Efficient Soft Analytical Side-Channel Attacks on Large-Scale Cryptographic Computations

Yiteng Sun[1], Zhuo Huang[2], Yan Zhuang[3], Shuo Sun[3], Xinyu Li[1], Yu Yu[2] and Weijia Wang[1](✉)

[1] Shandong University, School of Cyber Science and Technology, Qingdao, China
sunyiteng@mail.sdu.edu.cn,xinyuli2024@mail.sdu.edu.cn,wjwang@sdu.edu.cn
[2] Shanghai Jiao Tong University, Shanghai 200240, China
sh1kaku@sjtu.edu.cn,yyuu@sjtu.edu.cn
[3] China Mobile Internet zhuangyan@cmic.chinamobile.com,sunshuoshine@139.com

Abstract. Soft Analytical Side-Channel Attacks (SASCA) combine leakage-derived priors from multiple intermediate variables with their functional dependencies through belief propagation (BP). However, when applying SASCA to large-scale cryptographic computations where algorithms are abstracted into extensive factor graphs with large candidate sets per variable node, the memory and computational complexity of SASCA become prohibitive. A natural first choice for large-domain variables is to fragment them into smaller-domain variables when the underlying computation decomposes accordingly. For modular addition and multiplication, however, preserving cross-fragment dependencies can introduce short cycles and coupled factor updates, motivating alternative inference strategies. We consider the Number Theoretic Transform (NTT) in ML-DSA as a representative large-scale cryptographic computation, where standard SASCA (with FFT optimization) requires approximately 122 GB of memory for message propagation in an unprotected single-trace setting, even for a 6-layer sub-NTT component, while masking further amplifies the graph size and inference cost.

To address this limitation, we propose Greedy Region-Wise Pruning SASCA (GRWP-SASCA), a practical and efficient framework that enables scalable inference by dividing the global factor graph into manageable regions. The core idea is to replace global BP with a sequence of localized inference steps, where regions are incrementally merged, and the search space is reduced via greedy pruning of redundant structures and low-confidence candidates, thereby significantly reducing the memory and computational complexity of the BP algorithm. This design provides a flexible attack strategy for large-domain arithmetic factor graphs. Our results show that GRWP-SASCA transforms previously infeasible SASCA attacks on large-scale cryptographic computations into practical ones. Theoretical analysis of single-trace attacks on unprotected ML-DSA shows that GRWP-SASCA reduces the memory overhead associated with message propagation by a factor of up to 151 compared to the standard SASCA, while achieving an estimated speedup by a factor of 68. For d-order masked ML-DSA with multiple (say, t) traces, GRWP-SASCA achieves an estimated reduction in memory overhead associated with message propagation by a factor of up to $388.13t(d+1.14)/(d+2.44)$, while achieving a speedup by a factor of approximately $2d+4$. Real-device experiments on an ARM Cortex-M4 platform demonstrate that the secret can be recovered from an unprotected implementation within approximately 15 minutes with a single trace, with a peak message memory usage of approximately 0.9 GB. For first-order masked ML-DSA, the secret can be recovered within 3.2 hours using 8 traces, with a peak message memory usage of approximately 8.1 GB.

Keywords: SASCA · Belief propagation · NTT · ML-DSA · Masked implemen-

tations

1 Introduction

In cryptanalysis, side-channel attacks (SCAs), first introduced in [Koc96, KJJ99], are a class of physical attacks that exploit information leakage such as power consumption, electromagnetic radiation, or timing data to recover secret information. By circumventing the mathematical security of cryptographic algorithms, SCAs pose a severe threat to physical implementations, particularly in the context of embedded devices. In scenarios where multiple intermediate variables of a cryptographic algorithm exhibit side-channel leakage, SASCA [VGS14] models the leakage points as variable nodes and their functional dependencies as factor nodes within a factor graph, performing joint inference via the Belief Propagation (BP) algorithm [Pea82, Mac03] to recover secret information.

In this work, we define *large-scale cryptographic computations* as those in which cryptographic algorithms can be abstracted into extensive factor graphs with large candidate sets per variable node. Such structures commonly arise in modern cryptographic algorithms, which are often designed with increasing structural complexity and enlarged candidate spaces to enhance resistance against various forms of cryptanalysis. However, large-scale cryptographic computations pose significant challenges for SASCA inference. In BP, each factor-to-variable update aggregates information over all candidate assignments of the other variables adjacent to the same factor node. Hence, the update cost grows rapidly with both the factor-node degree, i.e., the number of adjacent variable nodes, and the sizes of their candidate sets. Meanwhile, every BP message is a distribution over a variable candidate set, so large candidate domains also lead to substantial memory overhead. These two effects make conventional full-graph SASCA prohibitive for large-scale cryptographic computations. A natural first choice for reducing large candidate domains is to fragment variables into smaller bit- or byte-level domains. While this is effective for Boolean or bitwise computations, faithful fragmentation of word-level arithmetic introduces cross-fragment dependencies and many short cycles, making it less suitable for large-domain arithmetic factor graphs.

The Number Theoretic Transform (NTT) with large modulus q serves as a representative example of such large-scale cryptographic computations. For instance, prior works [PPM17, PP19] have successfully applied SASCA to attack the NTT operation, enabling the recovery of private keys in schemes such as LPR [LPR10] and Kyber (now standardized as ML-KEM) [BDK+18, Nat24a]. In the typical case where modulus q ranges from 2^{13} to 2^{14}, the BP algorithm remains computationally feasible. However, when the modulus q is large, such as $q \approx 2^{23}$ in ML-DSA [DKL+18, Nat24b] or qTESLA [ABB+20], challenges arise. Taking ML-DSA with $q = 8\,380\,417$ as an example, we summarize these challenges as follows:

1. **Memory overhead:** The attack frameworks in [PPM17, PP19] require constructing the complete NTT factor graph and propagating probability messages over its variable domains. In typical high-performance NTT implementations, lazy reduction allows intermediate coefficients to temporarily exceed the canonical range modulo q. Consequently, a coefficient in the jth NTT layer may have a candidate domain whose size is on the order of jq. Using a conservative `float32` estimate for BP message tables, detailed later in Section 5, the message storage alone for an 8-layer single-polynomial ML-DSA-44 NTT is estimated to reach approximately 1.11 TB. This estimate excludes the storage of candidate values, graph metadata, and temporary workspace, and therefore already indicates that conventional full-graph SASCA is impractical.

2. **Computational overhead:** The time complexity of the BP algorithm scales significantly with q. For the NTT factor graph, the dominant computational cost originates from addition and subtraction operations, which can reach $\mathcal{O}(q^2)$ in the worst case. While the FFT-based optimization proposed in [PPM17] can reduce the complexity to $\mathcal{O}(q \log q)$, a large factor graph combined with a large q still renders the overall computation impractical.

3. **Compounding challenges under masked implementations:** When masking is applied for side-channel protection, each variable and operation is split into multiple shares. This multiplication of the graph's components directly amplifies the memory and computational challenges outlined above, making SASCA attacks significantly more challenging.

In realistic attack scenarios, the enormous candidate space makes applying standard SASCA to large-scale cryptographic computations memory-intensive and computationally prohibitive. This fundamental limitation motivates the development of a more efficient and practical attack framework proposed in this work.

State of the Art. Except for prior SASCA attacks on lattice-based NTTs [PPM17, PP19], the closest lines of work targeting large-scale cryptographic computations are SASCA attacks on 32-bit implementations and their graph-modeling optimizations [BS21, KPP20, YK21, YKSH23]. Bronchain et al. [BS21] target bitsliced masked implementations of AES and Clyde, where the factor graph is built from Boolean additions and multiplications operating on small parallel slices. Kannwischer et al. [KPP20] target Keccak, whose permutation is dominated by XORs, rotations, and small Boolean constraints; their model uses bit-level or clustered variables together with optimized message updates, including convolution-based handling of Hamming-weight leakage. Another related line of work uses fragment templates for 32-bit software implementations. You et al. [YK21] target Keccak by splitting word-level leakage into byte fragments before using the resulting fragment likelihoods in a bitwise BP/SASCA network. You et al. [YKSH23] extend this fragment-template strategy to ASCON, using byte or 16-bit fragments and combining the resulting likelihoods with BP and key enumeration. Table 1 summarizes the corresponding modeling units.

Table 1: Graph-modeling units used by related SASCA-style attacks.

Work	**Target**	**Variable-node unit**
Green et al. [GRO18]	AES [Nat01]	Byte variables
Bronchain et al. [BS21]	AES	Bitsliced Boolean variables
Kannwischer et al. [KPP20]	Keccak [BDPA11]	Bit / clustered variables
You and Kuhn [YK21]	Keccak	Byte fragments
You et al. [YKSH23]	ASCON [DEMS21]	Byte / 16-bit fragments
Primas et al.; Pessl et al. [PPM17, PP19]	ML-KEM [Nat24a]	NTT coefficients

The purpose of Table 1 is to position the graph-modeling choices of prior work, rather than to define direct experimental baselines for all of them. The 32-bit implementations considered in these works are also large-scale cryptographic computations, but their dominant constraints are native Boolean operations over bits, slices, or small clusters. For bitsliced AES, Keccak, and ASCON, operations such as XOR, rotation, and Boolean multiplication can therefore be represented at the bit or fragment level without introducing the carry, borrow, and partial-product state required by word-level arithmetic. Their modeling granularity is thus aligned with the structure of the target computation. A natural first choice for variables with large candidate domains is to fragment them into smaller-domain variables, and this remains preferable when the underlying computation

decomposes naturally at the corresponding granularity, as in Boolean or bitwise operations. For word-level arithmetic, however, faithful fragmentation must preserve cross-fragment dependencies. Addition and subtraction introduce carry and borrow chains, together with auxiliary variables and short cycles that auxiliary variables and short cycles that can make loopy BP less stable. Multiplication and modular reduction are even more strongly coupled: partial products and reduction terms jointly depend on multiple fragments, so exact factor updates still require evaluating coupled assignments across the fragmented word. Consequently, reducing the domain of each individual variable does not necessarily yield a comparable reduction in factor-update complexity. These cases, where fragmentation does not effectively resolve the inference bottleneck, motivate our region-wise inference and pruning strategy for large-domain arithmetic factor graphs.

More specifically, splitting a word-level arithmetic variable into k-bit chunks does not generally yield independent small factors. Omitting the resulting cross-fragment dependencies gives a less faithful representation of the original computation, whereas preserving them requires coupled factor updates across multiple fragments. Thus, although fragmentation reduces the domain size of individual variable nodes, the corresponding factor-update complexity, especially for multiplication and modular reduction, may remain substantial.

Fragment-template attacks [YK21, YKSH23] address a related but distinct issue at the leakage-modeling level. They reduce the cost of profiling and inference by modeling only a selected k-bit fragment at a time, while the leakage contributions of the remaining bits or fragments are absorbed as nuisance variation or switching noise. When several parts of a word jointly contribute to the measured leakage, this additional variation makes the selected fragment harder to distinguish and can result in less concentrated or less accurate leakage likelihoods. This issue is particularly relevant to large-domain arithmetic intermediates, especially in multiplication and modular reduction, where partial products and reduction terms jointly depend on multiple fragments. Thus, fragment-wise leakage modeling may both discard useful joint leakage information and provide weaker priors for subsequent BP.

Thus, fragmentation-based approaches, including decomposing large-domain variable nodes into smaller-domain variables and constructing fragment-level leakage models, remain the preferred first option when they are compatible with the target computation and leakage structure. Our approach is instead intended for large-domain arithmetic settings where such fragmentation does not effectively remove the BP bottleneck.

Green et al. [GRO18] study a related but different question: how static AES factor-graph choices affect BP-based key recovery. Their experiments show that including later-round components does not necessarily improve first-round key recovery. This setting remains byte-oriented, uses small variable domains, and selects the graph before BP is executed; it therefore does not address the large-domain arithmetic bottleneck considered here.

1.1 Our Contributions

In this work, we address the prohibitive memory and computational cost of SASCA when applied to large-scale cryptographic computations.

Greedy Region-Wise Pruning SASCA We observe that factor graphs arising from large-scale cryptographic computations can be naturally divided into region-wise subgraphs. Based on this insight, we propose Greedy Region-Wise Pruning SASCA (GRWP-SASCA), a flexible attack framework that transforms prohibitively expensive global inference into a sequence of localized and tractable inference steps. GRWP-SASCA incrementally expands the inference frontier by merging successive regions, while controlling the search space through a dual-level pruning mechanism that removes redundant structures and

low-confidence candidates. Unlike prior static approaches, our framework enables attacker-configurable inference, allowing the integration order and pruning strategy to be adapted to the algorithmic execution flow and leakage characteristics. By dynamically managing the search space in a region-wise manner, GRWP-SASCA significantly reduces both memory and computational overhead, thereby making SASCA practical for analyzing large-scale cryptographic implementations that were previously computationally infeasible.

Evaluation of GRWP-SASCA on NTT Structures We evaluate GRWP-SASCA on the NTT in ML-DSA, a representative large-scale cryptographic computation; while the proposed methodology applies to all NIST security parameter sets, we focus on ML-DSA-44 as a primary case study. We use the FFT-optimized SASCA methods of [PPM17, PP19] as the standard baseline. Based on theoretical analysis, the standard SASCA on an unprotected implementation under a single-trace setting requires about 122 GB of memory for message propagation and approximately 4 hours of runtime to attack a 6-layer sub-NTT component. [1] Under the same experimental setup and attack configuration, our framework transforms such infeasible attacks into practical ones, reducing the message memory overhead by a factor of up to 151 and achieving an estimated computational speedup by a factor of approximately 68 over standard SASCA. Real-device experiments on ARM Cortex-M4 further validate our approach: the secret can be recovered within 15 minutes from a single trace, with a peak message memory usage of about 0.9 GB.

Evaluation of GRWP-SASCA on Masked Implementations We also evaluate GRWP-SASCA on the NTT of masked implementations. Compared to the standard SASCA, our approach makes such attacks computationally feasible. We evaluate both the standard baseline and GRWP-SASCA under an identical computational environment with multiple (say, t) traces. For d-order masked implementations, GRWP-SASCA reduces the message memory overhead by a factor of up to $388.13t(d{+}1.14)/(d{+}2.44)$ compared to the standard SASCA. Furthermore, GRWP-SASCA achieves an estimated speedup scaling by a factor of approximately $2d + 4$. These evaluations are conducted on the attack of a 6-layer sub-NTT component, consistent with the unprotected setting. Real-device experiments on ARM Cortex-M4 demonstrate that 8 traces are required to recover the secret from the first-order masked implementation within 3.2 hours using GPU acceleration, with a peak message memory usage of about 8.1 GB.

