Masking the GLP Lattice-Based Signature Scheme at Any Order

Gilles Barthe[1], Sonia Belaïd[2], Thomas Espitau[3], Pierre-Alain Fouque[4], Benjamin Grégoire[5], Mélissa Rossi[6,7], and Mehdi Tibouchi[8](✉)

[1] IMDEA Software Institute, Madrid, Spain
gilles.barthe@imdea.org
[2] CryptoExperts, Paris, France
sonia.belaid@cryptoexperts.com
[3] UPMC, Paris, France
thomas.espitau@lip6.fr
[4] Univ Rennes, Rennes, France
pierre-alain.fouque@univ-rennes1.fr
[5] Inria Sophia Antipolis, Sophia Antipolis, France
benjamin.gregoire@sophia.inria.fr
[6] Thales, Paris, France
[7] Département d'informatique de l'École normale supérieure de Paris, CNRS, PSL Research University, INRIA, Paris, France
melissa.rossi@ens.fr
[8] NTT Secure Platform Laboratories, Tokyo, Japan
tibouchi.mehdi@lab.ntt.co.jp

Abstract. Recently, numerous physical attacks have been demonstrated against lattice-based schemes, often exploiting their unique properties such as the reliance on Gaussian distributions, rejection sampling and FFT-based polynomial multiplication. As the call for concrete implementations and deployment of postquantum cryptography becomes more pressing, protecting against those attacks is an important problem. However, few countermeasures have been proposed so far. In particular, masking has been applied to the decryption procedure of some lattice-based encryption schemes, but the much more difficult case of signatures (which are highly non-linear and typically involve randomness) has not been considered until now.

In this paper, we describe the first masked implementation of a lattice-based signature scheme. Since masking Gaussian sampling and other procedures involving contrived probability distribution would be prohibitively inefficient, we focus on the GLP scheme of Güneysu, Lyubashevsky and Pöppelmann (CHES 2012). We show how to provably mask it in the Ishai–Sahai–Wagner model (CRYPTO 2003) at any order in a relatively efficient manner, using extensions of the techniques of Coron et al. for converting between arithmetic and Boolean masking. Our proof relies on a mild generalization of probing security that supports the notion of public outputs. We also provide a proof-of-concept implementation to assess the efficiency of the proposed countermeasure.

Keywords: Side-channel · Masking · GLP lattice-based signature

J. B. Nielsen and V. Rijmen (Eds.): EUROCRYPT 2018, LNCS 10821, pp. 354–384, 2018.
https://doi.org/10.1007/978-3-319-78375-8_12

1 Introduction

As the demands for practical implementations of postquantum cryptographic schemes get more pressing ahead of the NIST postquantum competition and in view of the recommendations of various agencies, understanding the security of those schemes against physical attacks is of paramount importance. Lattice-based cryptography, in particular, is an attractive option in the postquantum setting, as it allows to design postquantum implementations of a wide range of primitives with strong security guarantees and a level of efficiency comparable to currently deployed RSA and elliptic curve-based schemes. However, it poses new sets of challenges as far as side-channels and other physical attacks are concerned. In particular, the reliance on Gaussian distributions, rejection sampling or the number-theoretic transform for polynomial multiplication have been shown to open the door to new types of physical attacks for which it is not always easy to propose efficient countermeasures.

The issue has in particular been laid bare in a number of recent works for the case of lattice-based signature schemes. Lattice-based signature in the random oracle model can be roughly divided into two families: on the one hand, constructions following Lyubashevsky's "Fiat–Shamir with aborts" paradigm [23], and on the other hand, hash-and-sign signatures relying on lattice trapdoors, as introduced by Gentry, Peikert and Vaikuntanathan [19]. Attempts have been made to implement schemes from both families, but Fiat–Shamir signatures are more common (although their postquantum security is admittedly not as well grounded). The underlying framework is called Fiat–Shamir *with aborts* because, unlike RSA and discrete logarithm-based constructions, lattice-based constructions involve sampling from sets that do not admit a nice algebraic structure. A naïve sampling algorithm would leak partial key information, in much the same way as it did in early heuristic schemes like GGH and NTRUSign; this is avoided by forcing the output signature to be independent of the secret key using rejection sampling. Many instantiations of the framework have been proposed [15,21,23,24,27], some of them quite efficient: for example, the BLISS signature scheme [15] boasts performance and key and signature sizes roughly comparable to RSA and ECDSA signatures.

