between $x + 1$ and x using `XOR`. The sign bit is then expanded to a w-bit word using `SecNEG`, forming `signmask`. By performing a `SecAND` between `diffmask`

and `signmask`, we collect the bits that differ according to the sign bit on `selmask`. Finally, `XOR` between x and `selmask` yields the correct selection between x or $x+1$, according to the sign bit. With the sign bit applied, we can proceed to the final step in the masked

Algorithm 7: `SecSQU` (based on `SecMUL` [SPOG19])

Input: Arithmetic shares of $(A)_{0\leq i\leq d} \in \mathbb{Z}_{2^w}$ where $\sum_{j=0}^{d} A_j = a \bmod 2^w$.
Output: Arithmetic shares of $(C)_{0\leq i\leq d} \in \mathbb{Z}_{2^w}$ where $\sum_{j=0}^{d} C_j = a \cdot a \bmod 2^w$.

for $j \leftarrow 0$ **to** d **do**
    $C_j = A_j \cdot A_j \mod 2^w$
for $i \leftarrow 0$ **to** d **do**
    for $j \leftarrow i+1$ **to** d **do**
        $r \leftarrow \mathbb{Z}_{2^w}$ `// uniformly generate a random` w`-bit integer`
        $C_i \leftarrow C_i - r \mod 2^w$
        $C_j \leftarrow C_j + 2 \cdot A_i \cdot A_j + r \mod 2^w$
return $(C)_{0\leq i\leq d}$

DGS procedure, which is to select the n accepted samples from the original m candidates. We perform this step in constant time as well, using as few non-linear operations as possible. Converting the acceptance bit b into the requisite bitmask with `SecNEG` allows us to efficiently collect the accepted samples. Here, we only reveal the acceptance bit b for the increment of the array index. This does not result in any information leakage as the rejection rate of the rejection sampling procedure is already public. We perform the acceptance bit sampling in masked form primarily to prevent leakage of the masked value u used in the procedure. After covering all these details, we present the fully masked DGS in Algorithm 8. Note that the statistical bound established in Theorem 4.4 of [WYF+23] also applies to Algorithm 8.

4 Security Proofs and Analysis

This section presents the d-probing security proofs for our gadgets and masked sampling algorithms. All proofs follow the standard simulation-based paradigm. We model the adversary probes and construct each proof backward from the output to the secret. To increase the clarity of the proofs, we omit the public constants in the algorithms. Our approach closely follows the techniques of [CC24].

Theorem 1. *The gadget* `SecNEG` *(Algorithm 3) is d-SNI secure.*

Proof. In Algorithm 3, the only non-linear step is a single `Refresh` call. The logical `AND` with a constant and two's complement negation are affine operations, which are performed share by share. Let $\mathcal{P}_1, \mathcal{P}_2, \mathcal{O}$ be the set of probes in `Refresh`, affine operations, and output shares (z_i), respectively. Also, let the corresponding simulation set on `Refresh` be $\mathcal{S}_1$, and on the affine block be $\mathcal{S}_2$. By d-SNI security of `Refresh`, there exists a simulation set $\mathcal{S}_1$ with $|\mathcal{S}_1| \leq |\mathcal{P}_1|$. Since the affine block is d-NI, the simulation set $\mathcal{S}_2$ on it satisfies $|\mathcal{S}_2| \leq |\mathcal{P}_2| + |\mathcal{S}_1| \leq |\mathcal{P}_2| + |\mathcal{P}_1|$ for the input shares (x_i), which never exceeds d_I. Because the required number of shares from the input (x_i) is bounded strictly by the internal probes and is completely independent of $|\mathcal{O}|$, `SecNEG` is d-SNI secure. □

Theorem 2. *The gadget* `SecINC` *(Algorithm 4) is d-SNI secure.*

Proof. We use Figure 1 for the demonstration of the proof. We start by first proving the security of the loop. Let us fix an iteration $j \in \{1, \ldots, W\}$ of Algorithm 4. Let an adversary place at most d_I internal probes $\mathcal{P}^j_{iter} = \bigcup_{i=1}^{4} \mathcal{P}^j_i$ on the gadgets of iteration j.

Algorithm 8: MAGNET

Input: Gaussian parameter σ, sample number n, acceptance probability p_*, security parameter λ.

Output: An array of Boolean shares of accepted discrete Gaussian samples $[samps_i]_{0\le i\le d}$.

Initialize $[samps_i]_{0\le i\le d}$ as a Boolean shares array of size n with zeros
$k \leftarrow 0$
for $j \leftarrow 0$ **to** $m-1$ **do**
  $(s_i^j)_{0\le i\le d} \leftarrow \text{MaskedLaplace}([p_i^{lap}]_{0\le i\le d}, [\boldsymbol{p}_i^{\mathbf{geo}}]_{0\le i\le d}, (\text{one}_i)_{0\le i\le d})$
  $(v_i)_{0\le i\le d} \leftarrow \texttt{SecADD}((s_i^j)_{0\le i\le d}, (-\sigma_i)_{0\le i\le d})$
  $(V_i)_{0\le i\le d} \leftarrow \texttt{SecB2A}((v_i)_{0\le i\le d})$
  $(U_i)_{0\le i\le d} \leftarrow \texttt{SecSQU}((V_i)_{0\le i\le d})$
  $(u_i)_{0\le i\le d} \leftarrow \texttt{SecA2B}((U_i)_{0\le i\le d})$
  $(b_i)_{0\le i\le d} \leftarrow \text{MaskedBernoulliExp}([\boldsymbol{p}_i^{\mathbf{exp}}]_{0\le i\le d}, (u_i)_{0\le i\le d}, (\text{one}_i)_{0\le i\le d})$
  $\mathbf{r}_{d+1} \leftarrow \texttt{GenUniRandBit}()$ `// Generates` $d+1$ `uniformly random bits`
  $(\text{sign}_i)_{0\le i\le d} \leftarrow \texttt{Refresh}(\mathbf{r}_{d+1})$
  $(\texttt{signmask}_i)_{0\le i\le d} \leftarrow \texttt{SecNEG}((\text{sign}_i)_{0\le i\le d})$
  $(s_i^j)_{0\le i\le d} \leftarrow (s_i^j \oplus \texttt{signmask}_i)_{0\le i\le d}$
  $(\hat{s}_i^j)_{0\le i\le d} \leftarrow \texttt{SecINC}((s_i^j)_{0\le i\le d})$
  $(\texttt{diffmask}_i)_{0\le i\le d} \leftarrow (\hat{s}_i^j \oplus s_i^j)_{0\le i\le d}$
  $(\texttt{selmask}_i)_{0\le i\le d} \leftarrow \texttt{SecAND}((\texttt{diffmask}_i)_{0\le i\le d}, (\texttt{signmask}_i)_{0\le i\le d})$
  $(t_i)_{0\le i\le d} \leftarrow (s_i^j \oplus \texttt{selmask}_i)_{0\le i\le d}$
  $(b_i')_{0\le i\le d} \leftarrow \texttt{SecNEG}((b_i)_{0\le i\le d})$
  $(t_i)_{0\le i\le d} \leftarrow \texttt{SecAND}((t_i)_{0\le i\le d}, (b_i')_{0\le i\le d})$
  $(samps_i^k)_{0\le i\le d} \leftarrow (samps_i^k \oplus t_i)_{0\le i\le d}$
  $b \leftarrow \texttt{FullXOR}((b_i)_{0\le i\le d})$
  $k \leftarrow \texttt{ADD}(k, b)$
return $[samps_i]_{0\le i\le d} = ((s_i^0)_{0\le i\le d}, \ldots, (s_i^{n-1})_{0\le i\le d})$

The probes target, respectively, the outputs of the `Refresh` ($\mathcal{P}_1^j$), the affine $(\wedge, \oplus)$ block ($\mathcal{P}_2^j$), the `SecAND` ($\mathcal{P}_3^j$) and the left-shift ($\mathcal{P}_4^j$). We denote the matching simulation sets by $\mathcal{S}_1, \mathcal{S}_2^{1,2}, \mathcal{S}_3^{1,2}, \mathcal{S}_4$, respectively. In addition, let $\mathcal{O}$ be the set of probes on the final output shares (z_i). Starting from the end of the iteration, `Refresh` is a d-SNI gadget, which satisfies $|\mathcal{S}_1| \le |\mathcal{P}_1^j|$. The affine block $(\wedge, \oplus)$ consists of linear operations and is performed share by share. Thus, it is d-NI and $|\mathcal{S}_2^1|, |\mathcal{S}_2^2| \le |\mathcal{P}_2^j| + |\mathcal{S}_1| \le |\mathcal{P}_2^j| + |\mathcal{P}_1^j|$. Proceeding to the left, the d-SNI `SecAND` gadget ensures $|\mathcal{S}_3^1|, |\mathcal{S}_3^2| \le |\mathcal{P}_3^j|$. Then, we have the d-NI left-shift, providing $|\mathcal{S}_4| \le |\mathcal{P}_4^j| + |\mathcal{S}_3^1| \le |\mathcal{P}_4^j| + |\mathcal{P}_3^j|$. Additionally, for the simulation sets of the input (g_i) for an iteration $\mathcal{S}_{iter}^j = \mathcal{S}_4 \cup \mathcal{S}_3^2 \cup \mathcal{S}_2^2$, we have $|S_{iter}^j| \le |\mathcal{P}_4^j| + |\mathcal{P}_3^j| + |\mathcal{P}_2^j| + |\mathcal{P}_1^j| = |\mathcal{P}_{iter}^j| \le d_I$. All the inequalities show that iteration j of the loop is d-SNI. Next, across the W iterations, the adversary can control at most $d_I \le d$ probes in total, and at least one iteration j^* is left unprobed. The `Refresh` of iteration j^* provides fresh randomness to its outputs; therefore, one can simulate every probe that appears after j^* with no more than d_I probes. For the iterations before j^*, the sets $\mathcal{S}_4, \mathcal{S}_2^2$ can be simulated with the same number of input shares, since affine $(\ll), (\wedge, \oplus)$ operations are done share by share. The set $\mathcal{S}_3^2$ of (g_i) can only be simulated with at most the number of input shares since it is input to `SecAND`, which is d-SNI. Thus, whole loop is d-SNI secure. After the loop, we have an affine $(\ll, \oplus)$ block and a final `Refresh`, with internal probe sets $\mathcal{P}_5$, $\mathcal{P}_6$, and the related simulating sets $\mathcal{S}_5^{1,2}, \mathcal{S}_6$, respectively. Because the final `Refresh` is d-SNI, it perfectly isolates the output probes. Consequently, to simulate $\mathcal{O}$ and $\mathcal{P}_6$, the required input set $\mathcal{S}_6$ is bounded strictly by its internal probes: $|\mathcal{S}_6| \le |\mathcal{P}_6|$. Proceeding to the left, the d-NI affine block $(\ll, \oplus)$ requires simulation sets $\mathcal{S}_5^1$ for (g_i) and $\mathcal{S}_5^2$ for (x_i). Due to the d-NI property, both sets

are bounded by $|\mathcal{S}_5^1|, |\mathcal{S}_5^2| \leq |\mathcal{P}_5| + |\mathcal{S}_6| \leq |\mathcal{P}_5| + |\mathcal{P}_6|$. The required shares from both the input (x_i) and the d-SNI loop output (g_i) are bounded strictly by internal probes and remain independent of $|\mathcal{O}|$. Thus, we conclude that `SecINC` is d-SNI secure. □

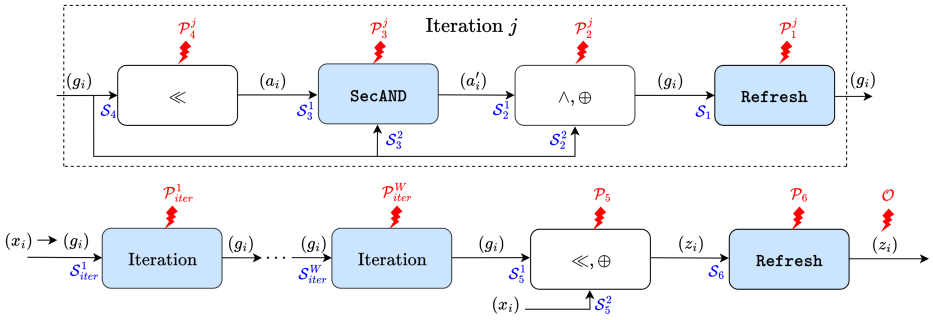


Figure 1: A graphical representation of the `SecINC` gadget. The blue and white gadgets are proved d-SNI and d-NI, respectively. Probing sets $\mathcal{P}_i^j$ for some i, j are given in red. Simulation sets $\mathcal{S}_i$ and $\mathcal{S}_i^j$ for some i, j are given in blue.