2 Background

2.1 Notation

For any integer q, the set of integers modulo q with centered reduction is denoted by $\mathbb{Z}_q$. We denote by $\mathbb{Z}_q[X]$ the ring of polynomials with coefficients in $\mathbb{Z}_q$. Let $\mathcal{R} = \mathbb{Z}[X]/(X^n{+}1)$ be the quotient ring of polynomials modulo $X^n + 1$, and $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$ denote the same ring with coefficients reduced modulo q. Variables in $\mathcal{R}_q$ are represented by roman lowercase letters, while vectors with polynomial elements in $\mathcal{R}$ or $\mathcal{R}_q$ are represented by bold lowercase letters, and matrices are represented by bold uppercase letters. In the following expressions, all polynomial operations are performed in $\mathcal{R}_q$.

For $\mathbf{w} \in \mathcal{R}_q^k$, where each polynomial in $\mathbf{w}$ has n coefficients, we denote by $\mathbf{w}[i, j]$ the jth coefficient of the ith polynomial. By default, we use the L_∞-norm, which is the maximum absolute value among the elements in a vector. For a vector $\mathbf{w}$, the L_∞ norm is expressed

[1] By combining lattice reduction techniques, we can attack multiple 6-layer sub-NTT components. This approach is sufficient to recover the full ML-DSA secret key without requiring the recovery of all coefficients.

as $||\mathbf{w}||_\infty = \max |\mathbf{w}[i,j]|$. Let $\mathcal{S}_\eta$ denote the set of polynomials p such that $\|p\|_\infty \leq \eta$. Let $\mathcal{B}_\eta$ denote the set of polynomials whose coefficients follow a centered binomial distribution with mean 0 and satisfy $\|p\|_\infty \leq \eta$.

2.2 Number Theoretic Transform

The Number Theoretic Transform (NTT) is the finite-field analogue of the Discrete Fourier Transform and is widely used in lattice-based cryptography to accelerate polynomial multiplication. In schemes based on Ring-LWE and Module-LWE [LPR10, LS15], the NTT transforms a polynomial $a(X) = \sum_{i=0}^{n-1} a_i X^i$ into its evaluation-domain representation over $\mathbb{Z}_q$. The core operation of the NTT is the butterfly unit (BFU). In an abstract modular description, given two inputs $a, b \in \mathbb{Z}_q$ and a twiddle factor ζ, the BFU computes

$$a' = a + \zeta b \pmod{q}, \qquad b' = a - \zeta b \pmod{q}.$$

In optimized lattice-based implementations, including ML-DSA, the BFU is typically implemented with lazy reduction. Instead of reducing the two outputs after each addition and subtraction, the implementation lets intermediate coefficients propagate as bounded integer representatives:

$$t \leftarrow \mathsf{Mont}(\zeta b), \qquad a' \leftarrow a + t, \qquad b' \leftarrow a - t.$$

The additions and subtractions are therefore not immediately followed by modular reductions. As the NTT layers progress, these intermediate values may temporarily exceed the canonical range of $\mathbb{Z}_q$; for example, values in the jth layer may lie in an implementation-dependent range whose size grows roughly proportionally to jq. A final or subsequent reduction step brings the coefficients back to the required representation when the NTT output is consumed by later operations.

2.3 ML-DSA

ML-DSA is the module-lattice-based digital signature scheme standardized from CRYSTALS-Dilithium [DKL+18, Nat24b]. Its security is based on the hardness of Module-LWE and Module-SIS problems [LS15]. In this work, we only rely on the parameters that are relevant to the NTT computation and to the secret distributions targeted by our side-channel analysis. These parameters are summarized in Table 2.

Table 2: ML-DSA parameters relevant to this work

NIST Level	q	n	k	ℓ	η
ML-DSA-44	8 380 417	256	4	4	2
ML-DSA-65	8 380 417	256	6	5	4
ML-DSA-87	8 380 417	256	8	7	2

From an implementation and side-channel perspective, ML-DSA exhibits two features that are central to this work. First, the secret vectors $\mathbf{s}_1$ and $\mathbf{s}_2$ consist of ℓ and k polynomials, respectively, each with $n = 256$ coefficients sampled from a small-norm distribution bounded by η. This small secret domain provides a strong structural prior for SASCA. Second, ML-DSA relies extensively on the NTT over the large modulus $q = 8\,380\,417$ to accelerate polynomial multiplications. In the implementation targeted in this work, the secret polynomials are transformed from the coefficient domain into the NTT domain as

$$\hat{\mathbf{s}}_1 = \mathsf{NTT}(\mathbf{s}_1) \quad \text{and} \quad \hat{\mathbf{s}}_2 = \mathsf{NTT}(\mathbf{s}_2).$$

Algorithm 1 Targeted Operations in ML-DSA Signing

Input: Secret vectors $\mathbf{s}_1, \mathbf{s}_2$, message M. **Output:** Signature $\sigma = (\mathbf{z}, c)$.

$\hat{\mathbf{s}}_1 \leftarrow \mathsf{NTT}(\mathbf{s}_1)$, $\hat{\mathbf{s}}_2 \leftarrow \mathsf{NTT}(\mathbf{s}_2)$ ▷ **Targeted NTT operations**
repeat
  Sample $\mathbf{y}$ and compute $\mathbf{w} \leftarrow \mathbf{Ay}$
  $\mathbf{w}_1 \leftarrow \mathsf{HighBits}_q(\mathbf{w})$
  $c \leftarrow \mathsf{H}(\mu \| \mathsf{w1Encode}(\mathbf{w}_1))$
  $\hat{c} \leftarrow \mathsf{NTT}(c)$
  $\mathbf{z} \leftarrow \mathbf{y} + c\mathbf{s}_1$, where $c\mathbf{s}_1 = \mathsf{NTT}^{-1}(\hat{c} \circ \hat{\mathbf{s}}_1)$
  $\mathbf{r} \leftarrow \mathbf{w} - c\mathbf{s}_2$, where $c\mathbf{s}_2 = \mathsf{NTT}^{-1}(\hat{c} \circ \hat{\mathbf{s}}_2)$
until $\mathbf{z}$ and $\mathbf{r}$ pass the bound checks
return $\sigma = (\mathbf{z}, c)$

Accordingly, this work targets the side-channel leakage arising from these secret-dependent forward NTT computations. The resulting NTT-domain representations are subsequently used in secret-dependent products such as

$$c\mathbf{s}_1 = \mathsf{NTT}^{-1}\big(\hat{c} \circ \hat{\mathbf{s}}_1\big) \quad \text{and} \quad c\mathbf{s}_2 = \mathsf{NTT}^{-1}\big(\hat{c} \circ \hat{\mathbf{s}}_2\big).$$

2.4 Stochastic Model-Based Multivariate Gaussian Templates

We adopt a stochastic-model-based multivariate Gaussian template construction [SLP05, CK14, CK18, CDSU23, YKSH23]. The leakage is modeled as a parametric function of the bits of the target variable. For each time sample i, the leakage is written as

$$\mathcal{L}^{(i)} = \mathbf{C}^{(i)}\psi(x) + \epsilon_i,$$

where $\psi(x)$ is a vector of basis functions derived from the binary representation of x, $\mathbf{C}^{(i)}$ is the coefficient vector for the ith time sample, and ϵ_i denotes Gaussian noise.

Let $\mathrm{bin}(x) = (b_0, b_1, \ldots, b_{w-1})$ be the w-bit representation of x. In our implementation, we use a second-order basis

$$\psi(x) = \big[1, \underbrace{b_0, b_1, \ldots, b_{w-1}}_{\text{first-order terms}}, \underbrace{b_0b_1, b_0b_2, \ldots, b_{w-2}b_{w-1}}_{\text{second-order interactions}}\big]^{\mathsf{T}}.$$

The first-order terms model the individual contribution of each bit, while the second-order terms capture pairwise bit interactions that are not represented by a simple Hamming-weight model. For a w-bit variable, this basis has dimension $m = 1 + w + \binom{w}{2}$. We restrict the basis to second-order terms because our goal is to obtain a sufficiently accurate leakage model without making the profiling step itself prohibitively expensive. Higher-order bases can be used in principle, but their dimension grows quickly (e.g., adding $\binom{w}{3}$ for third-order terms), which increases both profiling complexity and likelihood-evaluation costs.

Stacking all N selected time samples together, the full leakage trace is modeled as

$$\mathbf{L} = \mathbf{C}\psi(x) + \boldsymbol{\epsilon},$$

where $\mathbf{L} \in \mathbb{R}^N$ is the leakage trace, $\mathbf{C} \in \mathbb{R}^{N \times m}$ is the coefficient matrix, and $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$ is the noise vector. The coefficient matrix $\mathbf{C}$ is estimated from profiling traces and then used to dynamically derive the mean vector of any candidate value as $\boldsymbol{\mu}_x = \mathbf{C}\psi(x)$.

Following the common pooled-covariance assumption, the noise covariance matrix $\boldsymbol{\Sigma}$ is shared across all candidate values and estimated from the residuals $\mathbf{L} - \mathbf{C}\psi(x)$ over the profiling set. Consequently, the likelihood of observing a trace $\mathbf{L}$ given a specific candidate x is evaluated as:

$$p(\mathbf{L} \mid x) = \frac{1}{(2\pi)^{N/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{L} - \boldsymbol{\mu}_x)^{\mathsf{T}}\boldsymbol{\Sigma}^{-1}(\mathbf{L} - \boldsymbol{\mu}_x)\right).$$

2.5 Soft Analytical Side-Channel Attacks

Charvillon et al. proposed Soft Analytical Side-Channel Attacks (SASCA) in [VGS14]. SASCA models side-channel information and the structure of cryptographic algorithms as a Bayesian network, efficiently propagating probabilistic information via the BP algorithm [Pea82, Mac03]. SASCA models multiple sensitive variables and constructs a factor graph by capturing dependencies among multiple intermediate variables. The BP algorithm achieves convergence by propagating probability distributions that integrate information from the relevant leakage points.

2.5.1 Belief Propagation

We model the target algorithm using a bipartite factor graph $\mathcal{G} = (\mathcal{V}, \mathcal{F}, \mathcal{E})$. Here, $\mathcal{V}$ is the set of **variable nodes** (representing intermediate variables with candidate sets $\mathcal{C}_v$), and $\mathcal{F}$ is the set of **factor nodes** (encoding cryptographic operations or constraints). Edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{F}$ connect variables to their associated factors. Let $\Gamma(\cdot)$ denote the neighborhood of a node. Each factor $f \in \mathcal{F}$ has an indicator compatibility function:

$$\phi_f\big((x_u)_{u\in\Gamma(f)}\big) = \begin{cases} 1, & \text{if } (x_u)_{u\in\Gamma(f)} \text{ satisfies the operation encoded by } f, \\ 0, & \text{otherwise.} \end{cases}$$

For example, if f encodes $z = x + y$, then $\phi_f(x, y, z) = 1$ for valid assignments and 0 otherwise.

The BP algorithm iteratively updates marginal probability distributions (messages) along the edges of $\mathcal{G}$ to infer the most likely values of $\mathcal{V}$. The process proceeds as follows:

1. **Initialization**: Each variable $v \in \mathcal{V}$ is assigned a prior distribution $\Pr_I(v)$. If side-channel leakage $\mathcal{L}_v$ is observed, $\Pr_I(v) = \Pr(v \mid \mathcal{L}_v)$ (derived from templates or classifiers); otherwise, it is initialized with a uniform or Gaussian prior.

2. **Message Propagation**: Directed messages $m_{v\to f}$ and $m_{f\to v}$ (distributions over $\mathcal{C}_v$) are exchanged iteratively until convergence (difference $< \epsilon$ or iterations $> T_{\max}$). At iteration $t+1$, the messages evaluated at candidate $x \in \mathcal{C}_v$ are updated as:

   - **Variable-to-factor:**
   $$m_{v\to f}^{(t+1)}(x) = \alpha_{v\to f} \Pr_I(v = x) \prod_{h\in\Gamma(v)\setminus\{f\}} m_{h\to v}^{(t)}(x)$$

   - **Factor-to-variable:**
   $$m_{f\to v}^{(t+1)}(x) = \alpha_{f\to v} \sum_{\mathbf{x}_{-v}\in\prod_{u\in\Gamma(f)\setminus\{v\}}\mathcal{C}_u} \phi_f(x, \mathbf{x}_{-v}) \prod_{u\in\Gamma(f)\setminus\{v\}} m_{u\to f}^{(t)}(x_u).$$

   where $\alpha_{v\to f}$ and $\alpha_{f\to v}$ are normalization constants, and $\mathbf{x}_{-v}$ denotes assignments to $\Gamma(f) \setminus \{v\}$.

3. **Marginalization**: After convergence, the final marginal belief for candidate x is computed as:
   $$\mathcal{B}[v, x] = \alpha_v \Pr_I(v = x) \prod_{f\in\Gamma(v)} m_{f\to v}(x)$$
   where α_v is a normalization constant.

The complete Loopy BP procedure, incorporating the prior set $\mathcal{P}$, is detailed in Algorithm 2.

Algorithm 2 Standard BP Subroutine

Input:
$\mathcal{G} = (\mathcal{V}, \mathcal{F}, \mathcal{E})$: Factor graph.
$\mathcal{P}$: Prior distributions, where $\mathcal{P}[v, x] = \Pr_I(v = x)$ for $x \in \mathcal{C}_v$.
$T_{\max}$: Maximum number of BP iterations.