However, the picture becomes less rosy once physical attacks are taken into account. For instance, Groot Bruinderink et al. [20] demonstrated a cache attack targetting the Gaussian sampling of the randomness used in BLISS signatures, which recovers the entire secret key from the side-channel leakage of a few thousand signature generations. Fault attacks have also been demonstrated on all kinds of lattice-based signatures [6,17]. In particular, Espitau et al. recover the full BLISS secret key using a single fault on the generation of the randomness (and present a similarly efficient attack on GPV-style signatures). More recently, ACM CCS 2017 has featured several papers [18,26] exposing further side-channel attacks on BLISS, its variant BLISS–B, and their implementation in the strongSwan VPN software. They are based on a range of different side channels (cache attacks, simple and correlation electromagnetic analysis, branch tracing, etc.), and some of them target new parts of the signature generation algorithm, such as the rejection sampling.

In order to protect against attack such as these, one would like to apply powerful countermeasures like masking. However, doing so efficiently on a scheme like BLISS seems hard, as discussed in [18]. Indeed, the sampling of the Gaussian randomness in BLISS signature generation involves either very large lookup tables, which are expensive to mask efficiently, or iterative approaches that are hard to even implement in constant time–let alone mask. Similarly, the rejection sampling step involves transcendental functions of the secret data that have to be computed to high precision; doing so in masked form seems daunting.

However, there exist other lattice-based signatures that appear to support side-channel countermeasures like masking in a more natural way, because they entirely avoid Gaussians and other contrived distributions. Both the sampling of the randomness and the rejection sampling of signatures target uniform distributions in contiguous intervals. Examples of such schemes include the GLP scheme of Güneysu, Lyubashevsky and Pöppelmann [21], which can be seen as the ancestor of BLISS, and later variants like the Dilithium scheme of Ducas et al. [16] (but not Dilithium-G).

In this paper, we show how to efficiently mask the GLP scheme at any masking order, so as to achieve security against power analysis and related attacks (both simple power analysis and higher-order attacks like differential/correlation power analysis). This is to the best of our knowledge the first time a masking countermeasure has been applied to protect lattice-based signatures.

Related Work. Masking is a well-known technique introduced by Chari, Rao and Rohatgi at CHES 2002 [7] and essentially consists in splitting a secret value into $d+1$ ones (d is thus the masking order), using a secret sharing scheme. This will force the adversary to read many internal variables if he wants to recover the secret value, and he will gain no information if he observes fewer than d values. The advantage of this splitting is that linear operations cost nothing, but the downside is that non-linear operations (such as the AES S-box) can become quite expensive. Later, Ishai, Sahai and Wagner [22] developed a technique to prove the security of masking schemes in the threshold probing model (ISW), in which the adversary can read off at most d wires in a circuit. Recently, Duc, Dziembowski and Faust [14] proved the equivalence between this threshold model and the more realistic noisy model, in which the adversary acquires leakage on *all* variables, but that leakage is perturbed with some noise distribution, as is the case in practical side-channel attacks. Since the ISW model is much more convenient for designing and proving masking countermeasures, it is thus preferred, as the equivalence results of Duc et al. ultimately ensure that a secure implementation in the ISW model at a sufficiently high masking order is going to be secure against practical side-channel attacks up to a given signal-to-noise ratio.

Masking has been applied to lattice-based encryption schemes before [28,29]. However, in these schemes, only the decryption procedure needs to be protected, and it usually boils down to computing a scalar product between the secret key and the ciphertext (which is a linear operation in the secret data) followed by a comparison (which is non-linear, but not very difficult to mask). Oder et al. [25] point out a number of issues with those masked decryption algorithms, and describe another one, for a CCA2-secure version of Ring-LWE public-key encryption.