Theorem 3. *The gadget* `SecSQU` *(Algorithm 7) is d-SNI secure.*

Proof. The gadget `SecSQU` is structurally identical to `SecMUL` [SPOG19]. The only change is to replace the second input $(B_i)_{0\leq i\leq d}$ with the first input share $(A_i)_{0\leq i\leq d}$. Since `SecMUL` has already been proved d-SNI secure by showing, for each randomization and masked arithmetic operation, probe sets $\mathcal{P}$ and simulation sets $\mathcal{S}$ with $|\mathcal{S}| \leq |\mathcal{P}|$, the exact same simulation carries over when $(B_i) = (A_i)$. Thus, the gadget `SecSQU` is d-SNI secure. □

Theorem 4. *The masked Bernoulli sampling (Algorithm 1) is d-NI secure.*

Proof. We provide the security proof of MaskedBernoulli based on Figure 2. The security of the algorithm is determined by the security of its j-th iteration. The algorithm is a serial composition of μ iterations. We denote the probes on the final $\oplus$ with $\mathcal{P}_1^j$, on `SecAND` with $\mathcal{P}_2^j$, on the affine block $(\oplus, \neg, \oplus)$ with $\mathcal{P}_3^j$, and on `Refresh` with $\mathcal{P}_4^j$. We have the associated simulation sets $\mathcal{S}_1^{1,2}, \mathcal{S}_2^{1,2}, \mathcal{S}_3^{1,2}, \mathcal{S}_4$, respectively. Starting from the end of iteration j, $\oplus$ is affine; therefore, sets $\mathcal{S}_1^1$ and $\mathcal{S}_1^2$ of shares of (x_i) and (v_i), respectively, can be used to simulate $\mathcal{P}_1^j$. By the d-SNI security of `SecAND`, $|\mathcal{S}_2^1|, |\mathcal{S}_2^2| \leq |\mathcal{P}_2^j|$. Then, in the affine block $(\oplus, \neg, \oplus)$, each operation is performed share by share, which satisfies the d-NI property and $|\mathcal{S}_3^1|, |\mathcal{S}_3^2| \leq |\mathcal{P}_3^j| + |\mathcal{S}_2^1| + |\mathcal{S}_2^2|$. Lastly, `Refresh` is d-SNI and is fed by the $d+1$ fresh bits $\mathbf{r}_{d+1}$ produced by `GenUniRandBit`. Since at most d of those bits can be probed, this ensures $|\mathcal{S}_4| \leq |\mathcal{P}_4^j|$. Each iteration has two secret inputs: $(x_i), \mathbf{r}_{d+1}$. For the first iteration, (x_i) is assigned to (one_i), which is a public constant. However, for subsequent iterations x_i becomes a secret and therefore needs to be secured. For the shares $\mathcal{S}_1^1$, $\mathcal{S}_3^1$ of (x_i), we have $|\mathcal{S}_1^1| \leq |\mathcal{P}_1^j|$, $|\mathcal{S}_3^1| \leq |\mathcal{P}_3^j| + |\mathcal{S}_2^1| + |\mathcal{S}_2^2| \leq |\mathcal{P}_3^j| + |\mathcal{P}_2^j|$, respectively. Therefore, $|\mathcal{S}_1^1 \cup \mathcal{S}_3^1| \leq |\mathcal{P}_3^j| + |\mathcal{P}_2^j| + |\mathcal{P}_1^j| \leq d$. For the array $\mathbf{r}_{d+1}$, the $d+1$ uniformly random bits in it are generated from `GenUniRandBit`. To construct the secret bit, the adversary needs $d+1$ of the bits as $r = r_0 \oplus r_1 \oplus \cdots \oplus r_d$, and can probe at most d bits. Therefore, we do not need to prove the security of this function. However, we treat the output of it as a secret input because they collectively build a masked variable. For input $\mathbf{r}_{d+1}$, at most $|\mathcal{S}_4| \leq |\mathcal{P}_4^j| \leq d$ shares can be used to simulate all intermediate probes, due to the d-SNI security of `Refresh`. Taking into account all these points, all probes in iteration j can be simulated at most with the number of input shares in iteration. Thus, iteration j satisfies d-NI security. Since d-NI security is preserved under sequential composition, the entire loop remains d-NI. Consequently, MaskedBernoulli is d-NI secure. □

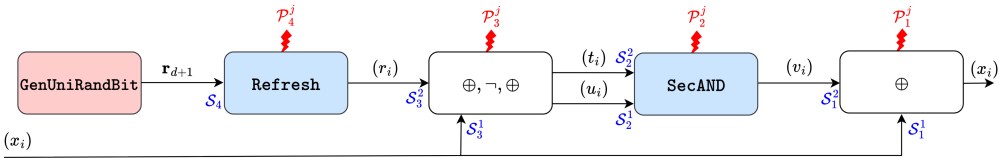


Figure 2: A graphical representation of the MaskedBernoulli algorithm. The blue (resp. white, red) gadgets are proved d-SNI (resp. d-NI, unmasked). Probing sets $\mathcal{P}_i^j$ for some i, j are given in red. Simulation sets $\mathcal{S}_i$ and $\mathcal{S}_i^j$ for some i, j are given in blue.

Theorem 5. *The masked geometric sampling (Algorithm 2) is d-NI secure.*

Proof. Since the structures of MaskedGeometric and MaskedBernoulli are quite similar, the proof follows the same pattern. We use Figure 3 to prove the security of the algorithm. We fix an iteration j. The adversary may probe set $\mathcal{P}_1^j$ in `Refresh`, set $\mathcal{P}_2^j$ in affine $\oplus$, set $\mathcal{P}_3^j$ in the left shift and set $\mathcal{P}_4^j$ in MaskedBernoulli. Let the corresponding simulation sets be $\mathcal{S}_1$, $\mathcal{S}_2^{1,2}$, $\mathcal{S}_3$ and $\mathcal{S}_4$, respectively. Although the random bits $\mathbf{r}_{d+1}$ are generated by MaskedBernoulli, we illustrate them as a secret input for clarity. Starting from the end of iteration j, `Refresh` is d-SNI secure with $|\mathcal{S}_1| \leq |\mathcal{P}_1^j|$. Next, we have the linear $\oplus$, which is d-NI with $|\mathcal{S}_2^1|, |\mathcal{S}_2^2| \leq |\mathcal{P}_2^j| + |\mathcal{S}_1| \leq |\mathcal{P}_2^j| + |\mathcal{P}_1^j|$. Input shares (t_i) are created share by share using the affine left-shift block, making it d-NI with $|\mathcal{S}_3| \leq |\mathcal{P}_3^j| + |\mathcal{S}_2^2| \leq |\mathcal{P}_3^j| + |\mathcal{P}_2^j| + |\mathcal{P}_1^j|$. Finally, MaskedBernoulli is a d-NI gadget. Simulating its internal probes and the required output set $\mathcal{S}_3$ demands a simulation set on the fresh random bits $\mathbf{r}_{d+1}$ bounded by $|\mathcal{S}_4| \leq |\mathcal{P}_4^j| + |\mathcal{S}_3| \leq |\mathcal{P}_4^j| + |\mathcal{P}_3^j| + |\mathcal{P}_2^j| + |\mathcal{P}_1^j|$. For the inputs of the iteration, (x_i) requires at most $|\mathcal{S}_2^1| \leq |\mathcal{P}_2^j| + |\mathcal{P}_1^j|$ shares, and the fresh randomness $\mathbf{r}_{d+1}$ requires at most $|\mathcal{S}_4| \leq |\mathcal{P}_4^j| + |\mathcal{P}_3^j| + |\mathcal{P}_2^j| + |\mathcal{P}_1^j|$ shares. Given that the sizes of both input simulation sets are strictly bounded by d_I, iteration j achieves d-NI security. Since d-NI security is preserved under sequential composition, MaskedGeometric is d-NI secure. □

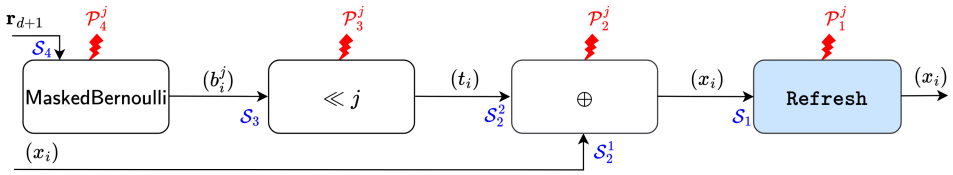


Figure 3: A graphical representation of the MaskedGeometric algorithm. The blue and white gadgets are proved d-SNI and d-NI, respectively. Probing sets $\mathcal{P}_i^j$ for some i, j are given in red. Simulation sets $\mathcal{S}_i$ and $\mathcal{S}_i^j$ for some i, j are given in blue.