Output: $\mathcal{B}[v] \mid v \in \mathcal{V}$: Marginal distributions, where $\mathcal{B}[v, x] \stackrel{\text{def}}{=} \Pr(v = x \mid \text{evidence})$.
Procedure: $\mathrm{BP}(\mathcal{G}, \mathcal{P}, T_{\max})$

Initialize all messages $m^{(0)}_{v\to f}$ and $m^{(0)}_{f\to v}$ to uniform distributions.
$t \leftarrow 0$
repeat
  for each factor node $f \in \mathcal{F}$ **do** ▷ Factor-to-variable message update.
    for each variable node $v \in \Gamma(f)$ **do**
      for each $x \in \mathcal{C}_v$ **do**
        $m^{(t+1)}_{f\to v}(x) \leftarrow \alpha_{f\to v} \sum_{\mathbf{x}_{-v}} \phi_f\left(x, \mathbf{x}_{-v}\right) \prod_{u\in\Gamma(f)\setminus\{v\}} m^{(t)}_{u\to f}(x_u)$
  for each variable node $v \in \mathcal{V}$ **do** ▷ Variable-to-factor message update.
    for each factor node $f \in \Gamma(v)$ **do**
      for each $x \in \mathcal{C}_v$ **do**
        $m^{(t+1)}_{v\to f}(x) \leftarrow \alpha_{v\to f} \Pr_I(v = x) \prod_{h\in\Gamma(v)\setminus\{f\}} m^{(t)}_{h\to v}(x)$
  $t \leftarrow t + 1$
until converged **or** $t \geq T_{\max}$
for each variable node $v \in \mathcal{V}$ **do** ▷ Compute marginal beliefs.
  for each $x \in \mathcal{C}_v$ **do**
    $\mathcal{B}[v, x] \leftarrow \alpha_v \Pr_I(v = x) \prod_{f\in\Gamma(v)} m_{f\to v}(x)$
Return $\mathcal{B}$

3 Main Idea

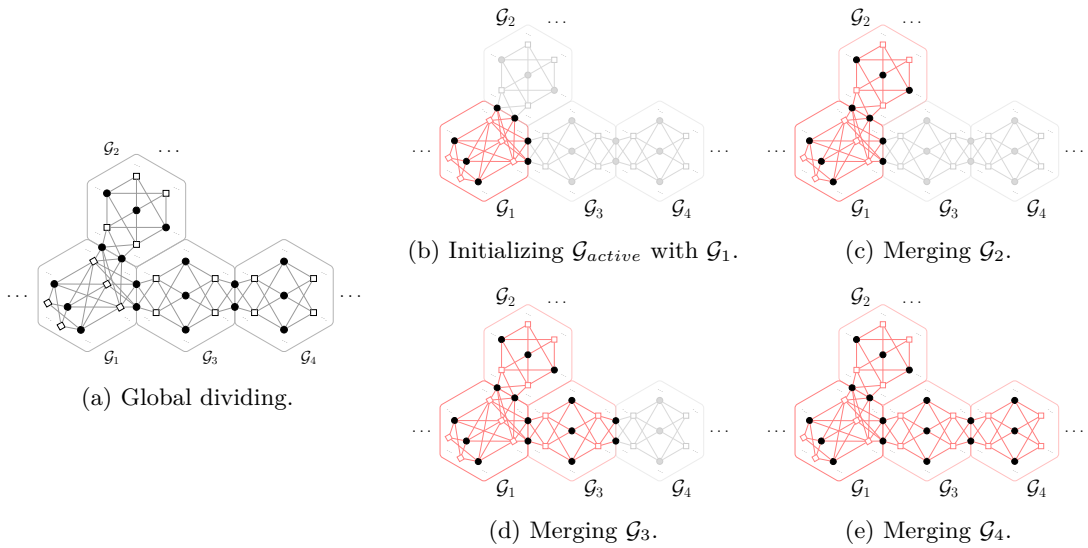
Section Goal. This section presents the general GRWP-SASCA framework. The goal is to show how leakage-derived local priors can be propagated efficiently without running BP on the full factor graph. We introduce the region-wise expansion, localized BP, and structural/value pruning primitives that form the core of GRWP-SASCA.

3.1 Greedy Region-Wise Pruning SASCA

Factor graphs arising from large-scale cryptographic computations can often be decomposed into multiple connected regions, as illustrated in Figure 1a. Let $\mathcal{G}_i = (\mathcal{V}_i, \mathcal{F}_i)$ denote the ith region, where $\mathcal{V}_i$ and $\mathcal{F}_i$ are the corresponding variable-node and factor-node sets. Adjacent regions communicate through interface variables. We denote by $\mathcal{V}^{\text{connect}}_{i,j}$ the set of variables shared by regions $\mathcal{G}_i$ and $\mathcal{G}_j$. When two regions are merged, the variables in $\mathcal{V}^{\text{connect}}_{i,j}$ are treated as the same variable nodes, so that information can pass across the regional boundary. Based on this observation, we propose an iterative, region-wise strategy termed *Greedy Region-Wise Pruning SASCA* (GRWP-SASCA), which combines region-wise factor graph expansion with pruning. As shown in Figure 1b-1e, GRWP-SASCA maintains an active graph $\mathcal{G}_{\text{active}}$ and incrementally absorbs new regions. After each expansion, BP is performed only on the current active graph rather than on the complete global graph, and the inferred beliefs are pruned before the next region is merged. The general execution flow for each integrated region is as follows:

- **Graph expansion:** The active factor graph $\mathcal{G}_{\text{active}}$ is expanded by merging the next region $\mathcal{G}_i$ via the interface variable set $\mathcal{V}^{\text{connect}}_{i,j}$. The merged interface variables connect the newly added region with the previously active subgraph.

- **Localized Belief Propagation:** BP is executed on the expanded $\mathcal{G}_{\text{active}}$ until convergence. The evidence used for this localized inference is derived from two sources. First, variables retained from previous regions carry information obtained in earlier inference steps. In the standard region-wise schedule, this information is represented



(a) Global dividing. (b) Initializing $\mathcal{G}_{active}$ with $\mathcal{G}_1$. (c) Merging $\mathcal{G}_2$. (d) Merging $\mathcal{G}_3$. (e) Merging $\mathcal{G}_4$.

Figure 1: The iterative strategy of the GRWP-SASCA framework. Figure 1a Region-wise division of a global factor graph. Figure 1b–1e The iterative merging execution: the active subgraph $\mathcal{G}_{\text{active}}$ incrementally absorbs new regions $\mathcal{G}_i$. Red highlights the current active graph, on which BP is executed after each merge; gray denotes regions pending integration.

by pruned beliefs and reused as soft priors. In masked multi-trace settings, however, trace-specific posteriors are not immediately used as priors for subsequent traces; instead, their trace-specific contributions are cached as soft evidence and combined after all traces for the current region have been processed. Second, variables newly introduced by $\mathcal{G}_i$ receive leakage-derived likelihoods when such leakage is available. Importantly, localized BP is performed on the entire expanded active graph, not only on the newly merged region. Therefore, information flows **bidirectionally** over all active edges: the newly added region can refine the beliefs of variables inherited from previous regions, while the retained variables can also constrain and guide inference in the newly added region. By integrating retained information and new leakage likelihoods through iterative message passing, BP refines the marginal distributions over the active graph and concentrates probability mass on candidates that are consistent with both leakage and functional constraints.

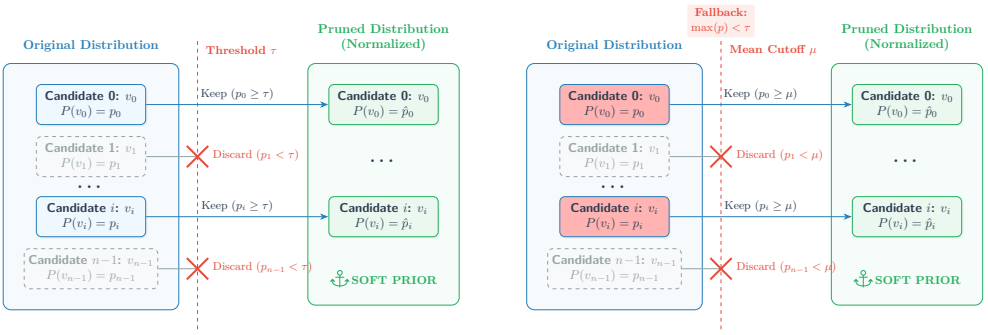
- **Dual-level pruning:** After localized BP, GRWP-SASCA reduces the active inference problem using two pruning primitives: structural pruning and value-level pruning. Structural pruning removes auxiliary graph components that no longer need to be retained, whereas value-level pruning reduces candidate domains by keeping high-probability candidates as refined soft priors. The two operations are conceptually separated because they serve different purposes: structural pruning controls graph size, while value-level pruning controls candidate-domain size.

  The timing of value-level pruning depends on the evidence structure of the target setting. In a single-trace or unprotected region-wise schedule, the evidence relevant to the current region is available after one localized BP run, so value-level pruning can be applied immediately. In a masked multi-trace setting, each trace provides only one conditionally independent evidence contribution for the same virtual secret variables. Therefore, applying value-level pruning after a single trace may prematurely commit to an overconfident but incorrect posterior. For this reason, our masked instantiation applies structural pruning after each trace-wise BP, caches the

trace-specific evidence externally, and delays value-level pruning until the aggregated virtual-layer belief has been obtained.

Structural pruning: Redundant structures can be removed from the topology according to the attacker-defined set $\mathcal{V}_{\text{remove}}$. Such structures may include temporary masking components, trace-specific auxiliary nodes, or intermediate operations that do not need to be retained across regions. These structures are pruned only after BP has propagated their information to the retained part of the graph. In the unprotected setting, this information is carried forward through posterior beliefs on retained variables. In the masked multi-trace setting, the information contributed by each trace is instead cached as trace-specific evidence and combined with the evidence from other traces before value-level pruning.

Value-level pruning: Value-level pruning is applied only after the relevant evidence for the current region has been integrated. Candidates with posterior probabilities above the threshold τ are retained and normalized. If no candidate exceeds τ, GRWP-SASCA falls back to a weaker cutoff, such as the mean of the current probability distribution, to preserve a non-empty candidate set. The retained distribution is then used as a soft prior in subsequent region expansions. In the masked multi-trace instantiation, the posterior used for value-level pruning is the aggregated virtual-layer belief obtained after combining all trace-specific evidence terms. The visualized illustration of the value-level pruning mechanism is presented in Figure 2.



(a) Threshold-based pruning.

(b) Fallback pruning when no candidate exceeds τ.

Figure 2: Value-level pruning. Candidates exceeding the threshold τ are retained and normalized. If no candidate exceeds τ, the mean probability μ is used as a fallback cutoff to avoid an empty candidate set. The resulting distribution is reused as a soft prior in subsequent GRWP-SASCA iterations.

We formalize the standard region-wise execution flow in Algorithm 3. At each iteration, the next region is merged into the active graph, BP is executed on the expanded active graph, and pruning is applied before the next region is processed. The algorithm should be interpreted as a generic scheduling interface for GRWP-SASCA. In the unprotected instantiation, the pruning step directly performs value-level pruning after each localized BP run. In the masked multi-trace instantiation, the same interface is realized with trace-wise structural pruning followed by delayed evidence aggregation and value-level pruning, as described in Section 4.3. The corresponding primitive pruning operations are given in Algorithm 4.

In Algorithm 3, the notation $\hat{\mathcal{B}}_{i-1} \cup \mathcal{L}_i$ denotes the construction of the current prior set: retained variables use the pruned beliefs from the previous iteration as soft priors, while newly introduced variables use leakage-derived likelihoods. Since BP is run on the

Algorithm 3 Greedy Region-Wise Pruning SASCA (GRWP-SASCA)

Input:
$\mathcal{G}$: The complete factor graph representing the cryptographic algorithm.
$\{\mathcal{L}_i\}_{i=1}^{\ell}$: Leakage-derived prior probability distributions for ℓ regions.
$\{\mathcal{G}_i = (\mathcal{V}_i, \mathcal{F}_i)\}_{i=1}^{\ell}$: A division of $\mathcal{G}$ into ℓ regions.
$\{\mathcal{V}_{\text{remove},i}\}_{i=1}^{\ell}$: Sets of auxiliary variable nodes to be structurally removed in each region.
$\{m_i\}_{i=1}^{\ell}$: Pruning modes, where $m_i \in$ structural, value, both.
τ: Pruning threshold.
Output: $\hat{\mathcal{B}}_l$
Procedure: GRWP-SASCA$(\mathcal{G}, \{\mathcal{L}_i\}_{i=1}^{\ell}, \{\mathcal{G}_i = (\mathcal{V}_i, \mathcal{F}_i)\}_{i=1}^{\ell}, \{\mathcal{V}_{\text{remove},i}\}_{i=1}^{\ell}, \{m_i\}_{i=1}^{\ell}, \tau)$

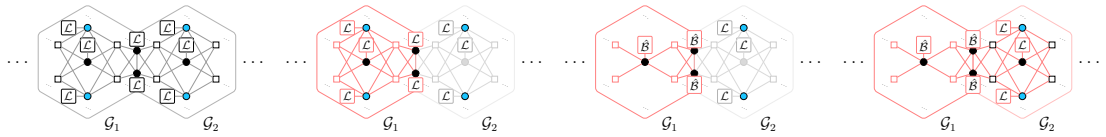
$\mathcal{G}_{\text{active}} \leftarrow \varnothing$, $\hat{\mathcal{B}}_0 \leftarrow \varnothing$
for $i \leftarrow 1$ **to** ℓ **do**
  $\mathcal{G}_{\text{active}} \leftarrow \mathcal{G}_{\text{active}} \cup \mathcal{G}_i$ ▷ Merge next region into active graph.
  $\mathcal{P}_i \leftarrow \hat{\mathcal{B}}_{i-1} \cup \mathcal{L}_i$ ▷ Assign previous pruned beliefs and new leakage likelihoods as priors.
  $\mathcal{B}_i \leftarrow \text{BP}(\mathcal{G}_{\text{active}}, \mathcal{P}_i)$ ▷ Run BP on the entire active graph; messages propagate bidirectionally.
  $\hat{\mathcal{B}}_i, \mathcal{G}_{\text{active}} \leftarrow$PRUNE$(\mathcal{G}_{\text{active}}, \mathcal{B}_i, \tau, \mathcal{V}_{\text{remove},i}, m_i)$ ▷ Dual-level pruning.
Return Final pruned beliefs $\hat{\mathcal{B}}_l$.