Theorem 6. *The masked Laplace sampling (Algorithm 5) is d-SNI secure.*

Proof. We use Figure 4 to prove the security of MaskedLaplace. Let an adversary place at most $d_I = \sum_{i=1}^{6} |\mathcal{P}_i|$ internal probes, where $\mathcal{P}_i$ denotes the probes on the corresponding gadget, and let $\mathcal{O}$ be the probes on the final output shares (v_i). We show the required simulation sets for the secret inputs are independent of $|\mathcal{O}|$ and bounded strictly by d_I. Starting from the end, the terminal `SecAND` is d-SNI. It perfectly isolates the output probes $\mathcal{O}$, requiring simulation sets $\mathcal{S}_1^1$ on (u_i) and $\mathcal{S}_1^2$ on (t_i) bounded strictly by its internal probes: $|\mathcal{S}_1^1|, |\mathcal{S}_1^2| \leq |\mathcal{P}_1|$. Moving left, both `SecNEG` (upper path) and `SecINC` (lower path) are d-SNI gadgets. Thus, they perfectly absorb the required output sets $\mathcal{S}_1^2$ and $\mathcal{S}_1^1$. Their simulators require sets bounded only by their internal probes: $|\mathcal{S}_3| \leq |\mathcal{P}_3|$ for `SecINC`,

and $|\mathcal{S}_2| \leq |\mathcal{P}_2|$ for `SecNEG`. On the upper path, the bitwise $\neg$ is a d-NI share-by-share operation, requiring a simulation set bounded by $|\mathcal{S}_4| \leq |\mathcal{P}_4| + |\mathcal{S}_2| \leq |\mathcal{P}_4| + |\mathcal{P}_2|$. Further left, the d-NI MaskedBernoulli requires a simulation set on its random bits bounded by $|\mathcal{S}_5| \leq |\mathcal{P}_5| + |\mathcal{S}_4| \leq |\mathcal{P}_5| + |\mathcal{P}_4| + |\mathcal{P}_2| \leq d_I$. On the lower path, the d-NI MaskedGeometric requires a simulation set on its random bits bounded by $|\mathcal{S}_6| \leq |\mathcal{P}_6| + |\mathcal{S}_3| \leq |\mathcal{P}_6| + |\mathcal{P}_3| \leq d_I$. Because the required number of shares from the inputs (the random bit arrays) are bounded entirely by the internal probes ($\leq d_I$) and remain completely independent of $|\mathcal{O}|$, MaskedLaplace is d-SNI secure. □

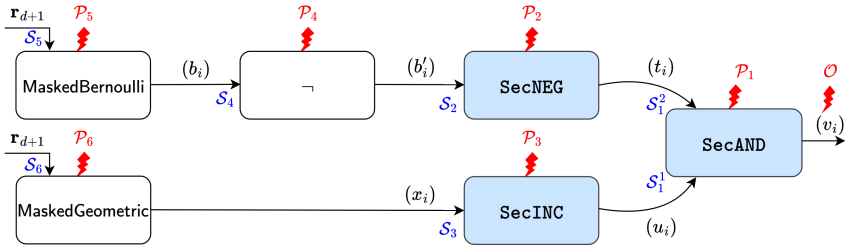


Figure 4: A graphical representation of the MaskedLaplace algorithm. The blue and white gadgets are proved d-SNI and d-NI, respectively. Probing sets $\mathcal{P}_i^j$ for some i, j are given in red. Simulation sets $\mathcal{S}_i$ and $\mathcal{S}_i^j$ for some i, j are given in blue.

Theorem 7. *The masked acceptance bit sampling (Algorithm 6) is d-NI secure.*

Proof. We provide the proof based on Figure 5. The procedure consists only of a loop with l iterations. We follow the same approach used in earlier proofs. To prove the security of an arbitrary iteration j of the loop, we begin with the d-SNI gadget `SecAND`. A probe set $\mathcal{P}_1^j$ on its d internal wires is simulated by two simulation sets $\mathcal{S}_1^1$ on (b_i) and $\mathcal{S}_1^2$ on (v_i), where $|\mathcal{S}_1^1|, |\mathcal{S}_1^2| \leq |\mathcal{P}_1^j|$. Stepping left, the d-SNI gadget `SecOR` is probed with set $\mathcal{P}_2^j$ and simulated by $\mathcal{S}_2^1$ on $(\tilde{u}_i^j)$ and $\mathcal{S}_2^2$ on (b'_i) with $|\mathcal{S}_2^1|, |\mathcal{S}_2^2| \leq |\mathcal{P}_2^j|$. The bitwise $\neg$ gate providing $(\tilde{u}_i^j)$ is affine and therefore only d-NI. Assuming its probe set as $\mathcal{P}_3^j$ and simulation set as $\mathcal{S}_3$, it ensures $|\mathcal{S}_3| \leq |\mathcal{P}_3^j| + |\mathcal{S}_2^1| \leq |\mathcal{P}_3^j| + |\mathcal{P}_2^j|$. Next, the probe set $\mathcal{P}_4^j$ on d-SNI `Refresh` is simulated with the simulation set $\mathcal{S}_4$ satisfying $|\mathcal{S}_4| \leq |\mathcal{P}_4^j|$. The call to MaskedBernoulli generates the random bits $\mathbf{r}_{d+1}$ with the corresponding simulation set $\mathcal{S}_5$ and probing set $\mathcal{P}_5^j$. The masked algorithm is d-NI, thus $|\mathcal{S}_5| \leq |\mathcal{P}_5^j| + |\mathcal{S}_2^2| \leq |\mathcal{P}_5^j| + |\mathcal{P}_2^j|$. Finally, the right-shift operation is affine and d-NI. Assuming its probe set as $\mathcal{P}_6^j$, and the related simulation set on its input shares (u_i) as $\mathcal{S}_6$, we get $|\mathcal{S}_6| \leq |\mathcal{P}_6^j| + |\mathcal{S}_4| \leq |\mathcal{P}_6^j| + |\mathcal{P}_4^j|$. For the inputs of iteration j, we have the random bits $\mathbf{r}_{d+1}$, and shares $(u_i), (b_i)$. The random bits $\mathbf{r}_{d+1}$ can be simulated with $|\mathcal{S}_5| \leq |\mathcal{P}_5^j| + |\mathcal{S}_2^2| \leq |\mathcal{P}_5^j| + |\mathcal{P}_2^j|$. Next, to simulate the input shares (u_i), at most $|\mathcal{S}_6| \leq |\mathcal{P}_6^j| + |\mathcal{S}_4| \leq |\mathcal{P}_6^j| + |\mathcal{P}_4^j|$ shares can be used. Finally, the input shares (b_i) can be simulated with at most $|\mathcal{S}_1^1| \leq |\mathcal{P}_1^j| \leq d_I$ shares. These inequalities show that all probes on the masked gadgets of MaskedBernoulliExp at iteration j can be simulated with no more than an equal number of shares of the inputs for that iteration, thus iteration j of the loop is d-NI secure. By composing the l d-NI iterations, we reach the conclusion that MaskedBernoulliExp is d-NI secure. □

Theorem 8. MAGNET *(Algorithm 8) is d-NIo secure.*

Proof. The structure of MAGNET is also similar to the masked sub-algorithms that construct it. We build the proof based on Figure 6. Let j be an arbitrary iteration. We define the probing sets on each gadget with $\mathcal{P}_i^j$ for $i = 1, \ldots, 17$, and the corresponding

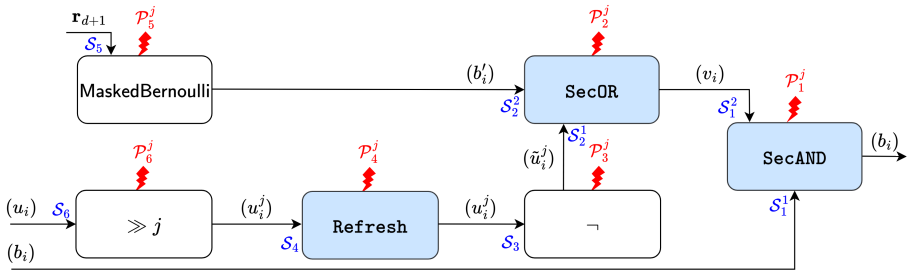


Figure 5: A graphical representation of the MaskedBernoulliExp algorithm. The blue and white gadgets are proved d-SNI and d-NI, respectively. Probing sets $\mathcal{P}_i^j$ for some i, j are given in red. Simulation sets $\mathcal{S}_i$ and $\mathcal{S}_i^j$ for some i, j are given in blue.

simulation sets on the gadgets with $\mathcal{S}_i$ or $\mathcal{S}_i^j$ for $i = 1, \ldots, 17$ and some j. We start from the last gadget of the iteration, `FullXOR`. This gadget, reveals the acceptance bit b and is proven d-NIo. Thus, it is simulated with at most $|\mathcal{S}_1| \leq |\mathcal{P}_1^j|$ shares of input (b_i) and the knowledge of b. The next gadget is affine $\oplus$, which is d-NI and operates share by share. One can use sets $\mathcal{S}_2^1$ and $\mathcal{S}_2^2$ of shares of $(samps_i^k)$ and (t_i), respectively, to simulate $\mathcal{P}_2^j$ while having the knowledge of b. The d-SNI gadgets `SecAND`, `SecNEG` and `SecINC` can be simulated with $|\mathcal{S}_3^{1,2}| \leq |\mathcal{P}_3^j|$, $|\mathcal{S}_4| \leq |\mathcal{P}_4^j|$, $|\mathcal{S}_5| \leq |\mathcal{P}_5^j|$ shares of their related inputs, respectively. Next, linear $\oplus$ gadgets ensure $|\mathcal{S}_6^1|, |\mathcal{S}_6^2| \leq |\mathcal{P}_6^j| + |\mathcal{S}_5| + |\mathcal{S}_7^2| + |\mathcal{S}_{10}^1|$, and $|\mathcal{S}_7^1|, |\mathcal{S}_7^2| \leq |\mathcal{P}_7^j| + |\mathcal{S}_3^1|$ for their inputs and the knowledge of b. Following `SecAND` and `SecNEG` are d-SNI, which guarantee $|\mathcal{S}_8^1|, |\mathcal{S}_8^2| \leq |\mathcal{P}_8^j|$ and $|\mathcal{S}_9| \leq |\mathcal{P}_9^j|$, respectively. For the d-NI $\oplus$ gadget, we get $|\mathcal{S}_{10}^1|, |\mathcal{S}_{10}^2| \leq |\mathcal{P}_{10}^j| + |\mathcal{S}_8^1|$ shares of its inputs and the knowledge of b. Subsequently, the d-SNI `Refresh` is fed with the random bits array $\mathbf{r}_{d+1}^1$, which satisfies $|\mathcal{S}_{11}| \leq |\mathcal{P}_{11}^j|$. MaskedBernoulliExp is proven d-NI in Theorem 7, therefore, it ensures $|\mathcal{S}_{12}^1|, |\mathcal{S}_{12}^2| \leq |\mathcal{P}_{12}^j| + |\mathcal{S}_1| + |\mathcal{S}_4|$ random inputs and the knowledge of b. Proceeding, we have the d-NI proven `SecA2B` gadget, which is simulated with at most $|\mathcal{S}_{13}| \leq |\mathcal{P}_{13}^j| + |\mathcal{S}_{12}^1|$ shares of (U_i) and the knowledge of b. `SecSQU` and `SecB2A` are d-SNI secure, and satisfy $|\mathcal{S}_{14}| \leq |\mathcal{P}_{14}^j|$ and $|\mathcal{S}_{15}| \leq |\mathcal{P}_{15}^j|$, respectively. `SecADD` is a d-NI secure non-linear gadget that guarantees having at most $|\mathcal{S}_{16}| \leq |\mathcal{P}_{16}^j| + |\mathcal{S}_{15}|$ shares of (s_i^j) and the knowledge of b. Finally, MaskedLaplace algorithm is proven d-SNI in Theorem 6, which ensures $|\mathcal{S}_{17}| \leq |\mathcal{P}_{17}^j|$. As the inputs of iteration j, we have the input shares $(samps_i^k)$ and uniformly random bit arrays $\mathbf{r}_{d+1}^i$ for $i = 1, \ldots, 3$. The input shares $(samps_i^k)$ are simulated with at most $|\mathcal{S}_2^1| \leq |\mathcal{P}_2^j|$ shares and the knowledge of b. The input arrays $\mathbf{r}_{d+1}^1$ and $\mathbf{r}_{d+1}^3$ can be simulated with their corresponding sets, which satisfy $|\mathcal{S}_{11}| \leq |\mathcal{P}_{11}^j|$ and $|\mathcal{S}_{17}| \leq |\mathcal{P}_{17}^j|$, respectively. The input array $\mathbf{r}_{d+1}^2$ can be simulated with at most $|\mathcal{S}_{12}^2| \leq |\mathcal{P}_{12}^j| + |\mathcal{S}_4| + |\mathcal{S}_1| \leq |\mathcal{P}_{12}^j| + |\mathcal{P}_4^j| + |\mathcal{P}_1^j|$ shares of its inputs and the knowledge of b. One can also verify that no gadget requires simulating more than d shares. We show this on the most complex case, with the probing set $\mathcal{P}_6^j$ on $\oplus$. Here, $\mathcal{S}_6^1 \cup \mathcal{S}_6^2$ should simulate $\mathcal{P}_6^j$ and $\mathcal{S}_5 \cup \mathcal{S}_7^2 \cup \mathcal{S}_{10}^1$. For this, we get $|\mathcal{S}_5| \leq |\mathcal{P}_5^j|$ shares of (s_i^j), $|\mathcal{S}_7^2| \leq |\mathcal{P}_7^j| + |\mathcal{S}_3^1| \leq |\mathcal{P}_7^j| + |\mathcal{P}_3^j|$ shares of (s_i^j) with the knowledge of b, and $|\mathcal{S}_{10}^1| \leq |\mathcal{P}_{10}^j| + |\mathcal{S}_8^1| \leq |\mathcal{P}_{10}^j| + |\mathcal{P}_8^j|$ shares of (s_i^j) with the knowledge of b. Therefore, the total number of shares $\mathcal{S}_6^1 \cup \mathcal{S}_6^2$ needs to simulate is less than $|\mathcal{P}_6^j| + |\mathcal{P}_5^j| + |\mathcal{P}_7^j| + |\mathcal{P}_3^j| + |\mathcal{P}_{10}^j| + |\mathcal{P}_8^j|$ and the knowledge of b, which is ensured to be $\leq d$, and the iteration j is d-NIo with public output b. Since one iteration is d-NIo, by composition, all probes across m iterations can be simulated with no more than the number of input shares and the knowledge of b, proving MAGNET is d-NIo secure. □

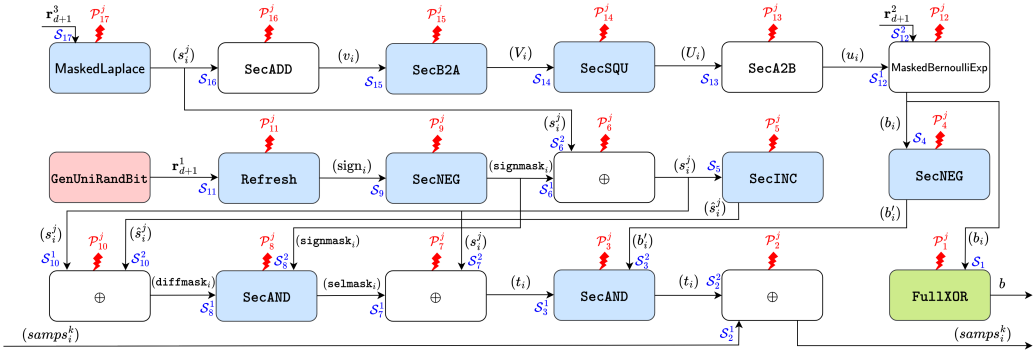


Figure 6: A graphical representation of MAGNET. The blue (resp. white, green, red) gadgets are proved d-SNI (resp. d-NI, d-NIo, unmasked). Probing sets $\mathcal{P}_i^j$ for some i, j are given in red. Simulation sets $\mathcal{S}_i$ and $\mathcal{S}_i^j$ for some i, j are given in blue.

5 Implementation and Evaluation

This section presents the experimental evaluation of the implementation of MAGNET with respect to performance and practical security. We first report performance results for the higher-order masked C implementation and the first-order Thumb-2 assembly implementation targeting the ARM Cortex-M4 platform. We then assess the practical security of the Cortex-M4 implementation through leakage evaluation on embedded hardware. Finally, we discuss the integration of MAGNET into the efficient post-quantum blockchain payment protocol MatRiCT$^+$ [ESZ22b].

5.1 Performance Evaluation

5.1.1 Higher-Order C Implementation

We implemented MAGNET in C to work at any masking order. For the masking gadgets required in the implementation, apart from those that we built, we make use of established constructions from the literature, as mentioned in Table 2. We adopt the Boolean-to-Arithmetic conversion gadgets of [BCZ18] and the Arithmetic-to-Boolean conversion gadgets of [CGTZ23] to perform squaring with arithmetic masking. In order to employ the bit-by-bit binary approximation technique of [DKM$^+$06] in bitwise Bernoulli sampling, we represent each precomputed probability value in μ-bit binary form and perform bitwise operations easily.

Masked implementations are particularly sensitive to the choice of randomness source due to frequent invocations of the `Refresh` gadget. To keep our benchmarks consistent, we adopt the 32-bit xoshiro128** pseudo-random number generator (PRNG) [BV21], which is used by [GR19] in their masked implementation. We emphasize that xoshiro128** is not cryptographically secure and is used here solely for experimental evaluation. In deployments that require provable security, randomness should be generated using well-studied, lattice-compatible cryptographically secure PRNGs such as SHAKE or AES.

The parameters used in MAGNET align with the settings in [WYF$^+$23]. To keep the algorithms clear, we omit the computation of parameters in the paper. The only fixed constant that we use in the implementation is p_*, the acceptance probability of rejection sampling. We set $p_* = 0.76$, as this is the acceptance probability value achieved by the DGS construction in the case of large σ. All other parameters in the implementation are adjustable based on security and performance requirements. The masking word size is set to 32 bits, a commonly adopted choice in masked lattice-based implementations. This can

be increased for scenarios that require higher masking security. Similarly, the masking order d, the security parameter λ, the standard deviation σ, and the number of discrete Gaussian samples n can be adjusted based on the specific needs of the application.

Performance results. We evaluated the performance of MAGNET by comparing it with the publicly available implementation of [GR19]. Although some studies have considered masking the discrete Gaussian sampler of FALCON [BPT+25a, BPT+25b], these implementations are limited to very small standard deviations (up to $\sigma_{\max} = 1.8205$) and target SamplerZ, which relies on floating-point arithmetic. [BPT+25a] proposed a masked implementation of SamplerZ. Their reported evaluation indicates that the design is not optimized and, even at first order, incurs an overhead of roughly 3600×. [EFG+22] presented a masked hybrid discrete Gaussian sampler, but did not provide an implementation or accompanying evaluation. Consequently, the CDT-based sampler of [GR19] remains, to our knowledge, the most efficient masked DGS implementation available and thus constitutes the most appropriate baseline for comparison.

The ARM Cortex-M4 platform is well suited for the deployment of MAGNET, and our implementation executes efficiently on this architecture. However, the [GR19] sampler relies on 128-bit operations when masking values retrieved from the CDT. As the Cortex-M4 architecture does not natively support 128-bit arithmetic, a direct performance comparison on this platform is not feasible. Therefore, to have a fair comparison between MAGNET and [GR19] sampler, we conduct the experiments on a Raspberry Pi 5 equipped with a Cortex-A76 processor clocked at 2.4 GHz, with 16 GB of RAM, running Raspberry Pi OS. On Raspberry Pi 5, we compiled the implementation using `gcc` targeting ARMv8-A architecture with the optimization flag `-O3`. In particular, we perform run-time comparisons on various masking orders and standard deviation values. Since [GR19] integrates their masked DGS into the masked qTESLA-I scheme [ABB+20], we configure both implementations to operate at the same security level (table-entry precision in [ABB+20]) of $\lambda = 64$. Following the original construction of qTESLA, [GR19] uses a CDT optimized for a fixed value σ. As our evaluation covers a range of σ values, we reconstruct the CDT for larger σ values using the method described in [GR19], and perform our experiments accordingly. Figure 7 presents the performance comparison between MAGNET and the CDT-based masked sampler of [GR19] for masking of first, second, and third order across increasing σ values. At $\sigma = 256$, MAGNET achieves speed-ups of 17.12× for first-order masking and 10.54× for second-order masking over the sampler of [GR19]. The performance gap widens dramatically as σ increases, indicating that MAGNET remains highly efficient for lattice-based schemes that require large σ parameters.

We also assess the overhead introduced by masking through a comparison between MAGNET and the unmasked `DP-DGS` baseline. Figure 8 presents the run-time comparison for masking orders 1 to 5 in various values of σ, for $n = 1000$ samples. The performance measurements of the masking overhead were carried out on an ARM Cortex-M4 processor operating at 64 MHz. Performance is evaluated for the security parameter $\lambda \in \{64, 128\}$, and the results indicate that the impact of increasing λ on the efficiency of our construction scales linearly. On the ARM Cortex-M4 CPU, for $\lambda = 64$ and $n = 1000$ at $\sigma = 1024$, masking of order-1 incurs a 4.77× overhead, masking of order-2 10.91× overhead, and masking of order-3 19.84× overhead. For $\lambda = 128$ under the same settings, the corresponding overheads are 4.79×, 10.88×, and 19.81×, respectively.

5.1.2 First-Order ARM Cortex-M4 Assembly Implementation

In addition to the C implementation, we implemented a first-order masked version of MAGNET in Thumb-2 assembly targeting the ARM Cortex-M4 architecture. The Cortex-M4 experiments were conducted on the nRF52840 Development Kit (DK) running Zephyr OS v4.1.99, which features an ARM Cortex-M4F processor clocked at 64 MHz with 256

σ	Order d	[GR19]	MAGNET	Speed-up
4	1	0.10	0.17	0.59×
	2	0.22	0.62	0.35×
	3	0.43	1.02	0.42×
16	1	0.37	0.22	1.68×
	2	0.83	0.81	1.02×
	3	1.61	1.34	1.20×
64	1	1.44	0.28	5.14×
	2	3.23	1.01	3.20×
	3	6.26	1.67	3.75×
256	1	5.65	0.33	17.12×
	2	12.65	1.20	10.54×
	3	24.50	1.99	12.31×
1024	1	22.14	0.39	56.77×
	2	49.67	1.40	35.48×
	3	96.09	2.31	41.59×

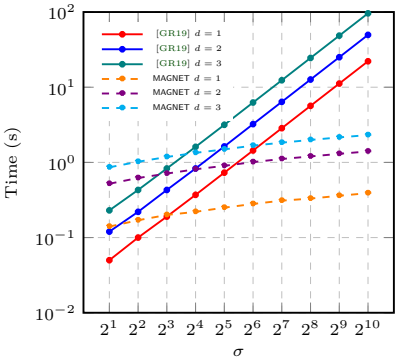


Figure 7: Masked DGS run-times (in seconds) for MAGNET and the CDT-based sampler of [GR19], with $n = 10000, \lambda = 64$, across increasing standard deviations and masking orders. The left table reports absolute timings and speed-ups of our work over [GR19], and the right plot visualizes the run-times. Each entry indicates the median of 100 independent timing measurements on an ARM Cortex-A76 CPU at 2.4GHz.