Algorithm 4 Pruning Subroutine

Input:
$\mathcal{G}_{\text{active}}$: The current active factor graph.
$\mathcal{B}$: Marginal distributions of variables in $\mathcal{G}$active.
τ: Pruning threshold.
$\mathcal{V}_{\text{remove}}$: Set of variable nodes to be structurally removed.
mode: structural, value, both.
Output:
$\hat{\mathcal{B}}$: Retained beliefs on the remaining active variables.
$\mathcal{G}_{\text{active}}$: Updated graph structure.
Procedure: PRUNE$(\mathcal{G}_{\text{active}}, \mathcal{B}, \tau, \mathcal{V}_{\text{remove}}, mode)$

$\hat{\mathcal{B}} \leftarrow \mathcal{B}[v] \mid v \in \mathcal{G}_{\text{active}}.\mathcal{V}$ ▷ Default: retain current beliefs.
if *mode* = structural **or** *mode* = both **then**
  for each variable node $v \in \mathcal{V}_{\text{remove}}$ **do** ▷ Structural-level pruning.
    $\mathcal{F}_{\text{adj}} \leftarrow f \in \mathcal{G}_{\text{active}}.\mathcal{F} \mid v \in \Gamma(f)$ ▷ Adjacent factor nodes.
    for each factor node $f \in \mathcal{F}$adj **do**
      $\mathcal{G}$active.remove_edge(f, v)
      if f is isolated **then**
        $\mathcal{G}$active.remove_node(f)
    $\mathcal{G}_{\text{active}}$.remove_node$(v)$
    remove $\hat{\mathcal{B}}[v]$
if *mode* = value **or** *mode* = both **then**
  for each variable node $v \in \mathcal{G}_{\text{active}}.\mathcal{V}$ **do** ▷ Value-level pruning and prior re-instantiation.
    $mask_1 \leftarrow [\mathcal{B}[v, x] \geq \tau$ for all $x \in \mathcal{C}_v]$ ▷ Threshold-based filter.
    if True $\in mask_1$ **then**
      $\hat{\mathcal{B}}[v] \leftarrow$ Normalize$(\mathcal{B}[v, mask_1])$ ▷ Store pruned normalized beliefs.
    else
      $mask_2 \leftarrow [\mathcal{B}[v, x] \geq$ mean$(\mathcal{B}[v])$ for all $x \in \mathcal{C}_v]$ ▷ Fallback filter.
      $\hat{\mathcal{B}}[v] \leftarrow$ Normalize$(\mathcal{B}[v, mask_2])$ ▷ Avoid an empty candidate set.
Return $\hat{\mathcal{B}}$, $\mathcal{G}_{\text{active}}$.

entire $\mathcal{G}_{\text{active}}$ after each merge, messages propagate bidirectionally over all active edges. In Algorithm 4, remove_edge(f, v) deletes the edge (f, v) from $\mathcal{G}_{\text{active}}$, while remove_node$(v)$ deletes node v. Any factor node that becomes isolated after edge removal is also deleted. The masks $mask_1$ and $mask_2$ select surviving candidates from the current belief distribution: $mask_1$ keeps candidates above the threshold τ, while $mask_2$ is used as a fallback when no candidate reaches τ, preventing the candidate set from becoming empty. The selected candidates are normalized to form $\hat{\mathcal{B}}$, which is reused as soft prior information in subsequent iterations. The attacker-defined parameter *mode* specifies whether structural pruning, value-level pruning, or both are applied. For masked multi-trace attacks, the generic pruning step is instantiated with trace-wise structural pruning followed by



(a) Input graph. (b) Merge $\mathcal{G}_1$. (c) Run BP and prune. (d) Merge $\mathcal{G}_2$.

Figure 3: Detailed visualization of Algorithm 3. Blue nodes represent the set $\mathcal{V}_{\text{remove}}$ of variable nodes to be removed.

evidence accumulation and delayed value-level pruning, as described in Section 4.3.

Figure 3 provides a visual illustration of the proposed GRWP-SASCA framework. Specifically, Figure 3a shows the overall factor graph used as input to Algorithm 3. Figure 3b-3c illustrates the first execution cycle of GRWP-SASCA. The attacker specifies the auxiliary variables $\mathcal{V}_{\text{remove}}$ for structural pruning. The active graph $\mathcal{G}_{\text{active}}$ then absorbs a new region, runs BP on the entire merged active graph, and applies dual-level pruning. Structural pruning removes the auxiliary variables after their information has been propagated to the retained variables, while value-level pruning keeps surviving candidates as soft priors for later iterations. The same merge–BP–prune cycle is then repeated for the next region. Importantly, after each merge, BP is executed on the whole $\mathcal{G}_{\text{active}}$, so information is exchanged bidirectionally between previously retained variables and the newly added region.

Remark 1 (Heuristic rationale of dual-level pruning). GRWP-SASCA should be viewed as a heuristic approximation to full-graph SASCA rather than an exact inference procedure. Its effectiveness relies on the observation that, after leakage likelihoods and local functional constraints are combined by BP, the posterior distributions of many variables often become concentrated on a relatively small subset of candidates. Keeping the full candidate domain and all auxiliary structures in subsequent regions is therefore unnecessary in many practical cases and leads to prohibitive memory and computational costs. The rationale behind dual-level pruning can be understood from the following two aspects:

Posterior concentration and value redundancy: In side-channel attacks, the observed leakage naturally induces non-uniform prior distributions, where candidates more consistent with the leakage model receive higher probabilities. BP further combines these leakage-derived priors with the functional constraints in the local factor graph, reinforcing candidates that are jointly consistent with leakage and computation while suppressing unlikely ones. As a result, the posterior probability mass is often concentrated on a small subset of candidates. Value-level pruning exploits this concentration by discarding candidates with negligible posterior probabilities and re-normalizing the surviving candidates as soft priors for subsequent regions. This step reduces the candidate-set size while preserving the information that BP has accumulated in the current active graph.

Local information flow and structural redundancy: The factor graph has a local message-passing structure: auxiliary nodes primarily contribute information through their neighboring factors and variables. Some temporary structures, such as masking components or intermediate arithmetic nodes, are needed to propagate local constraints within the current active graph, but they do not necessarily need to be retained in later regions. Once BP has propagated their contribution to the retained interface or secret-relevant variables, these auxiliary structures may become redundant for subsequent inference. Structural pruning removes such nodes after their information has been absorbed into the posterior beliefs of retained variables. This shifts the computational focus from large temporary substructures to the reduced set of variables that carry information forward across regions.

Remark 2 (Scheduling choices and pruning trade-offs). The sequential integration order $\{\mathcal{G}_1, \ldots, \mathcal{G}_l\}$ presented in Algorithm 3 should be interpreted as a computational schedule

optimized for information propagation, rather than a rigid constraint on the factor graph's topology. Consequently, the specific dividing strategy and pruning sequence are primarily dictated by the inherent structure of the target cryptographic algorithm. An adversary must flexibly align these regions with the algorithmic execution flow and the spatial availability of leakage points to maximize belief convergence. Furthermore, the selection of the pruning threshold τ serves as a critical heuristic parameter. It requires careful tuning to navigate the trade-off between computational efficiency and attack robustness, ensuring that ground-truth candidates are retained within the reduced domain while redundant states are excised.

4 Applying GRWP-SASCA to the NTT

Section Goal. This section instantiates the general GRWP-SASCA framework on the ML-DSA NTT, which serves as a representative example of large-scale arithmetic computations. The goal is to show how the abstract notions of regions, interface variables, localized BP, and pruning translate into an implementation-level BFU factor graph.

The NTT is a suitable target for GRWP-SASCA because it is built from repeated BFUs and therefore naturally admits a region-wise decomposition. This structure is particularly favorable for ML-DSA, where secret-related polynomials are sampled from small-norm distributions, so the input coefficients in the first NTT layer have small candidate domains. GRWP-SASCA exploits these low-entropy initial variables to obtain informative beliefs and then propagates the resulting information across subsequent BFU regions. We first identify the leakage-bearing variables inside a BFU, then define a layer-ordered region-wise schedule for unprotected NTT computations, and finally extend the construction to masked multi-trace attacks using a virtual unmasked layer and delayed evidence aggregation.

4.1 Leakage Point

We instantiate GRWP-SASCA on the implementation-level NTT factor graph. Let s denote a target polynomial whose coefficients are processed by the NTT. We use $s[i]$ and $\hat{s}[i]$ to denote the input and output coefficients of the NTT, respectively, and use $s_j[i]$ to denote an intermediate coefficient at the jth NTT layer.

For a single BFU, we denote by u the upper coefficient, by ℓ the lower coefficient, by t the Montgomery product, and by u_{out} and ℓ_{out} the two output coefficients. In the ML-DSA implementation, the BFU follows the lazy-reduction form described in Section 2.2:

$$t \leftarrow \mathsf{Mont}_\zeta(u), \qquad u\text{out} \leftarrow \ell - t, \qquad \ell_{\text{out}} \leftarrow \ell + t,$$

where ζ is the precomputed NTT twiddle factor used by the current BFU, and $\mathsf{Mont}_\zeta(u)$ denotes the Montgomery reduction.

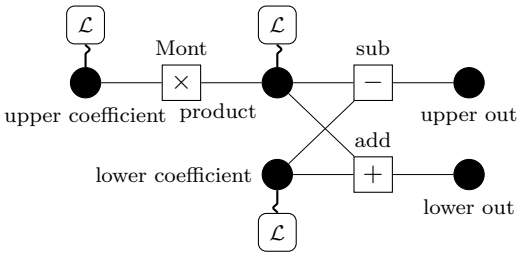


Figure 4: The factor graph of a single BFU.

Figure 4 illustrates a single BFU within the factor graph of the BP algorithm based on the above definition. The black circles denote variable nodes, the square nodes denote deterministic factor nodes, and the $\mathcal{L}$ nodes denote leakage-derived observation factors. The deterministic factors instantiate the generic compatibility function ϕ_f introduced in Section 2.5.1. Specifically, the Montgomery, subtraction, and addition factors are defined as:

$$\phi_{\mathrm{Mont}}(u,t) = \begin{cases} 1, & t = \mathsf{Mont}_\zeta(u), \\ 0, & \text{otherwise}, \end{cases}$$

and

$$\phi_{\mathrm{sub}}(\ell,t,u_{\mathrm{out}}) = \begin{cases} 1, & u_{\mathrm{out}} = \ell - t, \\ 0, & \text{otherwise}, \end{cases} \qquad \phi_{\mathrm{add}}(\ell,t,\ell_{\mathrm{out}}) = \begin{cases} 1, & \ell_{\mathrm{out}} = \ell + t, \\ 0, & \text{otherwise}. \end{cases}$$

For a variable node x, the corresponding observation factor is initialized from the leakage likelihood $\Pr(\mathcal{L}_x \mid x)$ and provides the local prior used by BP.

As indicated in Section 5.4 of [PPM17], Montgomery reduction is the dominant leakage source within the BFU. It provides informative likelihoods for the upper coefficient and the product. Although the leakage of additions and subtractions is weaker, it can still provide useful coarse information, such as sign or range information, for the lower coefficient and output variables. Therefore, the complete BFU, rather than the Montgomery operation alone, is used as the basic region in the NTT factor graph.

We do not apply the bijective-node simplification discussed in [GGSB20] to merge the Montgomery input and output nodes. Over the canonical modular domain $\mathbb{Z}_q$, multiplication by a nonzero twiddle factor is bijective. However, our factor graph follows the implementation-level lazy-reduction behavior of the NTT rather than an abstract canonical modular representation. Intermediate variables are represented by bounded integer candidate domains that may contain multiple representatives of the same residue modulo q. Therefore, the map implemented by the Montgomery node is not necessarily injective over the candidate domains used in our factor graph. Merging the upper coefficient and the product would collapse distinct lazy representatives that must remain distinguishable for the neighboring addition and subtraction constraints. We therefore keep the Montgomery operation as an explicit deterministic factor, while BP can still propagate information in both directions through this factor.

4.2 Applying GRWP-SASCA to NTT

We instantiate GRWP-SASCA on the NTT factor graph using BFU-level regions. In this instantiation, a region is not an entire NTT layer, but the factor subgraph corresponding to one BFU. The NTT layer index only provides a natural ordering and grouping of these BFU regions. Therefore, the resulting schedule is a layer-ordered region-wise schedule: BFU regions are still merged and pruned as regions, while regions belonging to the same NTT layer can be processed in a structured and potentially parallel manner.

Let $L = \log_2 n$ denote the number of NTT layers. For layer j, define the BFU index set

$$\mathcal{I}_j \stackrel{\mathrm{def}}{=} \left\{ (i,k) \;\middle|\; i \in [n - \frac{n}{2^j}, n), k \in [0, 2^{j-1} - 1] \right\}.$$

For each $(i,k) \in \mathcal{I}j$, we denote the corresponding BFU region by

$$\mathcal{G}_{j,i,k} \stackrel{\mathrm{def}}{=} (\mathcal{V}_{j,i,k}, \mathcal{F}_{j,i,k}).$$

Each region contains the upper coefficient, the Montgomery product, the lower coefficient, the corresponding output variables, and the deterministic BFU factors described in Section 4.1. The Montgomery product is therefore modeled locally inside each BFU region,

but it is not needed as a separate layer-level dependency set. For notation, we group the layer-j BFU inputs into upper coefficients $\mathbf{U}_j$ and lower coefficients $\mathbf{L}_j$:

$$\mathbf{U}_j \stackrel{\text{def}}{=} \{s_j[\pi_u(i,j,k)] \mid (i,k) \in \mathcal{I}_j\},$$
$$\mathbf{L}_j \stackrel{\text{def}}{=} \{s_j[\pi_l(i,j,k)] \mid (i,k) \in \mathcal{I}_j\},$$

where $\pi_u(i,j,k) \stackrel{\text{def}}{=} i - \frac{kn}{2^{j-1}}$ and $\pi_l(i,j,k) \stackrel{\text{def}}{=} i - \frac{(2k+1)n}{2^j}$ represent the index mapping functions for the BFU at layer j.

The layer-ordered schedule is motivated by the dependency structure induced by the upper-side BFU inputs. Constraining the upper coefficients in layer j induces backward constraints on a subset of the original input coefficients. We denote the set of input ancestors involved in the upper-side variables of layer j by

$$\mathbf{S}_j \stackrel{\text{def}}{=} \{s[\pi_u(i,j,k)] \mid (i,k) \in \mathcal{I}_j\}.$$

The cumulative set of input ancestors covered up to layer j is $\mathbf{N}_j \stackrel{\text{def}}{=} \bigcup_{r=1}^{j} \mathbf{S}_r$. Under the standard NTT ordering considered here, this set expands until $\mathbf{N}_L = s[1], \ldots, s[n-1]$, where $s[0]$ is excluded because it typically does not pass through a twiddle-factor multiplication. Its cardinality is $|\mathbf{N}_j| = 2^{L-j}(2^j - 1)$. Equivalently, the newly exposed input ancestors at layer j are contained in $\mathbf{N}_j \setminus \mathbf{N}_{j-1}$, whose size is 2^{L-j}.

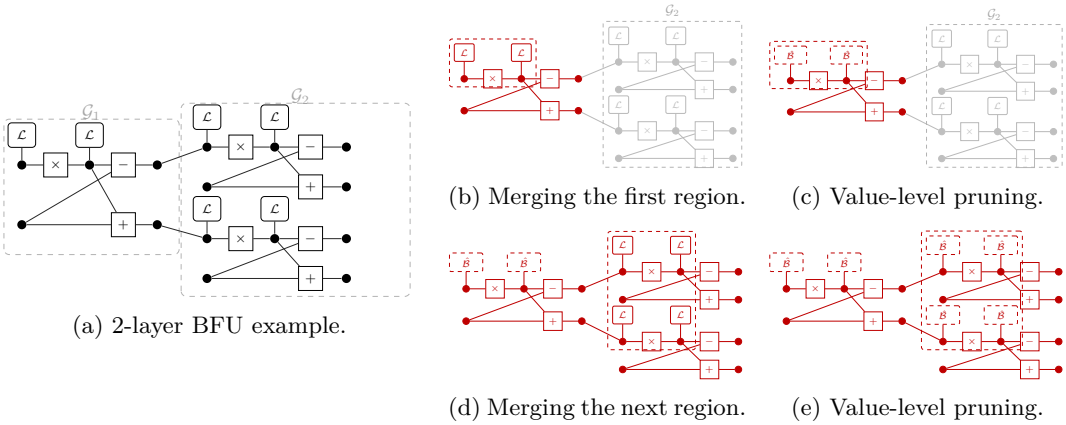
This dependency pattern is favorable to GRWP-SASCA because the original input coefficients come from the small secret domain $[-\eta, \eta]$. In the first layer, the relevant input variables therefore have only $2\eta+1$ candidates. After BP and pruning, the retained posterior beliefs of these variables are reused as soft priors. When BFU regions from deeper layers are merged, these soft priors constrain the newly introduced variables through the deterministic BFU relations. Thus, the attack avoids explicitly materializing the full candidate space induced by all possible combinations of input coefficients.