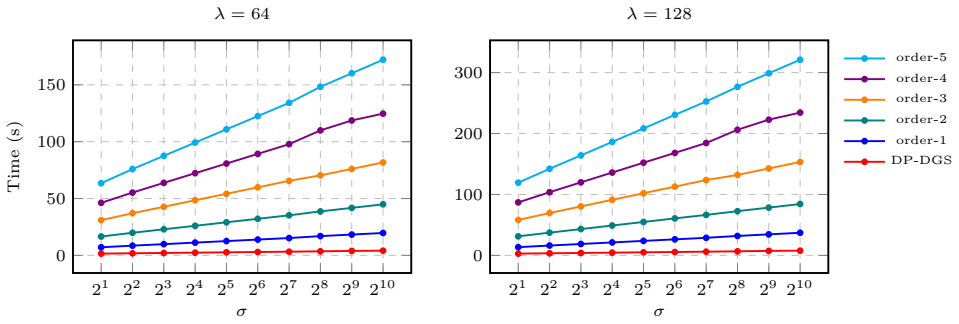


Figure 8: Performance comparison of unmasked (`DP-DGS`) and order-1 through order-5 masked DGS (MAGNET) runs for $n = 1000$ across varying σ values, $\lambda = 64$ on the left and $\lambda = 128$ on the right. Each plotted point is the median of 100 independent timing measurements on an ARM Cortex-M4 CPU at 64 MHz.

KB of RAM and 1 MB of on-chip flash memory. The first-order assembly implementation was compiled using `arm-zephyr-eabi-gcc`, while the higher-level C implementation was compiled using `gcc` version 12.2.0. When implementing MAGNET in assembly, a bottom-up approach was adopted, starting from small masking gadgets and composing them to build the masked sampling algorithms, prioritizing efficiency. For the randomness generation, we followed the same approach as in the C implementation and used the xoshiro128** PRNG, which was implemented in assembly without applying any masking countermeasures. The true random number generator (TRNG) provided by the Arm CryptoCell 310 security engine of the nRF52840 DK was used solely for seed generation. All parameters and probability values that depend on n, λ, and σ are precomputed on the caller side in C and accessed by the assembly implementation.

Performance results. The performance of the first-order MAGNET implementation on the ARM Cortex-M4 was evaluated by executing the first-order C and Cortex-M4 implementations under identical parameter settings and execution environment. In our experiments, when compiler optimizations were disabled by enabling `CONFIG_NO_OPTIMIZATIONS=y` in the Zephyr build system, the C implementation exhibited a substantially higher overhead,

Table 4: Cycle counts for generating one discrete Gaussian sample at first-order masking on ARM Cortex-M4 for $\lambda = 64$ and $n = 100$. Results are reported as the median over 1000 iterations across varying values of σ.

σ	4	16	64	256	1024
ASM	484434	644686	798839	952718	1107157
C (-O0)	1781922	2370928	2937483	3503500	4069945
C (-O3)	336394	448724	554412	662800	775999

approximately 3.68×, compared to the assembly implementation. In contrast, enabling compiler optimizations via `CONFIG_COMPILER_OPT="-O3"` allowed the C implementation to generate a single sample approximately 1.43× faster than the assembly implementation. It should be noted that compiler optimization flags do not affect hand-written assembly, and all performance optimizations for our Cortex-M4 implementation were applied manually at the instruction level. This outcome is expected, as the -O3-optimized C implementation benefits from aggressive compiler optimizations such as inlining, register allocation, and instruction scheduling, whereas our Thumb-2 assembly version is a naive implementation that lacks comparable low-level optimizations. We reported the cycle counts required to generate one discrete Gaussian sample for both the Thumb-2 assembly and C implementations across varying values of σ, with $\lambda = 64$ and $n = 100$, in Table 4. For each run, we generated m samples and divided the total cycle count by m to obtain the cycle count for a single sample generation.

5.2 Practical Security Evaluation

Test vector leakage assessment. We used the Test Vector Leakage Assessment (TVLA) methodology [GGJR+11], which is based on Welch's t-test, to analyze the practical side-channel leakage security of MAGNET. TVLA is a standardized statistical approach for detecting side-channel leakage in cryptographic implementations. It relies on the collection of two distinct sets of power traces and the application of statistical analysis to identify potential dependencies between the input data and the measured power consumption. For these two sets, we employed the widely adopted non-specific t-test methodology using the *fixed vs. random* approach. Let S_f denote the fixed trace set with mean μ_f, variance σ_f^2, and cardinality n_f. Similarly, let S_r denote the random trace set with mean μ_r, variance σ_r^2, and cardinality n_r. The t-statistic distinguishing the two sets is given by:

$$t = \frac{\mu_f - \mu_r}{\sqrt{\left(\frac{\sigma_f^2}{n_f} + \frac{\sigma_r^2}{n_r}\right)}}. \tag{5}$$

In this setting, the t-statistic serves as a threshold for detecting side-channel leakage revealed by the test. The TVLA methodology employs a point-wise computation of the t-statistic across the entire time domain of the trace to assess whether two measurement sets originate from populations with identical means. If the calculated statistic falls within the defined threshold, the null hypothesis remains valid. This indicates that the datasets are statistically indistinguishable, thereby confirming the absence of detectable first-order leakage. Selecting a fixed threshold of $|t| = 4.5$ generally provides a confidence level of approximately 0.99999 for verifying the absence of leakage. For our TVLA approach, to properly account for varying dataset sizes, we utilize the adaptive thresholding technique of [DZD+18]. This method dynamically adjusts the rejection criteria as a function of the error rate, which is set to $\alpha = 10^{-5}$, and the number of analog-to-digital converter (ADC) samples captured per trace, rather than relying on a fixed threshold value. Taking into

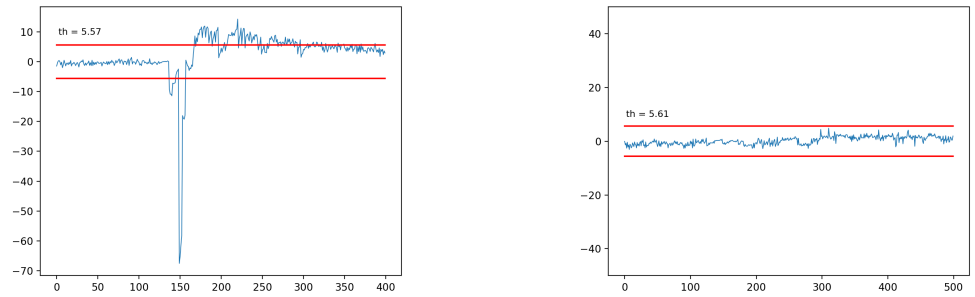
account the total cycle counts of the evaluated gadgets, the corresponding threshold value in our experiments varies between 5.6 and 6.1.

Measurement setup. We perform first-order side-channel leakage measurements using a ChipWhisperer-Lite capture platform [OC14] with a 32-bit STM32F303 target device. The target integrates an ARM Cortex-M4 processor running at up to 72 MHz, with 40 KB of SRAM and 256 KB of on-chip flash memory. To correctly configure the target, we adopt a generic configuration file included in the ChipWhisperer software package. With this setup, we set the external clock at 7.38 MHz. We also set the ADC sampling rate to four times the target clock frequency, resulting in four samples per target clock cycle. To run the tests, we use the TVLA driver script provided by [GG25], which we adapt to meet our experimental requirements. Regarding randomness, since protecting the randomness generation within the gadgets is outside the scope of this work, we perform our evaluations by externally supplying precomputed randomness to the gadgets. We conduct the leakage assessments by collecting 200,000 power traces for each gadget. For all masked gadgets and sampling algorithms, we provide the TVLA measurement scripts for both the naive and hardened implementations in the public repository mentioned earlier.

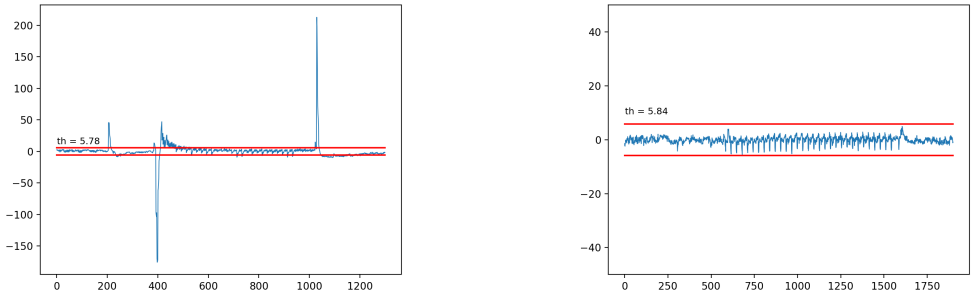
TVLA-guided gadget hardening. Despite the theoretical resistance provided by masking, practical vulnerabilities often emerge upon deployment due to unforeseen micro-architectural leakage. When we implemented MAGNET on the ARM Cortex-M4 in Thumb-2 assembly with first-order masking, we exhibited side-channel leakage in almost every gadget. To mitigate these leakages and protect MAGNET against first-order attacks, we applied a trial-and-error-based hardening strategy on the affected gadgets. Throughout this process, we sought to preserve MAGNET's efficiency as much as possible, and deliberately avoided relying on automated leakage prevention tools from the literature, such as PoMMES [ZM24]. Instead, we adopted a set of carefully selected practical countermeasures and iteratively eliminated individual leakage sources through repeated testing until a leakage-free implementation was obtained. In line with the findings reported in [GG25], we also observed that micro-architectural leakage can be largely mitigated by inserting random load and store operations between critical memory accesses, introducing dummy `XOR` zeroing between dependent ALU operations, and adding carefully placed `NOP` instructions.

Specifically, we applied systematic bus scrubbing to eliminate leakage caused by consecutive transitions of sensitive values on the memory bus. We identified that sequential stores of two output shares caused the data bus to momentarily combine information from both shares, resulting in a direct first-order leakage of the secret. To prevent this effect, we inserted dummy store instructions that write a fixed, non-secret value to the memory between the legitimate output stores. This enforces a transition to a known state on the data bus and removes the correlation between consecutive share-dependent transitions. In addition, we employed register scrubbing by explicitly clearing general-purpose registers immediately after their last use with sensitive data using dummy `XOR` operations, thereby preventing residual register state from influencing subsequent power consumption. We further addressed leakage stemming from the persistence of sensitive data in memory. We observed that deallocating stack space alone was insufficient, as physical memory locations retained secret-dependent values that leaked during subsequent stack pointer updates. In some cases, local stack buffers retained intermediate secret values even after stack deallocation. We mitigated this by explicitly zeroing all such buffers before restoring the stack pointer. After an iterative, measurement-driven hardening process, we eliminated all observable leakage from all gadgets and masked sampling routines. Figure 9 presents the t-test results for the proposed masking gadgets `SecNEG`, `SecINC`, and `SecSQU`, comparing naive assembly implementations (left) with their hardened

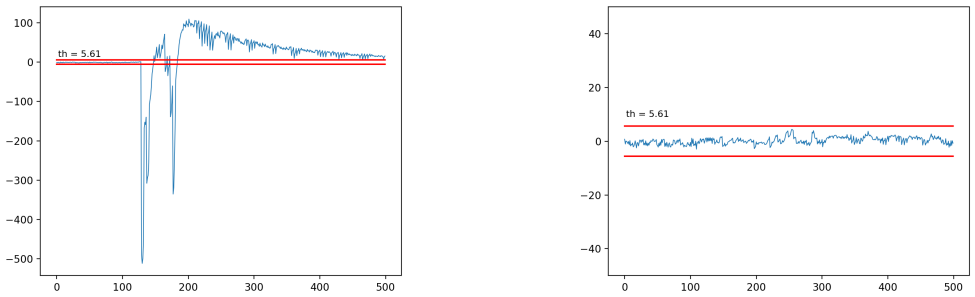
counterparts (right). Similarly, Figure 10 provides the corresponding t-test results for the masked sampling routines MaskedBernoulli, MaskedGeometric, MaskedLaplace, and MaskedBernoulliExp, following the same presentation.



(a) `SecNEG` (Naive) (b) `SecNEG` (Hardened)



(c) `SecINC` (Naive) (d) `SecINC` (Hardened)



(e) `SecSQU` (Naive) (f) `SecSQU` (Hardened)

Figure 9: First-order t-test results for the proposed masking gadgets under fixed vs. random testing, comparing naive (left) and hardened (right) implementations, evaluated over 200,000 traces on an ARM Cortex-M4 CPU.

We evaluated the overhead introduced by hardening on first-order masked gadget implementations. In the experiments, we compared three variants of each gadget: the -O3 optimized C implementation, the naive assembly implementation (ASM_n), and its hardened assembly variant (ASM_h). Our results show that, when compared against the fully optimized C implementation, hardened assembly gadgets incur an overhead of approximately 1.63× to 2.47×. Note that, for all gadgets, randomness is provided externally, and no randomness is generated within the gadget itself. The detailed cycle

Table 5: Cycle counts of first-order masked gadgets for C, naive assembly, hardened assembly implementations. Each measurement corresponds to the median over 10,000 independent executions, measured on an ARM Cortex-M4 CPU.

Gadget	C (-O3)	ASM_n	ASM_h	Overhead (ASM_h/C)
`SecNEG`	32	57	75	2.34×
`SecINC`	279	291	456	1.63×
`SecSQU`	34	38	84	2.47×
MaskedBernoulli	176	279	302	1.72×
MaskedGeometric	430	697	880	2.05×
MaskedLaplace	1033	1392	2201	2.13×
MaskedBernoulliExp	711	1014	1215	1.71×

counts are reported in Table 5. The observed overheads significantly outperform the automated hardening approach of [ZM24], which reports an overhead of approximately 4.7× to 11.88× for first-order masking (see Table 1 in [ZM24]).

In addition to TVLA-guided hardening, we also explored the 'lazy engineering' strategy from [BGG+14] as an alternative approach to achieving practical security. This strategy establishes that transition-based leakages effectively halve the security order of a masking scheme in implementation, hence doubling the order suffices to reach the target security. Although relying on higher-order masking reduces initial development effort, our timing tests for the `SecNEG` gadget demonstrated a severe performance penalty compared to manual assembly hardening. Specifically, the naive order-1 C implementation (with -O3) requires 32 cycles, while our manually hardened order-1 assembly implementation requires 75 cycles, representing a 2.34× overhead. In contrast, the naive order-2 C implementation (with -O3) requires 143 cycles to achieve first-order security, resulting in a higher overhead of 4.46×. Ultimately, this comparison demonstrates a clear trade-off, illustrating that high efficiency requires a necessary sacrifice in development time.

The ChipWhisperer-Lite platform supports a maximum trace length of 24,400 samples, which is insufficient to capture a full iteration of MAGNET even under the smallest practical parameter settings. As a result, we do not perform an end-to-end leakage evaluation of the complete MAGNET sampling procedure. Instead, we carry out leakage assessments for each individual gadget and sub-sampling routine that composes MAGNET, and verify that none exhibits observable leakage. This component-wise evaluation demonstrates the feasibility of a first-order side-channel attack resistant implementation of MAGNET.

5.3 Integration of MAGNET into MatRiCT$^+$

In order to better demonstrate the practicality and efficiency of MAGNET, we integrated it into a post-quantum cryptographic scheme that operates in large σ setting and requires intensive discrete Gaussian sampling. For this purpose, we selected MatRiCT$^+$ [ESZ22b], a lattice-based private payment protocol that employs discrete Gaussian sampling. The integration of a masked Gaussian sampler into MatRiCT$^+$ is motivated by the need to protect the discrete Gaussian sampler, which represents one of the most leakage-sensitive and computationally dominant components of the scheme. In a practical implementation, an unprotected Gaussian sampler represents a realistic attack surface that can enable key-recovery or forgery attacks even when the underlying lattice assumptions remain secure. Therefore, maintaining side-channel resistance without significant performance loss is critical, and the integration of MAGNET into MatRiCT$^+$ directly addresses this issue.

To assess the practicality of this integration, we evaluate the runtime performance of MatRiCT$^+$ on a Linux Mint machine equipped with an Intel i9-14900HX CPU and 32 GB RAM. With the Intel Turbo Boost technology disabled, the CPU ran at the stable clock speed of 2.2GHz. In our evaluation, we replaced the AVX2-based FACCT Gaussian

Table 6: Runtime comparison of the MatRiCT$^+$ spend operation with the MAGNET and [GR19] samplers integrated, evaluated across different masking orders. Reported runtimes are given in seconds and correspond to the median of 50 independent measurements.

Sampler	Order-1	Order-2	Order-3	Order-4
MAGNET	46.93	154.41	260.13	555.31
[GR19]	2724.44	-	-	-

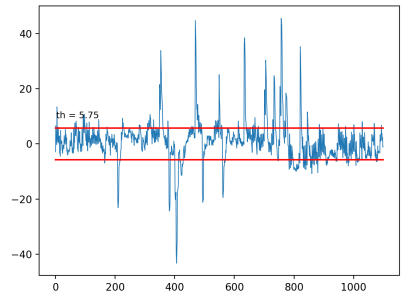
sampler [ZSS20b] used in the 'spend' phase of the protocol, which uses $\sigma = 6390.52$. To establish a comparison baseline and to demonstrate that MAGNET is better suited for large σ, we replaced the FACCT sampler with both MAGNET and [GR19] samplers and analyzed the differences in the resulting performance and behavior. For such a large σ, we were able to run [GR19] sampler only by embedding a precomputed CDT of size 52,694 as a static table. Based on the performance evaluation, a spend operation under the first-order masking setting takes about 2724.44 seconds when using the sampler of [GR19], whereas the same procedure takes about 46.93 seconds with MAGNET, indicating an approximately 58× speedup in masked sample generation. Under second-order masking, the spend operation completes in approximately 154.41 seconds with MAGNET. However, when using the sampler of [GR19], the computation did not terminate even after approximately two hours. Consequently, for masking orders higher than first-order, we were unable to obtain runtime measurements for MatRiCT$^+$ with the sampler of [GR19] integrated. In contrast, we were able to integrate MAGNET into MatRiCT$^+$ without issues and successfully generate masked discrete Gaussian samples, even at order-4 masking. The corresponding runtime results are reported in Table 6. This integration demonstrates that MAGNET can be incorporated into an advanced cryptographic construction, even under large σ settings and without relying on large precomputed tables, highlighting its suitability for practical deployment. We provide a MatRiCT$^+$ implementation integrated with both MAGNET and the sampler of [GR19] in the repository introduced earlier.

6 Conclusion

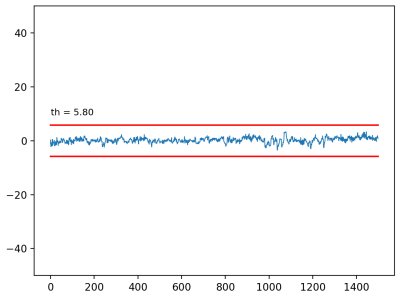
In this work, we present the first masked discrete Gaussian sampler that avoids both precomputed tables and floating-point arithmetic. We propose a new masking design by modifying the method of [WYF+23], and introduced three new gadgets to efficiently apply our design. By implementing MAGNET, we show that having efficient side-channel security in DGS is possible even at large σ settings. We also formally prove the d-probing security of our masking scheme using established techniques in the literature. A performance comparison with the CDT-based masked sampler of [GR19] demonstrates that our table-free design incurs low masking overhead at moderate orders, making it suitable for embedded devices. In addition to the theoretical analysis, we develop a first-order implementation of MAGNET on an ARM Cortex-M4 microcontroller and experimentally assess its resistance to first-order side-channel attacks through TVLA. We iteratively remove the observed leakage at the gadget level. Finally, we deploy MAGNET within MatRiCT$^+$ to validate its applicability in realistic high σ post-quantum settings, showing that efficient and practical protection of discrete Gaussian sampling against side-channel attacks can be achieved in full protocol implementations.

Acknowledgements

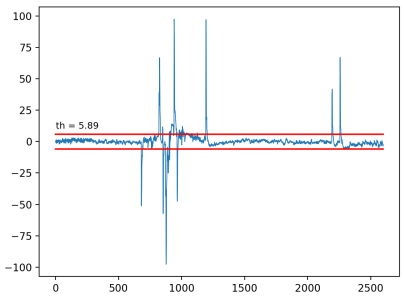
This work was partially supported by Australian Research Council Discovery Grants DP250100229 and DP260100245.



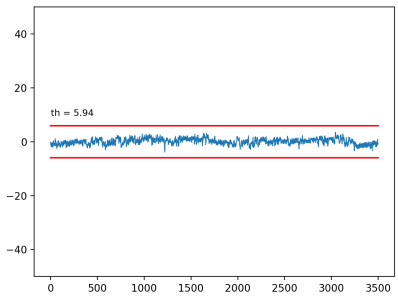
(a) MaskedBernoulli (Naive)



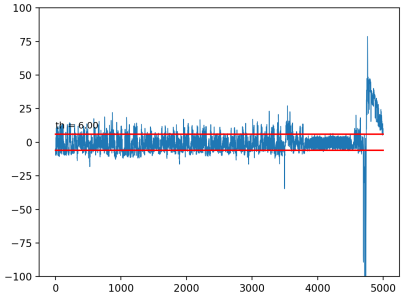
(b) MaskedBernoulli (Hardened)



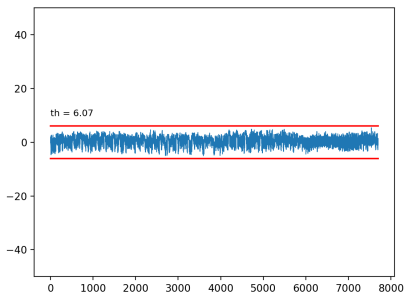
(c) MaskedGeometric (Naive)



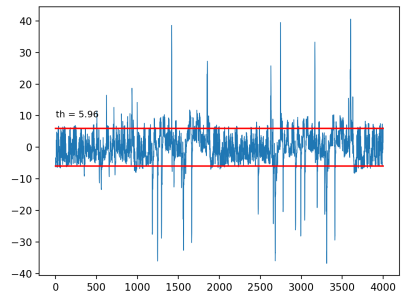
(d) MaskedGeometric (Hardened)



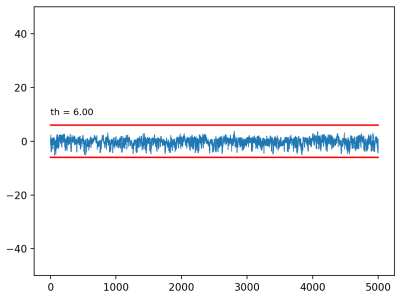
(e) MaskedLaplace (Naive)



(f) MaskedLaplace (Hardened)



(g) MaskedBernoulliExp (Naive)



(h) MaskedBernoulliExp (Hardened)

Figure 10: First-order t-test results for the proposed masked sampling algorithms under fixed vs. random testing, comparing naive (left) and hardened (right) implementations, evaluated over 200,000 traces on an ARM Cortex-M4 CPU.