In contrast, a monolithic SASCA on the full NTT graph may face a rapid growth of feasible candidates. In a naive representation, a layer-j intermediate coefficient may depend on up to 2^{j-1} original small-norm coefficients. Combined with the expanded integer ranges caused by lazy reduction, its candidate domain can grow on the order of $\min((2\eta+1)^{2^{j-1}}, jq)$. GRWP-SASCA mitigates this growth by applying BP and pruning after each layer-ordered expansion of BFU regions. We refer to this effect as low-entropy inheritance: concentrated beliefs obtained from earlier BFU regions are carried forward as soft priors, reducing the effective uncertainty seen by later regions.

Operationally, the attack processes the NTT dependency cone using a region-wise schedule. At each step, the BFU regions required at the current layer are merged into the active graph $\mathcal{G}_{\text{active}}$. BP is executed on the expanded active graph, and the resulting beliefs are pruned and retained as soft prior information for subsequent regions. For $j > 1$, structural pruning further restricts the candidate domains of newly merged BFU variables using the retained candidate sets from previous layers and the deterministic BFU constraints.

When the BFU regions up to layer j have been processed, the active graph decomposes into 2^{L-j} independent sub-NTT components. These components can be processed independently, enabling parallel BP execution. Each component introduces one new input ancestor relative to the previously processed layers, so up to 2^{L-j} input coefficients can be newly constrained in parallel at layer j. As the schedule proceeds to deeper layers, these independent components are gradually merged into larger dependency structures, eventually covering the target set $\mathbf{N}_L$.

For intuition, Figure 5 illustrates GRWP-SASCA on a two-layer NTT dependency cone with one first-layer BFU and two second-layer BFUs. The first-layer BFU forms the initial region $\mathcal{G}_1$, while the two second-layer BFUs form the pending region $\mathcal{G}_2$. As shown



(a) 2-layer BFU example.

(b) Merging the first region.

(c) Value-level pruning.

(d) Merging the next region.

(e) Value-level pruning.

Figure 5: An illustrative example of GRWP-SASCA on a two-layer NTT dependency cone. The graph contains one first-layer BFU and two second-layer BFUs. Red nodes and edges denote the current active graph, while gray nodes and edges denote regions not yet merged. The red dashed box marks the leakage-bearing nodes introduced in the current iteration, rather than the whole BFU structure. The observation factors $\mathcal{L}$ represent leakage-derived priors, while $\hat{\mathcal{B}}$ denotes refined soft priors retained after value-level pruning.

in Figure 5b, $\mathcal{G}_1$ is first merged into the active graph and BP is executed using the leakage-derived priors $\mathcal{L}$. The red dashed box highlights only the leakage-bearing Montgomery path introduced in the current iteration, rather than the whole BFU structure. After value-level pruning, the retained posteriors are stored as refined soft priors $\hat{\mathcal{B}}$ (Figure 5c). The next iteration merges $\mathcal{G}_2$ and runs BP on the entire expanded active graph (Figure 5d), allowing information to propagate bidirectionally between the retained beliefs from $\mathcal{G}_1$ and the newly introduced leakage and constraints in $\mathcal{G}_2$. The resulting beliefs are then pruned again, as shown in Figure 5e.

4.3 Attacking Masked Implementations

We now extend GRWP-SASCA to masked implementations. In masked implementations of ML-DSA [ABC+22, ABC+23, CGTZ23, BBE+24], the secret polynomial s is protected via arithmetic masking. That is, s is split into $d+1$ shares $\{s_0, \ldots, s_d\}$ with $s_i \in \mathcal{R}_q$ for d-order masked implementations, such that $\sum_{i=0}^{d} s_i \equiv s \bmod q$.

Three-Dimensional Factor Graph As a showcase example, we take a first-order masking scenario ($d = 1$), where the secret s is split into two shares satisfying $s_0 + s_1 \equiv s \pmod q$. Due to the linearity of the NTT, this relationship is preserved in the transform domain: $\mathrm{NTT}(s_0) + \mathrm{NTT}(s_1) \equiv \mathrm{NTT}(s) \bmod q$.

As illustrated in Figure 6, we therefore construct a three-dimensional factor graph (3D-FG). The share planes model the NTT computations of the randomized shares, while the virtual unmasked plane contains the secret variables targeted by the attack. Corresponding share variables and virtual secret variables are connected by modular addition factors, enforcing the recombination constraint. This construction allows leakage from independently randomized shares to be fused into beliefs on the same virtual secret variables. The same construction extends naturally to higher-order masking by adding more share planes and one recombination constraint per virtual variable.

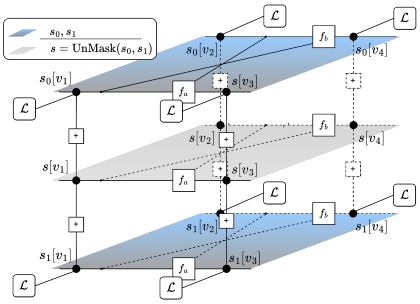


Figure 6: A showcase example of a three-dimensional factor graph for first-order masking.

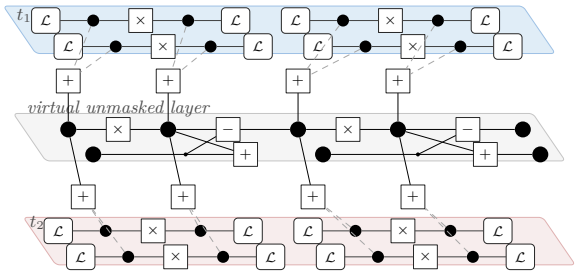


Figure 7: 3D-FG of a first-order masked 2-layer BFU structure using $T = 2$ power traces.

Multi-Trace Attack In masked implementations, the shares are refreshed for each execution, whereas the underlying secret remains fixed. Therefore, information from multiple traces can be accumulated through common secret variables. This follows the multi-trace SASCA principle introduced by Veyrat-Charvillon et al. [VGS14], where trace-specific factor graphs are replicated and connected through common key variables. Green et al. [GRO18] further study different graphical-model strategies for combining multiple traces, including combinations through common secret nodes and Bayesian combinations of posterior distributions.

GRWP-SASCA adopts a sequential and memory-efficient variant of this idea. For each trace, the corresponding trace-specific share subgraph is merged into the active graph and connected to the virtual unmasked layer through recombination constraints. BP is then executed on the expanded graph, and the resulting belief on the virtual variables is interpreted as trace-specific soft evidence. After BP, the trace-specific share subgraph is structurally pruned, but value-level pruning is not applied immediately.

After all traces for the current BFU region have been processed, their trace-specific evidence is combined on the virtual variables. Value-level pruning is then applied only to the aggregated virtual-layer belief. This avoids committing to an overconfident posterior from a single trace before the remaining traces have contributed their evidence. The pruned aggregated belief is reused as the soft prior for subsequent BFU regions.

Equivalently, assuming that the traces are conditionally independent given a secret variable x, the accumulated belief follows the usual Bayesian combination

$$\Pr(x \mid \mathcal{L}^{(1)}, \ldots, \mathcal{L}^{(T)}) \propto \Pr(x) \prod_{t=1}^{T} \Pr(\mathcal{L}^{(t)} \mid x).$$

In our sequential implementation, each trace contribution is refined by BP on its trace-specific factor graph before being combined on the virtual variables.

4.3.1 GRWP-SASCA on Masked Implementations

GRWP-SASCA remains applicable to masked implementations because the 3D-FG preserves the BFU-level region structure of the unmasked NTT, while adding recombination constraints between the share planes and the virtual unmasked layer. For a selected BFU region, the active graph contains the virtual unmasked variables, the trace-specific share subgraphs, and the modular recombination factors that connect corresponding share variables to the virtual secret variables.

The masked attack follows a trace-wise merge–BP–structural-prune schedule followed by delayed value-level pruning. Let $\mathcal{B}_{\text{base}}$ denote the belief obtained from the clean virtual graph before injecting any trace-specific share region. For each trace r, GRWP-SASCA

builds the same clean local graph, temporarily merges the corresponding trace-specific share region, connects it to the virtual layer through recombination constraints, and runs BP. The resulting posterior on the selected virtual variables is denoted by $\mathcal{B}_r$.

Since $\mathcal{B}_r$ contains both the trace-specific contribution and the common clean-graph contribution, we convert it into trace evidence by removing the clean base belief:

$$\Delta_r[v, x] \propto \frac{\mathcal{B}_r[v, x]}{\mathcal{B}_{\text{base}}[v, x]}.$$

In practice, this ratio is computed in the log domain with a small numerical floor. After extracting Δ_r, the trace-specific share structure is removed, and the next trace is processed from the same clean virtual graph rather than from the posterior of the previous trace.

After all traces have been processed, the evidence terms are accumulated as

$$\Delta_{\text{agg}}[v, x] \propto \prod_{r=1}^{T} \Delta_r[v, x].$$

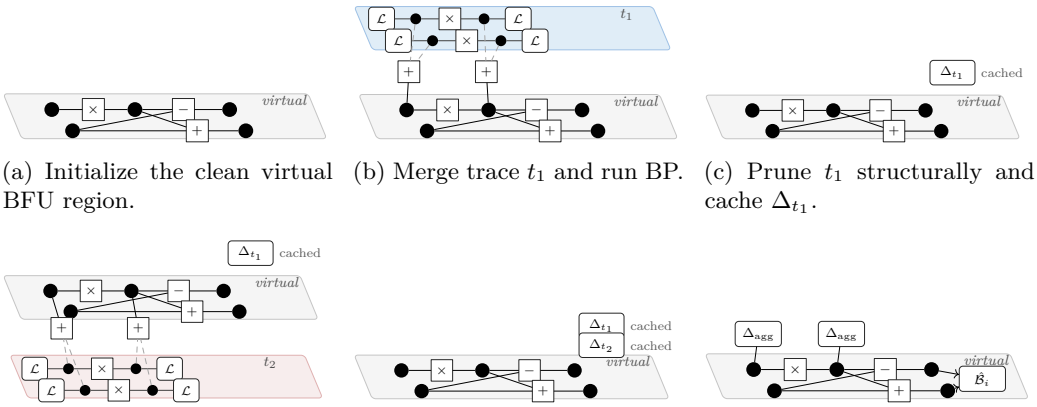
The aggregated trace evidence is then attached to the clean virtual graph as an additional observation factor, and BP is run once more to synchronize the aggregated evidence with the virtual-layer constraints. Value-level pruning is applied only after this final synchronization step. The resulting pruned belief is then used to constrain subsequent BFU regions.

Figure 7 gives a compact example of the masked setting on a first-order two-layer BFU structure with $T = 2$ traces. The blue and red planes correspond to the trace-specific share subgraphs for t_1 and t_2, respectively, while the gray plane represents the virtual unmasked layer that contains the target secret variables. The dashed vertical edges denote modular recombination constraints, which enforce the relation between the randomized shares and the corresponding virtual unmasked variables.

A fully expanded masked factor graph could include all share-side NTT structures for all traces and all BFU regions. Such a graph preserves the complete implementation-level dependency structure, but it is unnecessarily expensive for GRWP-SASCA because most of these trace-specific structures are auxiliary and do not need to be retained across iterations. In our construction, only the trace-specific share subgraphs associated with the current BFU region are temporarily merged. Moreover, since the informative leakage is concentrated on the Montgomery path, the merged share-side structures focus on the leakage-bearing upper and product variables, together with the recombination factors that connect them to the virtual unmasked layer. The remaining share-side BFU structures, which do not provide useful leakage or interface constraints for the current step, are not included in the active region. The temporarily merged share-side variables are then placed in $\mathcal{V}_{\text{remove}}$ and structurally pruned after their information has been propagated to the selected virtual variables.

The resulting masked GRWP-SASCA schedule is summarized in Figure 8. First, the virtual BFU region is initialized in the active graph, and the clean virtual-layer belief $\mathcal{B}_{\text{base}}$ is obtained before injecting any trace-specific share region. Then, for trace t_1, the corresponding share region is temporarily merged and connected to the virtual variables through modular recombination factors; BP is executed on this expanded graph. After BP, the posterior belief induced on the selected virtual variables is converted into trace-specific soft evidence Δ_{t_1} by removing the clean base contribution, and the trace-specific share structures are structurally pruned.

The same procedure is independently repeated for trace t_2 from the same clean virtual graph, rather than from the posterior of t_1. Thus, Δ_{t_1} and Δ_{t_2} are treated as independent trace evidence terms conditioned on the same virtual secret variables. After all trace-specific evidence terms have been collected, they are combined into the aggregated evidence Δ_{agg}. The aggregated evidence is then attached to the retained virtual graph, a



(a) Initialize the clean virtual BFU region.

(b) Merge trace t_1 and run BP.

(c) Prune t_1 structurally and cache Δ_{t_1}.

(d) Process t_2 independently from the clean virtual graph.

(e) Prune t_2 structurally and cache Δ_{t_2}.

(f) Attach Δ_{agg} and apply final BP and value pruning.

Figure 8: Six-step masked GRWP-SASCA workflow for a first-order BFU region with two traces. Trace-specific evidence is extracted independently, accumulated as Δ_{agg}, and value-pruned after a final virtual-layer synchronization step.

final localized BP step propagates it through the virtual-layer constraints, and value-level pruning is applied only after this synchronization step. The resulting pruned belief is used to constrain subsequent BFU regions.

4.4 Lattice Reduction

For the recovery of the secret polynomial, it is unnecessary for SASCA to recover all coefficients. As noted in [PPM17], when a sufficiently large subset of coefficients is known, the complete secret polynomial can be reconstructed via lattice reduction. In our setting, we assume that recovering a sufficiently large subset of coefficients of a polynomial $s \in \mathbf{s}$ suffices to fully recover $\mathbf{s}$ via lattice reduction, following the approaches in [PPM17, HMS+23].