References

[ABB+20] Erdem Alkim, Paulo S. L. M. Barreto, Nina Bindel, Juliane Krämer, Patrick Longa, and Jefferson E. Ricardini. The lattice-based digital signature scheme qtesla. In *Applied Cryptography and Network Security: 18th International Conference, ACNS 2020, Rome, Italy, October 19–22, 2020, Proceedings, Part I*, page 441–460, Berlin, Heidelberg, 2020. Springer-Verlag.

[AMAR17] Carlos Aguilar-Melchor, Martin R. Albrecht, and Thomas Ricosset. Sampling from arbitrary centered discrete gaussians for lattice-based cryptography. In Dieter Gollmann, Atsuko Miyaji, and Hiroaki Kikuchi, editors, *Applied Cryptography and Network Security*, pages 3–19, Cham, 2017. Springer International Publishing.

[BBD+15] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, and Pierre-Yves Strub. Verified proofs of higher-order masking. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015*, pages 457–485, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[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.

[BBE+18] Gilles Barthe, Sonia Belaïd, Thomas Espitau, Pierre-Alain Fouque, Benjamin Grégoire, Mélissa Rossi, and Mehdi Tibouchi. Masking the GLP lattice-based signature scheme at any order. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2018*, volume 10821 of *Lecture Notes in Computer Science*, pages 354–384. Springer, 2018.

[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.

[BCG+13] 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 E. Lauter, and Petr Lisonek, editors, *Selected Areas in Cryptography - SAC 2013 - 20th International Conference, Burnaby, BC, Canada, August 14-16, 2013, Revised Selected Papers*, volume 8282 of *Lecture Notes in Computer Science*, pages 402–417. Springer, 2013.

[BCNS15] Joppe Bos, Craig Costello, Michael Naehrig, and Douglas Stebila. Post-quantum key exchange for the tls protocol from the ring learning with errors problem. In S Peisert, editor, *Proceedings of the 2015 IEEE Symposium on Security and Privacy*, pages 553–570. Institute of Electrical and Electronics Engineers Inc., United States of America, 2015.

[BCZ18] Luk Bettale, Jean-Sébastien Coron, and Rina Zeitoun. Improved high-order conversion from boolean to arithmetic masking. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(2):22–45, May 2018.

[BDE+18] Jonathan Bootle, Claire Delaplace, Thomas Espitau, Pierre-Alain Fouque, and Mehdi Tibouchi. LWE without modular reduction and improved side-channel attacks against BLISS. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part I*, volume 11272 of *LNCS*, pages 494–524. Springer, Cham, December 2018.

[BGG+14] Josep Balasch, Benedikt Gierlichs, Vincent Grosso, Oscar Reparaz, and François-Xavier Standaert. On the Cost of Lazy Engineering for Masked Software Implementations. pages 64–81, 11 2014.

[BHLY16] 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.

[BPT+25a] Pierre-Augustin Berthet, Justine Paillet, Cédric Tavernier, Lilian Bossuet, and Brice Colombier. Masked Computation of the Floor Function and Its Application to the FALCON Signature. *IACR Communications in Cryptology*, 1(4):1–23, January 2025.

[BPT+25b] Pierre-Augustin Berthet, Justine Paillet, Cédric Tavernier, Lilian Bossuet, and Brice Colombier. Improving the masked division for the FALCON signature. Cryptology ePrint Archive, Paper 2025/628, 2025.

[BV21] David Blackman and Sebastiano Vigna. Scrambled linear pseudorandom number generators. *ACM Trans. Math. Softw.*, 47(4), September 2021.

[CC24] Keng-Yu Chen and Jiun-Peng Chen. Masking floating-point number multiplication and addition of falcon: First- and higher-order implementations and evaluations. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2024(2):276–303, Mar. 2024.

[CGMZ22] Jean-Sébastien Coron, François Gérard, Simon Montoya, and Rina Zeitoun. High-order table-based conversion algorithms and masking lattice-based encryption. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(2):1–40, 2022.

[CGTV15] Jean-Sébastien Coron, Johann Großschädl, Mehdi Tibouchi, and Praveen Kumar Vadnala. Conversion from arithmetic to boolean masking with logarithmic complexity. In Gregor Leander, editor, *Fast Software Encryption*, volume 9054 of *Lecture Notes in Computer Science*, pages 130–149. Springer, 2015.

[CGTZ23] Jean-Sébastien Coron, François Gérard, Matthias Trannoy, and Rina Zeitoun. Improved gadgets for the high-order masking of dilithium. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2023(4):110–145, Aug. 2023.

[CGV14] Jean-Sébastien Coron, Johann Großschädl, and Praveen Kumar Vadnala. Secure conversion between Boolean and arithmetic masking of any order. In Lejla Batina and Matthew Robshaw, editors, *CHES 2014*, volume 8731 of *LNCS*, pages 188–205. Springer, Berlin, Heidelberg, September 2014.

[CJRR99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In Michael Wiener, editor, *Advances in Cryptology — CRYPTO' 99*, pages 398–412, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[Cor14] Jean-Sébastien Coron. Higher order masking of look-up tables. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, volume 8441 of *Lecture Notes in Computer Science*, pages 441–458. Springer, 2014.

[DDLL13] Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. Lattice signatures and bimodal gaussians. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 40–56. Springer, 2013.

[DG14] Nagarjun C. Dwarakanath and Steven D. Galbraith. Sampling from discrete gaussians for lattice-based cryptography on a constrained device. *Appl. Algebra Eng., Commun. Comput.*, 25(3):159–180, June 2014.

[DKL+18] Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. Crystals-dilithium: A lattice-based digital signature scheme. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(1):238–268, 2018.

[DKM+06] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. Our data, ourselves: Privacy via distributed noise generation. In Serge Vaudenay, editor, *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings*, volume 4004 of *Lecture Notes in Computer Science*, pages 486–503. Springer, 2006.

[DLP14] Léo Ducas, Vadim Lyubashevsky, and Thomas Prest. Efficient identity-based encryption over NTRU lattices. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part II*, volume 8874 of *LNCS*, pages 22–41. Springer, Berlin, Heidelberg, December 2014.

[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, Berlin, Heidelberg, December 2012.

[DZD+18] A. Adam Ding, Liwei Zhang, Francois Durvaux, Francois-Xavier Standaert, and Yunsi Fei. Towards sound and optimal leakage detection procedure. In Thomas Eisenbarth and Yannick Teglia, editors, *Smart Card Research and Advanced Applications*, pages 105–122. Springer International Publishing, 2018.

[EEN+24] Muhammed F. Esgin, Thomas Espitau, Guilhem Niot, Thomas Prest, Amin Sakzad, and Ron Steinfeld. Plover: Masking-friendly hash-and-sign lattice signatures. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology – EUROCRYPT 2024*, pages 316–345, Cham, 2024. Springer Nature Switzerland.

[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 *Advances in Cryptology*

– *EUROCRYPT 2022: 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 – June 3, 2022, Proceedings, Part III*, page 222–253, Berlin, Heidelberg, 2022. Springer-Verlag.

[EFGT17] Thomas Espitau, Pierre-Alain Fouque, Benoît Gérard, and Mehdi Tibouchi. Side-channel attacks on BLISS lattice-based signatures: Exploiting branch tracing against strongSwan and electromagnetic emanations in microcontrollers. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1857–1874. ACM Press, October / November 2017.

[ESLR23] Muhammed F. Esgin, Ron Steinfeld, Dongxi Liu, and Sushmita Ruj. Efficient hybrid exact/relaxed lattice proofs and applications to rounding and VRFs. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part V*, volume 14085 of *LNCS*, pages 484–517. Springer, Cham, August 2023.

[ESZ22a] Muhammed F. Esgin, Ron Steinfeld, and Raymond K. Zhao. Efficient verifiable partially-decryptable commitments from lattices and applications. In *Public Key Cryptography (PKC) (1)*, volume 13177 of *Lecture Notes in Computer Science*, pages 317–348. Springer, 2022. (Full version at ia.cr/2022/142).

[ESZ22b] Muhammed F. Esgin, Ron Steinfeld, and Raymond K. Zhao. MatRiCT$^+$: More efficient post-quantum private blockchain payments. In *IEEE Symposium on Security and Privacy (S&P)*, pages 1281–1298. IEEE, 2022. (Full version at ia.cr/2021/545).

[GG25] François Gérard and Morgane Guerreau. High-Order and Cortex-M4 First-Order Implementations of Masked FrodoKEM. *IACR Cryptol. ePrint Arch.*, 2025.

[GGJR+11] Benjamin Jun Gilbert Goodwill, Josh Jaffe, Pankaj Rohatgi, et al. A testing methodology for side-channel resistance validation. In *In NIST non-invasive attack testing workshop*, volume 7, pages 115–136, 2011.

[Gou01] Louis Goubin. A sound method for switching between Boolean and arithmetic masking. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *CHES 2001*, volume 2162 of *LNCS*, pages 3–15. Springer, Berlin, Heidelberg, May 2001.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, STOC '08, page 197–206, New York, NY, USA, 2008. Association for Computing Machinery.

[GR19] François Gérard and Mélissa Rossi. An efficient and provable masked implementation of qtesla. In Sonia Belaïd and Tim Güneysu, editors, *Smart Card Research and Advanced Applications - 18th International Conference, CARDIS 2019, Prague, Czech Republic, November 11-13, 2019, Revised Selected Papers*, volume 11833 of *Lecture Notes in Computer Science*, pages 74–91. Springer, 2019.

[ISW03] Yuval Ishai, Amit Sahai, and David A. Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481. Springer, 2003.

[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.

[KRVV19] Angshuman Karmakar, Sujoy Sinha Roy, Frederik Vercauteren, and Ingrid Verbauwhede. Pushing the speed limit of constant-time discrete gaussian sampling. a case study on the falcon signature scheme. In *Proceedings of the 56th Annual Design Automation Conference 2019*, DAC '19, New York, NY, USA, 2019. Association for Computing Machinery.

[LSS24] Vadim Lyubashevsky, Gregor Seiler, and Patrick Steuer. The lazer library: Lattice-based zero knowledge and succinct proofs for quantum-safe privacy. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, CCS '24, page 3125–3137, New York, NY, USA, 2024. Association for Computing Machinery.

[Lyu12] Vadim Lyubashevsky. Lattice signatures without trapdoors. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, pages 738–755, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[MGTF19] Vincent Migliore, Benoît Gérard, Mehdi Tibouchi, and Pierre-Alain Fouque. Masking Dilithium - efficient implementation and side-channel evaluation. In Robert H. Deng, Valérie Gauthier-Umaña, Martín Ochoa, and Moti Yung, editors, *ACNS 19International Conference on Applied Cryptography and Network Security*, volume 11464 of *LNCS*, pages 344–362. Springer, Cham, June 2019.

[MOP08] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks: Revealing the secrets of smart cards*. Springer Science & Business Media, 2008.

[MP12] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, pages 700–718, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[MW17] Daniele Micciancio and Michael Walter. Gaussian sampling over the integers: Efficient, generic, constant-time. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part II*, volume 10402 of *Lecture Notes in Computer Science*, pages 455–485. Springer, 2017.

[MWG+22] Soundes Marzougui, Nils Wisiol, Patrick Gersch, Juliane Kramer, and Jean-Pierre Seifert. Machine-learning side-channel attacks on the galactics constant-time implementation of bliss. In *Proceedings of the 17th International Conference on Availability, Reliability and Security*, ARES '22, New York, NY, USA, 2022. Association for Computing Machinery.

[OC14] Colin O'Flynn and Zhizhang (David) Chen. ChipWhisperer: An Open-Source Platform for Hardware Embedded Security Research", booktitle="Constructive Side-Channel Analysis and Secure Design. pages 243–260, Cham, 2014. Springer International Publishing.

[Pei10] Chris Peikert. An efficient and parallel gaussian sampler for lattices. In Tal Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, pages 80–97, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[Pre15] Thomas Prest. *Gaussian sampling in lattice-based cryptography.* Theses, Ecole normale supérieure - ENS PARIS, December 2015.

[Pre23] Thomas Prest. A key-recovery attack against mitaka in the t-probing model. In *Public-Key Cryptography – PKC 2023: 26th IACR International Conference on Practice and Theory of Public-Key Cryptography, Atlanta, GA, USA, May 7–10, 2023, Proceedings, Part I*, page 205–220, Berlin, Heidelberg, 2023. Springer-Verlag.

[RP10] Matthieu Rivain and Emmanuel Prouff. Provably secure higher-order masking of AES. In Stefan Mangard and François-Xavier Standaert, editors, *CHES 2010*, volume 6225 of *LNCS*, pages 413–427. Springer, Berlin, Heidelberg, August 2010.

[SD25] Vladimir Sarde and Nicolas Debande. Improvement of side-channel attacks on mitaka. Cryptology ePrint Archive, Paper 2025/873, 2025.

[SPOG19] Tobias Schneider, Clara Paglialonga, Tobias Oder, and Tim Güneysu. Efficiently masking binomial sampling at arbitrary orders for lattice-based crypto. In *Public-Key Cryptography – PKC 2019: 22nd IACR International Conference on Practice and Theory of Public-Key Cryptography, Beijing, China, April 14-17, 2019, Proceedings, Part II*, page 534–564, Berlin, Heidelberg, 2019. Springer-Verlag.

[TW20] Mehdi Tibouchi and Alexandre Wallet. One bit is all it takes: A devastating timing attack on bliss's non-constant time sign flips. *Journal of Mathematical Cryptology*, 15:131–142, 11 2020.

[WYF+23] Chengkun Wei, Ruijing Yu, Yuan Fan, Wenzhi Chen, and Tianhao Wang. Securely sampling discrete gaussian noise for multi-party differential privacy. In Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda, editors, *ACM CCS 2023*, pages 2262–2276. ACM Press, November 2023.

[ZM24] Jannik Zeitschner and Amir Moradi. PoMMES: Prevention of Micro-architectural Leakages in Masked Embedded Software. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 3:342–376, 2024.

[ZSS20a] Raymond K. Zhao, Ron Steinfeld, and Amin Sakzad. COSAC: COmpact and Scalable Arbitrary-Centered discrete gaussian sampling over integers. In Jintai Ding and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography*, Lecture Notes in Computer Science, pages 284–303. Springer, 2020.

[ZSS20b] Raymond K. Zhao, Ron Steinfeld, and Amin Sakzad. FACCT: FAst, Compact, and Constant-Time discrete gaussian sampler over integers. *IEEE Transactions on Computers*, 69(1):126–137, January 2020.

A Supplementary Algorithms

Algorithm A.1: SamplingCircuit-$\mathcal{B}(p)$ [DKM+06, WYF+23]

Input: Parameter $p = 0.b_0 \dots b_{\mu-1}$ (μ-bit)
Output: Bernoulli sample bit x

$x \leftarrow 1$
for $i \leftarrow \mu - 1$ **to** 0 **do**
    $r \leftarrow \texttt{URBIT}()$
    $t \leftarrow \texttt{XOR}(b_i, r)$
    $x \leftarrow \texttt{MUX}(t,\ \texttt{NOT}(r),\ x)$
return x

Algorithm A.2: SamplingCircuit-$\mathcal{B}(p)$ (with our modifications)

Input: Parameter $p = 0.b_0 \dots b_{\mu-1}$ (μ-bit)
Output: Bernoulli sample bit x

$x \leftarrow 1$
for $i \leftarrow \mu - 1$ **to** 0 **do**
    $r \leftarrow \texttt{URBIT}()$
    $t_1 \leftarrow \texttt{XOR}(b_i,\ r)$
    $t_2 \leftarrow \texttt{XOR}(x,\ \texttt{NOT}(r))$
    $t_3 \leftarrow \texttt{AND}(t_1,\ t_2)$
    $x \leftarrow \texttt{XOR}(x,\ t_3)$
return x

Algorithm A.3: SamplingCircuit-$\mathcal{G}_N(p)$ [WYF+23]

Input: Parameter p, range $N = 2^\kappa$
Output: Geometric sample x
Precompute: $p_i = p^{2^i}/(1 + p^{2^i})$ for $i = 0, \dots, \kappa - 1$

for $i \leftarrow 0$ **to** $\kappa - 1$ **do**
    $b_i \leftarrow$ SamplingCircuit-$\mathcal{B}(p_i)$
$x \leftarrow 0$
for $i \leftarrow 0$ **to** $\kappa - 1$ **do**
    $x \leftarrow x + 2^i b_i$
return x

Algorithm A.4: SamplingCircuit-$\mathcal{G}_N(p)$ (with our modifications)

Input: Parameter p, range $N = 2^\kappa$
Output: Geometric sample x
Precompute: $p_i = p^{2^i}/(1 + p^{2^i})$ for $i = 0, \dots, \kappa - 1$

$x \leftarrow 0$
for $i \leftarrow 0$ **to** $\kappa - 1$ **do**
    $b_i \leftarrow$ SamplingCircuit-$\mathcal{B}(p_i)$
    $t \leftarrow b_i \ll i$
    $x \leftarrow \texttt{XOR}(x,\ t)$
return x

Algorithm A.5: SamplingCircuit-$\mathcal{L}_{\mathbb{Z},N}(t)$ [WYF+23]

Input: Laplace param. t, range $N = 2^{\kappa+1}$
Output: Sample v from $\mathcal{L}_{\mathbb{Z},N}(t)$
Precompute: $p = e^{-1/t}, p_0 = f_{\mathcal{L}_{\mathbb{Z},\mathbb{N}}}(0; t)$

$b \leftarrow$ SamplingCircuit-$\mathcal{B}(p_0)$
$x \leftarrow$ SamplingCircuit-$\mathcal{G}_{N-1}(p)$
$s \leftarrow \texttt{URBIT}()$
$v \leftarrow \texttt{MUX}(b, 0, \texttt{ADD}(x, 1))$
$v \leftarrow \texttt{MUX}(s, v, \texttt{NEG}(v))$
return v

Algorithm A.6: SamplingCircuit-$\mathcal{L}_{\mathbb{Z},N}(t)$ (with our modifications)

Input: Laplace param. t, range $N = 2^{\kappa+1}$
Output: Sample v from $\mathcal{L}_{\mathbb{Z},N}(t)$
Precompute: $p = e^{-1/t}, p_0 = f_{\mathcal{L}_{\mathbb{Z},\mathbb{N}}}(0; t)$

$b \leftarrow$ SamplingCircuit-$\mathcal{B}(p_0)$
$x \leftarrow$ SamplingCircuit-$\mathcal{G}_{N-1}(p)$
$t_1 \leftarrow \texttt{NEG}(\texttt{NOT}(b))$
$t_2 \leftarrow \texttt{ADD}(x, 1)$
$v \leftarrow \texttt{AND}(t_1, t_2)$
return v

Algorithm A.7: SamplingCircuit-$\mathcal{B}_{\exp}(u, r)$ [WYF+23]

Input: Integer $u = \sum_{i=0}^{l-1} u_i 2^i$ with $u_i \in \{0, 1\}, r > 0$
Output: Sample b from $\mathcal{B}(\exp(-u/r))$
Precompute: $p_i = \exp(-2^i/r)$ for $i = 0, \ldots, l-1$

$b \leftarrow 1$
for $i \leftarrow 0$ **to** $l - 1$ **do**
  $b_i' \leftarrow$ SamplingCircuit-$\mathcal{B}(p_i)$
  $v \leftarrow \texttt{OR}(\texttt{NOT}(u_i),\ b_i')$
  $b \leftarrow \texttt{AND}(b, v)$
return b

Algorithm A.8: SamplingCircuit-$\mathcal{N}_{\mathbb{Z}}(\sigma, n)$ [WYF+23]

Input: Parameter σ, sample number n
Output: m samples $s_0, \ldots, s_{m-1}$ and acceptance bits $b_0, \ldots, b_{m-1}$
Precompute:
$t = \sigma^2/\lfloor\sigma\rceil, \quad r = 2\sigma^2 \qquad$ if $\sigma \geq 1$
$t = \sigma^2/\lceil 1/\sigma\rceil, r = 2\sigma^2\lceil 1/\sigma\rceil^2 \quad$ if $0 < \sigma < 1$

for $i \leftarrow 0$ **to** $m - 1$ **do**
  $s_i \leftarrow$ SamplingCircuit-$\mathcal{L}_{\mathbb{Z},N}(t)$
  if $\sigma \geq 1$ **then**
    $v_i \leftarrow \texttt{SUB}(\texttt{ABS}(s_i), \lfloor\sigma\rceil)$
  else
    $v_i \leftarrow \texttt{SUB}(\texttt{MUL}(\lceil 1/\sigma\rceil, \texttt{ABS}(s_i)), 1)$
  $u_i \leftarrow \texttt{SQU}(v_i)$
  $b_i \leftarrow$ SamplingCircuit-$\mathcal{B}_{\exp}(u_i, r)$
return $s = (s_0, \ldots, s_{m-1})$,
  $b = (b_0, \ldots, b_{m-1})$

Algorithm A.9: SamplingCircuit-$\mathcal{N}_{\mathbb{Z}}(\sigma, n)$ (with our modifications)

Input: Parameter $\sigma \geq 1$, sample number n
Output: n accepted samples
Precompute: $t = \sigma^2/\lfloor\sigma\rceil,\ r = 2\sigma^2$

Initialize $\boldsymbol{s}$ as an n-length array of zeros
$j \leftarrow 0$
for $i \leftarrow 0$ **to** $m - 1$ **do**
  $s \leftarrow$ SamplingCircuit-$\mathcal{L}_{\mathbb{Z},N}(t)$
  $v \leftarrow \texttt{ADD}(s, -\lfloor\sigma\rceil)$
  $u \leftarrow \texttt{SQU}(v)$
  $b \leftarrow$ SamplingCircuit-$\mathcal{B}_{\exp}(u, r)$
  sign $\leftarrow \texttt{URBIT}()$
  signmask $\leftarrow \texttt{NEG}(\text{sign})$
  $s \leftarrow \texttt{XOR}(s, \text{signmask})$
  $\hat{s} \leftarrow \texttt{ADD}(s, 1)$
  diffmask $\leftarrow \texttt{XOR}(\hat{s}, s)$
  selmask $\leftarrow \texttt{AND}(\text{diffmask}, \text{signmask})$
  $t' \leftarrow \texttt{XOR}(s, \text{selmask})$
  $b' \leftarrow \texttt{NEG}(b)$
  $t' \leftarrow \texttt{AND}(t', b')$
  $\boldsymbol{s}_j \leftarrow \texttt{XOR}(\boldsymbol{s}_j, t')$
  $j \leftarrow \texttt{ADD}(j, b)$
return $\boldsymbol{s} = (s_0, \ldots, s_{n-1})$