The number of coefficients provided to the lattice-reduction step is determined by the depth of the recovered NTT dependency cone. As defined in Section 4.2, $\mathbf{N}_j$ denotes the cumulative set of input ancestors covered by the upper-side BFU variables up to depth j. For ML-DSA with $n = 256$ and $L = \log_2 n$, its size is

$$|\mathbf{N}_j| = 2^{L-j}(2^j - 1).$$

This relation determines the stopping point of GRWP-SASCA. Since the lattice-reduction post-processing is much cheaper than the side-channel inference phase, it is unnecessary to continue GRWP-SASCA to deeper NTT layers once the number of correctly recovered coefficients exceeds the lattice-reduction threshold.

The lattice-reduction step is constructed from the ML-DSA relation $\mathbf{t} = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2 \pmod q$. Following the lattice construction based on this relation, as considered by Oliveira et al. [OVCG25], we fix the coefficients recovered by GRWP-SASCA and subtract their contribution from the equation. This post-processing step assumes that the coefficients fixed from GRWP-SASCA are correct; incorrect fixed coefficients would change the residual equation and can prevent the lattice solver from succeeding. For all NIST security levels of ML-DSA, our experiments show that recovery succeeds when 243 coefficients per polynomial are known, leaving only 13 unknown coefficients per polynomial to be resolved via the lattice-based solver. The lattice-reduction step takes approximately 10 seconds in our implementation, which is negligible compared with the overall SASCA runtime.

Therefore, lattice reduction mainly determines how many correct coefficients GRWP-SASCA must provide, rather than contributing significantly to the attack runtime. In principle, a 5-layer sub-NTT already provides $|\mathbf{N}_5| = 248$ coefficients per polynomial, exceeding the observed 243-coefficient threshold. In the main evaluation, however, we use a 6-layer sub-NTT, which provides $|\mathbf{N}_6| = 252$ coefficients per polynomial. This gives additional redundancy against occasional low-confidence or unusable SASCA outputs while still avoiding the cost of processing the full NTT.

5 Simulated Evaluation and Comparison

Section Goal. This section evaluates GRWP-SASCA in a controlled simulated setting. The experiments are designed to answer three questions. First, we test whether GRWP-SASCA can recover enough ML-DSA secret coefficients from unprotected and first-order masked NTT computations. Second, we compare GRWP-SASCA with standard SASCA following prior NTT-based SASCA attacks under the same leakage model and CuPy-accelerated environment. Third, we study how pruning parameters and sub-NTT depth affect the robustness and efficiency of the attack.

5.1 Leakage Simulation

We instantiate the stochastic-model-based Gaussian templates described in Section 2.4 in a simulated leakage setting. For efficiency, the simulated leakage uses the first-order stochastic basis, i.e., $\psi_{\text{sim}}(x) = (1, b_0, b_1, \ldots, b_{w-1})^\top$ for the binary representation of x, and omits the pairwise interaction terms used in the general second-order basis. For each target variable x, the simulated leakage vector is generated as

$$\mathbf{L} = \mathbf{C}_{\text{sim}}\psi_{\text{sim}}(x) + \boldsymbol{\epsilon}, \qquad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \sigma^2 I_N),$$

where N denotes the number of selected points of interest and σ controls the noise level. Unless stated otherwise, the entries of $\mathbf{C}_{\text{sim}}$ are independently sampled from $\mathcal{N}(1, 0.05^2)$, modeling small variations in the sensitivity of different first-order basis terms.

For each simulation run, leakage observations are generated for the leakage-bearing upper coefficients and Montgomery product variables in the NTT factor graph. The stochastic Gaussian template then evaluates the likelihood $p(\mathbf{L} \mid x)$ for each candidate value x. The resulting likelihoods are normalized over the candidate set and used as local evidence for BP. We repeat the evaluation under different noise levels to study the robustness and performance of GRWP-SASCA.

5.2 Computational Resources and Implementation Details

All experiments are conducted using identically configured workstations, each equipped with an Intel Core i7-12700 CPU, 32 GB of RAM, and an NVIDIA GeForce RTX 4090 GPU with 24 GB of video memory. The implementation is written in Python using `NumPy` and `CuPy`.

For each BP call, we run 20 loopy iterations and use damped message passing with damping factor $\alpha = 0.9$, following the damping strategy used in [PP19]. In GRWP-SASCA, the pruning threshold is set to $\tau = 0.9$ unless stated otherwise. For dense value-domain SASCA updates, modular addition and subtraction factors are accelerated using FFT-based convolution techniques as in [PPM17]. This acceleration is mainly used for the standard SASCA setting and for masked GRWP-SASCA when the active candidate domains are still dense. In contrast, the unmasked GRWP-SASCA regions are rapidly reduced to small candidate sets after value-level pruning; in this case, direct sparse candidate-domain updates are cheaper than FFT-based updates.

5.3 Evaluation

GRWP-SASCA is applicable to the ML-DSA NTT across different security parameter sets. In our evaluation, we use ML-DSA-44 as a representative example. Following Section 4.4, we target 6-layer sub-NTT components. Across the four independent components of one polynomial, the total coefficient coverage is $|\mathbf{N}_6| = 252$, which is sufficient for the subsequent lattice-reduction step.

Unless stated otherwise, simulated leakage is generated for the leakage-bearing upper coefficients and Montgomery product variables. We use $N = 5$ points of interest for each such variable and derive local likelihoods using the stochastic-model-based Gaussian templates from Section 5.1.

Evaluation of GRWP-SASCA. Figure 9 shows the simulated single-trace result for unprotected ML-DSA using GRWP-SASCA. Figure 10 reports the number of traces required for successful recovery in the first-order masked setting. As expected, masking requires multi-trace evidence accumulation, and the required number of traces increases as the noise level grows. GRWP-SASCA is implemented using our `CuPy`-accelerated GPU framework. For the unprotected setting, recovering one 6-layer sub-NTT component with GRWP-SASCA takes approximately 3.5 minutes. For the first-order masked setting, the runtime is evaluated under a four-device execution schedule, in which independent active subgraphs within the same GRWP expansion layer are processed in parallel, while different expansion layers are processed sequentially. Under this schedule, processing one trace for one polynomial through the first six NTT layers takes approximately 6 minutes on average.

Evaluation against Standard SASCA. We compare GRWP-SASCA with an FFT-optimized standard-SASCA baseline following the factor-graph implementation of [PPM17], where addition and subtraction factors are accelerated using FFT-based message updates. To ensure a fair comparison, both the standard and GRWP-SASCA are implemented in the same `CuPy`-accelerated environment. Even with FFT optimization, attacking a 5-layer sub-NTT component with standard SASCA takes about 1.6 hours in our implementation, making repeated experiments over multiple noise levels costly. For a 6-layer sub-NTT component, the required memory exceeds that available in our experimental environment, preventing direct execution; based on the computational-cost estimate in Section 5.4.2, a single execution would take approximately 4 hours if sufficient memory were available. Compared with the approximately 3.5 minutes required by GRWP-SASCA under the same 6-layer configuration, this corresponds to an estimated runtime reduction by a factor of approximately 68.

Therefore, for the controlled comparison between standard SASCA and GRWP-SASCA, we use a 4-layer sub-NTT component and evaluate both methods under the same leakage model and noise settings. We report both the attack success rate and the execution time, highlighting the effect of region-wise pruning on computational efficiency and recovery robustness, which is depicted in Figure 12. Small non-monotonic fluctuations or plateaus may appear across adjacent noise levels. In this setting, a single GRWP-SASCA trial takes about 14 seconds, whereas a standard SASCA trial takes about 10 minutes.

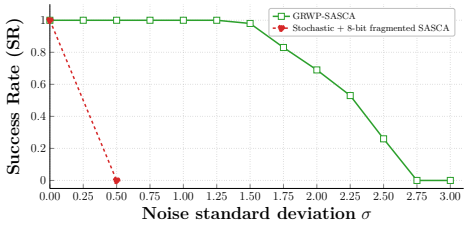


Figure 9: Success rate comparison between GRWP-SASCA and the stochastic 8-bit fragmented SASCA model under different noise levels with a single trace.

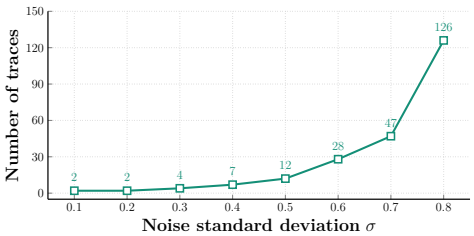


Figure 10: Traces required for successful key recovery vs. noise level for first-order masked ML-DSA.

Interestingly, in this controlled experiment, the success rate of standard SASCA decreases earlier and remains slightly lower than that of GRWP-SASCA at some noise levels. We hypothesize that this behavior is related to the dense loopy structure of the complete factor graph: full-graph BP may repeatedly propagate correlated evidence through many short cycles, which can lead to overconfident or less stable marginal beliefs under noisy leakage. In contrast, GRWP-SASCA performs localized BP followed by pruning, which may reduce the effect of such short-loop amplification by carrying forward only concentrated soft beliefs. We emphasize that this observation is empirical and mainly serves to show that the efficiency gain of GRWP-SASCA does not come at the cost of reduced recovery robustness.

Evaluation against 8-bit Fragmented SASCA Models. To complement the discussion of fragment-based SASCA approaches in the state of the art, we construct 8-bit fragmented SASCA baselines and compare them with the proposed full-domain GRWP-SASCA model. We evaluate a Hamming-weight-only 8-bit fragmented SASCA model for the ML-DSA NTT. This model follows the intuition of prior attacks on 32-bit Boolean implementations [KPP20], where word-level leakage is often represented through Hamming-weight or small-cluster information. In our experiment, each coefficient is split into four 8-bit blocks, and addition and subtraction are modeled with explicit carry and borrow variables across byte blocks. To isolate the limitation of the leakage abstraction and the fragmented graph structure, we conduct the experiment in a noiseless setting with $\sigma = 0$. Thus, the Hamming weight of each leakage-bearing intermediate is assumed to be observed without noise.

Even under this noiseless Hamming-weight setting, the fragmented model only remains useful for the earliest NTT layers. This is because the original ML-DSA secret coefficients lie in the small domain $[-\eta, \eta]$, so the first few layers inherit strong low-entropy priors from the input coefficients. In our simulations, the Hamming-weight based fragmented model can recover coefficients only up to the first three layers. When the depth exceeds three layers, the model can no longer reliably recover a non-negligible fraction of the newly exposed coefficients in $\mathbf{N}_j \setminus \mathbf{N}_3$ for $j > 3$. This is insufficient for the subsequent lattice-reduction step, since three layers do not provide enough correct coefficients to recover the full secret polynomial within an acceptable time.

We then evaluate a stronger fragmented model using the stochastic-model-based likelihoods described in Section 2.4. For each word-level variable, we first evaluate the likelihood over its complete candidate domain using the full-word stochastic model. The resulting distribution is then marginalized onto each of the four 8-bit fragments, and the four marginal distributions are assigned as leakage-derived priors to the corresponding fragment variables. Compared with Hamming-weight classes, these likelihoods provide value-dependent priors and therefore significantly improve the informativeness of the local evidence. As expected, this model performs better than the Hamming-weight-only variant and can support deeper recovery. However, the fragmented representation still

becomes unstable in deeper layers. As shown in Figure 9, the stochastic 8-bit fragmented model achieves full recovery only in the noiseless case ($\sigma = 0$), while its success rate already drops to zero at $\sigma = 0.5$. In our simulations, when the depth exceeds four layers, the model can no longer reliably recover a non-negligible fraction of the newly exposed coefficients in $\mathbf{N}_j \setminus \mathbf{N}_4$ for $j > 4$. This shows that stronger leakage priors alleviate, but do not eliminate, the difficulty caused by the fragmented arithmetic factor graph. We also evaluated the corresponding masked fragmented variant under a highly favorable simulated setting, using stochastic-model-based likelihoods with $\sigma = 0.1$. For a 4-layer sub-NTT component, we attempted recovery with 10 traces. Nevertheless, the resulting coefficient estimates did not exhibit reliable convergence: the recovered values were highly unstable and appeared nearly random over the target coefficients. The computation was also costly, taking approximately 35 minutes per trace on average. These results indicate that the direct masked fragmented model is not a practical recovery strategy even under very low-noise simulated leakage.

These results highlight two limitations of directly transferring Keccak-style fragmented SASCA to the ML-DSA NTT. First, Hamming-weight leakage is too coarse for large-modulus arithmetic variables: even if the Hamming weight is observed without noise, many candidates still remain indistinguishable. Second, even with stronger stochastic-model-based likelihoods, the byte-level arithmetic graph introduces substantial structural overhead. Addition and subtraction require explicit carry and borrow chains across byte blocks, while Montgomery multiplication is even harder to decompose into independent byte-level factors. Since the Montgomery product depends jointly on all byte blocks of the input word and the modular reduction terms, its factor evaluation still has word-level computational and temporary-memory cost. Only after this factor has been evaluated can the resulting messages be represented on the four byte-level output blocks. Thus, byte fragmentation reduces the apparent candidate size of individual variable nodes, but it does not remove the coupled arithmetic dependencies that BP must handle. Montgomery multiplication, carry/borrow propagation, and lazy-reduction dependencies still couple different byte blocks and create many auxiliary variables and short loops. This is why our main evaluation keeps coefficient-level arithmetic variables and addresses scalability through region-wise expansion and pruning, rather than through byte-level fragmentation.

5.3.1 Impact of Pruning and Region Strategies.

We evaluate how the pruning configuration and region granularity affect the performance of GRWP-SASCA.

Evaluation on the Pruning Threshold. In GRWP-SASCA, the pruning threshold τ controls the number of candidates retained after BP. The pruning rule follows Algorithm 4: candidates whose posterior probabilities exceed τ are retained, while a mean-based fallback is used when no candidate satisfies the threshold. Figure 11 evaluates the recovery success rate for a 6-layer sub-NTT component of ML-DSA under different threshold settings at $\sigma = 1.50$. For ML-DSA-44, the input secret coefficients have $\eta = 2$, so the initial candidate set has size $2\eta + 1 = 5$. A threshold above 0.5 tends to behave like a single-candidate decision, since at most one normalized probability can exceed 0.5. This may prematurely discard the correct candidate under noisy leakage. In contrast, a lower threshold allows multiple candidates to be retained, improving robustness against uncertainty in the posterior distribution. In our vectorized implementation, changing τ has little impact on the measured runtime for this experiment, since the dominant BP updates are still executed by the same batched `CuPy` kernels. Therefore, Figure 11 focuses on the effect of τ on recovery success rate.

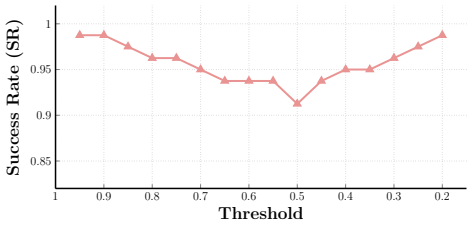


Figure 11: Recovery success rate for a 6-layer ML-DSA sub-NTT component under different pruning thresholds at $\sigma = 1.50$.

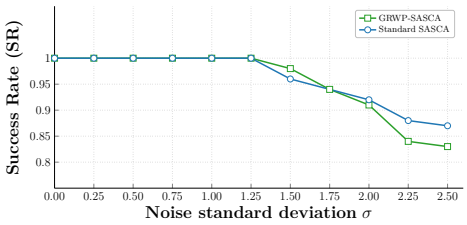


Figure 12: Success-rate comparison between GRWP-SASCA and standard SASCA on the simulated 4-layer ML-DSA sub-NTT component.

Evaluation of Structural Pruning. We further evaluate the impact of structural pruning in the masked multi-trace setting. Both variants follow the same trace-wise evidence-aggregation strategy described in Section 4.3: for each trace, the corresponding share region is merged into the active graph, BP is performed, and the resulting information is transferred to the virtual unmasked variables. The only difference is whether the temporary masked structures are removed after each trace. In the default GRWP-SASCA schedule, these trace-specific share variables and recombination factors are structurally pruned once their information has been propagated to the virtual layer. In contrast, the no-structural-pruning variant keeps all merged masked structures in $\mathcal{G}_{\text{active}}$ throughout the multi-trace procedure.

Due to device-memory limitations, we conduct this ablation on a 3-layer sub-NTT component under the masked setting with $\sigma = 0.5$. Figure 13 compares the two variants in terms of runtime and recovery success rate. For the no-structural-pruning variant, the runtime is measured as the average per-trace runtime, reflecting the cost of retaining all trace-specific masked structures in the active graph. For the structural-pruning variant, we report the maximum runtime of $\mathcal{G}_{\text{active}}$ during the aggregation process, since temporary masked structures are removed once their information has been transferred to the virtual layer. The comparison confirms that structural pruning effectively controls the growth of the active graph, at the expense of a moderate degradation in recovery performance.

Evaluation of Region Granularity. We further evaluate how the granularity of regions affects the efficiency–correctness trade-off of GRWP-SASCA. Following the notation in Section 4.2, the elementary region is the BFU subgraph $\mathcal{G}_{j,i,k}$. We compare three ways of grouping these elementary BFU regions before each BP-and-pruning step. In the *one-BFU* setting, each region contains a single BFU subgraph $\mathcal{G}_{j,i,k}$. In the *one-layer* setting, all BFU regions belonging to the same leakage layer are merged together, i.e., $\mathcal{R}_j^{\text{layer}} = \bigcup_{(i,k)\in\mathcal{I}_j} \mathcal{G}_{j,i,k}$. This setting integrates the leakage and constraints associated with the layer-j upper-side variables $\mathbf{U}_j$ before pruning, and therefore processes the newly exposed input-ancestor set $\mathbf{N}_j \setminus \mathbf{N}_{j-1}$ at the layer level. In the *full-trace* setting, all leakage-bearing BFU regions from one trace are merged before pruning, $\mathcal{R}^{\text{trace}} = \bigcup_j \bigcup_{(i,k)\in\mathcal{I}_j} \mathcal{G}_{j,i,k}$.

We run the two-trace attack under $\sigma = 0.5$. Figure 14 reports the runtime of BP on the largest active graph $\mathcal{G}_{\text{active}}$ encountered during the GRWP-SASCA procedure under the two-trace setting with $\sigma = 0.5$ and the relevant success rate.

These preliminary results indicate that finer-grained regions can substantially reduce the cost of localized BP and make greedy pruning more practical. By contrast, coarser regions lead to much larger active factor graphs, so the computational overhead increases quickly even before considering the final recovery rate. This observation supports the use of fine-grained BFU-level regions in our main experiments.

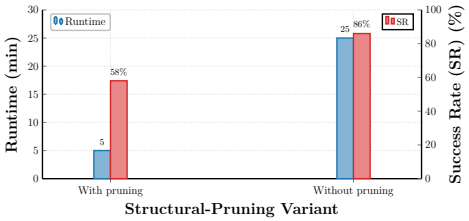


Figure 13: Effect of structural pruning on the runtime of BP over $\mathcal{G}_{\text{active}}$ and the recovery success rate under the masked setting with $\sigma = 0.5$.

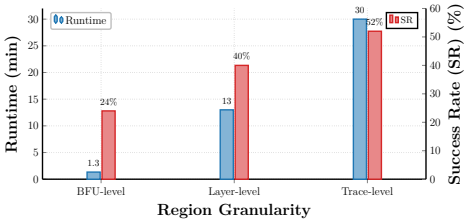


Figure 14: Effect of region granularity on the runtime of BP over the largest active graph $\mathcal{G}_{\text{active}}$ in GRWP-SASCA under the two-trace setting with $\sigma = 0.5$.

5.4 Comparison with Standard SASCA

We compare our method with power analysis attacks that also target the NTT [PPM17, PP19] employing SASCA. A concise comparison is presented in Table 3. Next, we present a more detailed comparison.

While these prior studies established the foundation for attacking NTT-based PQC schemes via factor graphs, they face significant scalability challenges when applied to schemes with a large modulus q, such as ML-DSA. As analyzed in Section 1, their methods typically involve maintaining a global factor graph where the search space for all intermediate variables remains largely unconstrained until the final BP convergence. For ML-DSA, the large modulus q results in extremely large message distributions and factor node computations, leading to prohibitive memory consumption and computational overhead. In comparison, the main challenges addressed in our work are as follows:

Table 3: Concise comparison with prior SASCA attacks on NTT-based implementations.

Setting	Method	Graph scope	Message domain	For Large Modulus
Unprotected	Primas et al. [PPM17]	Global	Dense	Impractical
	Pessl et al. [PP19]	Global	Dense	Impractical
	Our Section 4.2	GRWP	Pruned	Practical
Masked	Primas et al. [PPM17]	Global	Dense	Impractical
	Pessl et al. [PP19]	Global	Dense	Impractical
	Our Section 4.3	GRWP	Pruned	Practical

5.4.1 The Memory Overhead during SASCA

We estimate the memory consumption by counting the message probability distributions stored on variable–factor edges. For a factor graph $\mathcal{G} = (\mathcal{V}, \mathcal{F}, \mathcal{E})$, the message memory is estimated as

$$M(\mathcal{G}) \approx 2 \cdot 4 \sum_{(v,f)\in\mathcal{E}} |\mathcal{C}_v| \quad \text{bytes},$$

where $\mathcal{C}_v$ is the candidate domain of variable node v. The factor 4 corresponds to `float32` storage, and the factor 2 accounts for the two message directions on each variable–factor edge. Thus, variables with larger degree are counted multiple times through their incident edges. Prior-probability edges are omitted, since their contribution is small compared with the dense messages over large arithmetic domains.

For the unpruned standard graph, $|\mathcal{C}_v|$ is assigned according to the type and layer index of v. Specifically, if v is an upper or lower coefficient variable in layer k, we set

$$|\mathcal{C}_v| = \mathcal{D}_k^{\text{coef}} = \min\left((2\eta+1)^{2^{k-1}}, kq\right),$$

If v is a Montgomery product variable in layer k, we set

$$|\mathcal{C}_v| = \mathcal{D}_k^{\text{prod}} = \min\left((2\eta+1)^{2^{k-1}}, q\right).$$

These domain sizes are then substituted into the message-counting rule above for all upper coefficients, lower coefficients, and Montgomery product variables in the full active sub-NTT. Unless stated otherwise, all message-memory estimates in this subsection are reported for one j-layer sub-NTT component, since the 2^{L-j} independent components are processed sequentially. In contrast, $|\mathbf{N}_j|$ denotes the total number of input coefficients covered across all 2^{L-j} components of one polynomial.

Table 4: Estimated peak per-component BP-message memory of standard SASCA and GRWP-SASCA for unprotected ML-DSA-44.

j-layer	$\lvert\mathbf{N}_j\rvert$	Standard (GB)	GRWP (GB)	Estimated Reduction
$j=5$	248	28.35	0.69	≈41×
$j=6$	252	122.63	0.81	≈151×
$j=7$	254	397.11	0.94	≈422×
$j=8$	255	1137.88	1.06	≈1073×

For GRWP-SASCA, the same memory-counting rule is applied to the largest active graph that appears during the region-wise execution, rather than to the full sub-NTT graph. This peak active-graph memory is the relevant metric because regions are processed sequentially: after localized BP, temporary structures can be removed, and their information is carried forward only through retained beliefs on the remaining variables. In our estimates, variables already recovered by previous regions are counted as singleton domains, affected input-side variables remain bounded by the small secret domain $2\eta+1$, and only the current active BFU region keeps the remaining dense coefficient and product domains. Therefore, the GRWP values in Table 4 represent the estimated maximum message memory resident at any time during the attack.

For masked implementations, the same message-counting rule is applied to the 3D-FG. Let d denote the masking order, so that each sensitive variable is represented by $d+1$ shares, and let t denote the number of traces. In the standard full 3D-FG, all trace-specific share graphs and all recombination constraints are kept simultaneously. Hence, the peak memory grows with both t and d. In contrast, GRWP-SASCA processes traces sequentially. For each trace, only the corresponding share structure is merged into the active graph, BP is executed, and the trace-specific structure is then removed after its contribution has been extracted as trace-specific evidence on the virtual variables. The trace-specific evidence terms are aggregated before final BP and value-level pruning are applied on the virtual layer. Therefore, the peak message memory of GRWP-SASCA contains the virtual active region plus one trace-specific share structure, rather than all t trace graphs simultaneously.

Table 5 reports the estimated message-memory consumption for masked ML-DSA-44. To visualize the overall trend, Figure 15 plots the corresponding memory reduction ratio of GRWP-SASCA over the standard SASCA as a function of the number of traces t and the masking order d. The different surfaces correspond to different sub-NTT depths j.

5.4.2 The Computational Overhead during SASCA

We estimate the dominant computational overhead under the same `CuPy`-based GPU implementation used in our experiments. In standard SASCA, the main cost comes from dense addition and subtraction factors over the large ML-DSA modulus, which are evaluated using FFT-based convolution. Montgomery-product factors are implemented by

Table 5: Estimated peak per-component BP-message memory for d-order masked ML-DSA-44 with t traces.

j-layer	$\lvert\mathbf{N}_j\rvert$	standard (GB)	GRWP (GB)	Symbolic Reduction
$j=5$	248	$122.88t(d+1)+11.13t+28.35$	$0.75d+1.80$	$\approx 163.84t(d+1.09)/(d+2.40)$
$j=6$	252	$337.67t(d+1)+48.23t+122.63$	$0.87d+2.12$	$\approx 388.13t(d+1.14)/(d+2.44)$
$j=7$	254	$887.13t(d+1)+156.41t+397.11$	$1.00d+2.44$	$\approx 887.13t(d+1.18)/(d+2.44)$
$j=8$	255	$2253.80t(d+1)+448.68t+1137.88$	$1.12d+2.75$	$\approx 2012.32t(d+1.20)/(d+2.46)$

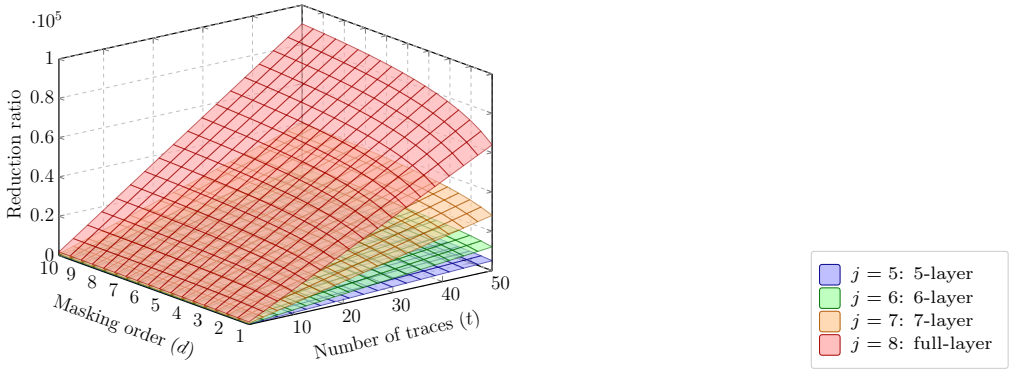


Figure 15: Estimated memory reduction ratio of GRWP-SASCA over the standard SASCA for masked ML-DSA-44. The reduction increases with the number of traces because the standard keeps all trace-specific share graphs simultaneously, whereas GRWP-SASCA processes traces sequentially and structurally prunes each trace-specific graph after BP.

vectorized table operations and are not the dominant runtime component in our implementation.

In the unprotected setting, GRWP-SASCA reduces this cost by value-level pruning. Once the active candidate domains become sparse, the addition and subtraction factors are evaluated over the retained candidate sets instead of over modulus-sized dense domains. Thus, the main speedup comes from replacing dense FFT-based Add/Sub updates by sparse candidate-domain updates after localized BP and pruning. For an unprotected j-layer sub-NTT component, absorbing the constant cost of the corresponding message updates into T_{FFT}, the FFT-dominated cost of standard SASCA is estimated as

$$T^{\mathrm{FFT}}_{\mathrm{standard,unprotected}} \approx I_{\mathrm{BP}} \cdot T_{\mathrm{FFT}} \cdot j \cdot 2^j.$$

For a j-layer masked sub-NTT component with t traces, masking order d, and I_{BP} BP iterations, the FFT-dominated cost of standard SASCA is estimated as

$$T^{\mathrm{FFT}}_{\mathrm{standard,masked}} \approx I_{\mathrm{BP}} \cdot T_{\mathrm{FFT}} \cdot j \cdot 2^j \cdot \big(t(d+1)+1+t\big).$$

The terms $t(d+1)$, 1, and t correspond to the trace-specific share graphs, the virtual unmasked layer, and the trace-to-virtual recombination structures, respectively. In contrast, masked GRWP-SASCA processes traces sequentially and removes each trace-specific share structure after extracting its trace-specific evidence. Since the retained virtual-layer candidates are sparse after the previous region-wise pruning steps, the remaining dense FFT-dominated work mainly comes from the current trace-side region and its connection to the virtual layer:

$$T^{\mathrm{FFT}}_{\mathrm{GRWP,masked}} \approx I_{\mathrm{BP}} \cdot T_{\mathrm{FFT}} \cdot j \cdot 2^{j-1} \cdot t.$$

These expressions are intended to compare the dominant dense-update cost. They exclude pruning, graph construction, message normalization, and GPU scheduling overheads, which are included in the measured runtimes reported in the experiments.

Using the measured 5-layer runtime reported in Section 5.3 and the above cost model, we estimate that a 6-layer standard-SASCA execution would take approximately 4 hours if sufficient memory were available. Moreover, the ratio between the two masked-setting estimates is $2(d+2+1/t)$, which approaches approximately $2d+4$ as the number of traces increases.

6 Attacking a Real Device

Section Goal. To validate our approach in a practical setting, we apply GRWP-SASCA to power traces collected from a physical implementation. Only the source of the leakage-derived priors differs from the simulated evaluation; the factor-graph structure, region-wise schedule, pruning configuration, and BP procedure remain unchanged. Therefore, the memory consumption and computational complexity of the inference stage are identical to those in the simulated evaluation. This experiment evaluates whether GRWP-SASCA remains effective under real-device leakage and noise.

6.1 Attack Setup

Our target is the reference ML-DSA implementation [2] on a 32-bit STM32F4 microcontroller running at 8 MHz. For power acquisition, we utilize the PicoScope 5244D oscilloscope with an approximate sampling frequency of 32 MHz. Our experiments are conducted in a controlled environment using the ChipWhisperer 308 board.

PoI Selection. For each BFU execution, we first identify the corresponding time window after trace alignment. Assume that this BFU window contains N_{win} samples. We select PoIs using a profiling-based Hamming-weight correlation analysis. For each sample position, we compute the absolute Pearson correlation between the measured leakage and the Hamming weight of the known BFU intermediate value in the profiling traces. The sample positions are then ranked by their correlation scores, and the top-$\lfloor N_{\text{win}}/2 \rfloor$ samples are retained as PoIs.

6.2 Attack Results and Analysis

During the profiling phase, we collected power consumption traces from the first six layers of the NTT. To ensure high predictive accuracy for the stochastic model described in Section 2.4, we acquired a large-scale dataset comprising 2 million power traces, which occupy approximately 3.7 TB of disk space. Given the large modulus $q = 8\,380\,417$, the upper coefficient can in principle take q possible values. Assuming this full candidate space, the stochastic model described in Section 2.4 ranks the correct candidate at an average position of approximately 7.5×10^4.

For BP, we use a maximum of 20 loopy iterations and evaluate each setting over 20 independent randomized experiments. For the unprotected implementation, the single-trace attack achieves a success rate of 1.0, with an average runtime of approximately 3.5 minutes per 6-layer sub-NTT component. Since $L = 8$, each secret polynomial contains $2^{L-6} = 4$ independent 6-layer components, resulting in an average recovery time of approximately 15 minutes per polynomial when these components are processed sequentially. The four polynomials in the ML-DSA-44 secret vector are independent and are processed in parallel on four devices; hence, the recovery time remains approximately 15 minutes.

For the first-order masked implementation, GRWP-SASCA recovers the secret vector with a success rate of 1.0 using eight power traces. Under the four-device execution

[2] The specific implementation is detailed in the ML-DSA repositories on GitHub: https://github.com/pq-crystals/dilithium or https://github.com/openHiTLS/openHiTLS/tree/main/crypto/mldsa

schedule used in the runtime evaluation in Section 5.3, processing one trace for one secret polynomial through the first six NTT layers takes approximately 6 minutes. Since the eight traces and four secret polynomials are processed sequentially, recovering the complete secret vector requires approximately 3.2 hours. The corresponding runtime and peak message memory usage are reported in Table 6.

Table 6: Performance of GRWP-SASCA on unprotected and masked implementations.

Implementation	Number of Traces	SR	Avg. Recovery Time	Message Memory
Unprotected	1	1.0	$\approx$ 15 min	$\approx$ 0.9 GB (peak)
First-order masked	8	1.0	$\approx$ 3.2 hours	$\approx$ 8.1 GB (peak)

7 Conclusion

Practical Takeaways. GRWP-SASCA should be viewed as an attacker-configured heuristic framework rather than an automatic graph-reduction algorithm. For a large-domain arithmetic factor graph, the attacker must select the regions, retained interface variables, $\mathcal{V}_{\text{remove}}$, integration order, and pruning threshold according to the computation schedule, available leakage, and variables required by subsequent inference. In the ML-DSA NTT, for example, BFUs provide natural elementary regions, and the first layer is a suitable starting point because the input secret coefficients have small candidate domains. This allows the attack to first obtain concentrated beliefs from low-entropy variables and then propagate them to deeper BFU regions.

Structural pruning should remove temporary components only after localized BP has transferred their useful information to retained interface or secret-relevant variables. Value-level pruning should be applied only when the posterior is sufficiently concentrated to remove unlikely candidates without an unacceptable risk of discarding the correct one; the retained beliefs are then reused as soft priors in subsequent expansions. In multi-trace settings, trace-specific evidence should first be aggregated on the shared virtual variables, and value-level pruning should be delayed until all traces for the current region have been processed. The concrete region granularity and pruning threshold remain target-dependent and should be selected based on empirical evaluation.

Potential Applications. Although our experiments focus on the ML-DSA NTT, the same principle can be transferred more directly to other NTT-based lattice schemes. For example, qTESLA also uses NTT-style polynomial arithmetic with repeated modular butterfly operations, so GRWP-SASCA can be adapted by changing the modulus, twiddle schedule, candidate domains, and leakage model.

The same principle may also be relevant to other multi-precision modular-arithmetic implementations. For example, ECC field multiplication and reduction on 32-bit platforms typically represent a 256-bit field element by several word-size variables. These computations involve multiplication, accumulation, carry propagation, and modular reduction, and therefore induce arithmetic factor graphs with strongly coupled 32-bit variable nodes. In such settings, a single trace may not provide sufficiently concentrated likelihoods for aggressive pruning of 32-bit variables. A possible GRWP-SASCA instantiation would aggregate evidence from multiple traces on retained interface variables before applying value-level pruning and expanding to adjacent arithmetic regions. However, the region definitions, shared variables, and leakage model would be implementation-specific.

For Keccak-like implementations, the situation is different from the large-domain arithmetic setting targeted in this work. Although Keccak may be implemented on 32-bit words, its algorithmic constraints are mainly Boolean or bitwise operations. Therefore, when large candidate spaces arise in such implementations, fragmenting them into smaller

bit- or byte-level domains is a natural first choice. In this regime, bit-level, clustered, or fragment-based SASCA models are naturally aligned with the computation structure. Thus, fragmentation greatly reduces both message storage and factor-update costs without introducing the carry, borrow, partial-product, or reduction dependencies that arise in modular arithmetic. Consequently, for Keccak-like implementations, direct fragmentation already provides an effective way to control the BP complexity, and applying GRWP-SASCA is generally unnecessary. GRWP-SASCA is instead intended for large-domain arithmetic settings where such fragmentation does not effectively remove the BP bottleneck.

Future Work. While this work empirically demonstrates the efficiency of GRWP-SASCA for large-domain arithmetic factor graphs, the pruning mechanism inevitably introduces approximation and potential information loss. A promising direction for future research is to develop a formal framework for quantifying this loss. In particular, theoretical bounds on the approximation error with respect to full-graph inference, as well as conditions under which pruning preserves the correct candidate with high probability, would provide stronger guarantees for region-wise SASCA.

Another important direction is to apply GRWP-SASCA to other large-scale cryptographic implementations. Beyond the ML-DSA NTT considered in this work, many post-quantum and conventional cryptographic algorithms involve large-domain modular arithmetic, multi-precision operations, or repeated structured arithmetic layers. Instantiating the same region-wise inference and pruning framework on these targets would further clarify its general applicability and practical limits.

Concluding Remarks. In this work, we analyze the challenges of applying SASCA to large-scale arithmetic computations. We show that standard full-graph SASCA can become prohibitive when the target factor graph contains large-domain variables and dense modular constraints. To address this bottleneck, we propose GRWP-SASCA, a region-wise inference framework based on localized BP, structural pruning, and value-level pruning. By retaining informative interface variables and removing temporary graph structures after their information has been propagated, GRWP-SASCA reduces the memory and computational cost of BP.

We validate GRWP-SASCA on the ML-DSA NTT, a representative large-domain arithmetic computation with a multi-layered structure. Our simulated and real-device evaluations show that GRWP-SASCA significantly improves the speed and memory efficiency of SASCA under the considered leakage settings, making attacks on large-scale and masked NTT implementations practical in our experimental setup.

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. pages 441–460, 2020.

[ABC+22] Melissa Azouaoui, Olivier Bronchain, Gaëtan Cassiers, Clément Hoffmann, Yulia Kuzovkova, Joost Renes, Markus Schönauer, Tobias Schneider, François-Xavier Standaert, and Christine van Vredendaal. Leveling Dilithium against leakage: Revisited sensitivity analysis and improved implementations. *IACR Cryptol. ePrint Arch.*, page 1406, 2022.

[ABC+23] Melissa Azouaoui, Olivier Bronchain, Gaëtan Cassiers, Clément Hoffmann, Yulia Kuzovkova, Joost Renes, Tobias Schneider, Markus Schönauer, François-Xavier Standaert, and Christine van Vredendaal. Protecting Dilithium against

leakage revisited sensitivity analysis and improved implementations. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2023(4):58–79, 2023.

[BBE+24] 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. *J. Cryptol.*, 37(1):5, 2024.

[BDK+18] Joppe W. Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS - Kyber: A CCA-secure module-lattice-based KEM. pages 353–367, 2018.

[BDPA11] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. The Keccak reference, version 3.0. https://keccak.team/files/Keccak-reference-3.0.pdf, January 2011.

[BS21] Olivier Bronchain and François-Xavier Standaert. Breaking masked implementations with many shares on 32-bit software platforms or when the security order does not matter. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(3):202–234, 2021.

[CDSU23] Gaëtan Cassiers, Henri Devillez, François-Xavier Standaert, and Balazs Udvarhelyi. Efficient regression-based linear discriminant analysis for side-channel security evaluations towards analytical attacks against 32-bit implementations. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2023(3):270–293, 2023.

[CGTZ23] Jean-Sébastien Coron, François Gérard, Matthias Trannoy, and Rina Zeitoun. Improved gadgets for the high-order masking of Dilithium. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2023(4):110–145, 2023.

[CK14] Marios O. Choudary and Markus G. Kuhn. Efficient stochastic methods: Profiled attacks beyond 8 bits. In Marc Joye and Amir Moradi, editors, *Smart Card Research and Advanced Applications - 13th International Conference, CARDIS 2014, Paris, France, November 5-7, 2014. Revised Selected Papers*, Lecture Notes in Computer Science, pages 85–103. Springer, 2014.

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

[DEMS21] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1.2: Lightweight authenticated encryption and hashing. *Journal of Cryptology*, 34(3):33, 2021.

[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. https://doi.org/10.13154/tches.v2018.i1.238-268.

[GGSB20] Qian Guo, Vincent Grosso, François-Xavier Standaert, and Olivier Bronchain. Modeling soft analytical side-channel attacks from a coding theory viewpoint. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(4):209–238, 2020.

[GRO18] Joey Green, Arnab Roy, and Elisabeth Oswald. A systematic study of the impact of graphical models on inference-based attacks on AES. In Begül Bilgin and Jean-Bernard Fischer, editors, *Smart Card Research and Advanced Applications, 17th International Conference, CARDIS 2018, Montpellier, France, November 12-14, 2018, Revised Selected Papers*, Lecture Notes in Computer Science, pages 18–34. Springer, 2018.

[HMS+23] Julius Hermelink, Erik Mårtensson, Simona Samardjiska, Peter Pessl, and Gabi Dreo Rodosek. Belief propagation meets lattice reduction: Security estimates for error-tolerant key recovery from decryption errors. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2023(4):287–317, 2023.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[Koc96] Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Neal Koblitz, editor, *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.

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

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Monaco / French Riviera, May 30 - June 3, 2010. Proceedings*, volume 6110 of *Lecture Notes in Computer Science*, pages 1–23. Springer, 2010.

[LS15] Adeline Langlois and Damien Stehlé. Worst-case to average-case reductions for module lattices. *Des. Codes Cryptogr.*, 75(3):565–599, 2015.

[Mac03] David J. C. MacKay. *Information theory, inference, and learning algorithms*. Cambridge University Press, 2003.

[Nat01] National Institute of Standards and Technology. Advanced encryption standard (AES). Federal Information Processing Standards Publication (FIPS) 197, U.S. Department of Commerce, November 2001.

[Nat24a] National Institute of Standards and Technology. FIPS 203: Module-lattice-based key-encapsulation mechanism standard. Federal Information Processing Standards Publication FIPS 203, National Institute of Standards and Technology, 2024.

[Nat24b] National Institute of Standards and Technology. FIPS 204: Module-lattice-based digital signature standard. Federal Information Processing Standards Publication FIPS 204, National Institute of Standards and Technology, 2024.

[OVCG25] Paco Azevedo Oliveira, Andersson Calle Viera, Benoît Cogliati, and Louis Goubin. Uncompressing Dilithium's public key. In Yael Tauman Kalai and Seny F. Kamara, editors, *Advances in Cryptology - CRYPTO 2025 - 45th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2025, Proceedings, Part I*, volume 16000 of *Lecture Notes in Computer Science*, pages 417–443. Springer, 2025.

[Pea82] Judea Pearl. Reverend Bayes on inference engines: A distributed hierarchical approach. In David L. Waltz, editor, *Proceedings of the National Conference on Artificial Intelligence, Pittsburgh, PA, USA, August 18-20, 1982*, pages 133–136. AAAI Press, 1982.

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

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

[SLP05] Werner Schindler, Kerstin Lemke, and Christof Paar. A stochastic model for differential side channel cryptanalysis. In Josyula R. Rao and Berk Sunar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2005, 7th International Workshop, Edinburgh, UK, August 29 - September 1, 2005, Proceedings*, volume 3659 of *Lecture Notes in Computer Science*, pages 30–46. Springer, 2005.

[VGS14] Nicolas Veyrat-Charvillon, Benoît Gérard, and François-Xavier Standaert. Soft analytical side-channel attacks. pages 282–296, 2014.

[YK21] Shih-Chun You and Markus G. Kuhn. Single-trace fragment template attack on a 32-bit implementation of Keccak. In Vincent Grosso and Thomas Pöppelmann, editors, *Smart Card Research and Advanced Applications - 20th International Conference, CARDIS 2021, Lübeck, Germany, November 11-12, 2021, Revised Selected Papers*, volume 13173 of *Lecture Notes in Computer Science*, pages 3–23. Springer, 2021.

[YKSH23] Shih-Chun You, Markus G. Kuhn, Sumanta Sarkar, and Feng Hao. Low trace-count template attacks on 32-bit implementations of ASCON AEAD. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2023(4):344–366, 2023.