Our Results. Masking lattice-based signatures, even in the comparatively simple case of GLP, turns out to be surprisingly difficult—possibly more so than any of the previous masking countermeasures considered so far in the literature. The probabilistic nature of signature generation, as well as its reliance on rejection sampling, present challenges (both in terms of design and of proof techniques) that had not occurred in earlier schemes, most of them deterministic. In addition, for performance reasons, we are led to require a stronger security property of the original, unprotected signature scheme itself, which we have to establish separately. More precisely, the following issues arise.

Conversion between Boolean and mod-p arithmetic masking. Most steps of the signing algorithm involve linear operations on polynomials in the ring $\mathcal{R} = \mathbb{Z}_p[x]/(x^n + 1)$. They can thus be masked very cheaply using mod-p arithmetic masking: each coefficient is represented as a sum of $d+1$ additive shares modulo p. For some operations, however, this representation is less convenient.

This is in particular the case for the generation of the randomness at the beginning of the algorithm, which consists of two polynomials $\mathbf{y}_1, \mathbf{y}_2$ with uniformly random coefficients in a subinterval $[-k, k]$ of $\mathbb{Z}_p$. Generating such a random value in masked form is relatively easy with Boolean masking, but seems hard to do efficiently with arithmetic masking. Therefore, we have to carry out a conversion from Boolean masking to mod-p arithmetic masking. Such conversions have been described before [11,13], but only when the modulus p was a power of 2. Adapting them to our settings requires some tweaks.

Similarly, the rejection sampling step amounts to checking whether the polynomials in the signature have their coefficients in another interval $[-k', k']$. Carrying out the corresponding comparison is again more convenient with Boolean masking, and hence we need a conversion algorithm in the other direction, from mod-p arithmetic masking to Boolean masking. We are again led to adapt earlier works on arithmetic-to-Boolean masking conversion [12,13] to the case of a non-prime modulus.

Security of the signature scheme when revealing the "commitment" value. One of the operations in signature generation is the computation of a hash function mapping to polynomials in $\mathcal{R}$ of a very special shape. Masking the computation of this hash function would be highly inefficient and difficult to combine with the rest of the algorithm. Indeed, the issue with hashing is not obtaining a masked bit string (which could be done with something like SHA-3), but expanding that bit string into a random-looking polynomial $\boldsymbol{c}$ of fixed, low Hamming weight in masked form. The corresponding operation is really hard to write down as a circuit. Moreover, even if that could be done, it would be terrible for performances because subsequent multiplications by $\boldsymbol{c}$ are no longer products by a known sparse constant, but full-blown ring operations that have to be fully masked.

But more importantly, this masking *should* intuitively be unnecessary. Indeed, when we see the signature scheme as the conversion of an identification protocol under the Fiat–Shamir transform, the hash function computation corresponds to the verifier's sampling of a random challenge $\mathbf{c}$ after it receives the commitment value $\mathbf{r}$ from the prover. In particular, the verifier always learns the commitment value $\mathbf{r}$ (corresponding to the input of the hash function), so if the identification protocol is "secure", one should always be able to reveal this value without compromising security. But the security of the signature scheme only offers weak guarantees on the security of the underlying identification protocol, as discussed by Abdalla et al. [1].

In usual Fiat–Shamir signatures, this is never an issue because the commitment value can always be publicly derived from the signature (as it is necessary for signature verification). However, things are more subtle in the Fiat–Shamir with aborts paradigm, since the value $\mathbf{r}$ is not normally revealed in executions of the signing algorithm that do not pass the rejection sampling step. In our setting, though, we would like to unmask the value to compute the hash function in all cases, before knowing whether the rejection sampling step will be successful. If we do so, the side-channel attacker can thus learn the pair $(\mathbf{r}, \mathbf{c})$ corresponding to rejected executions as well, and this is not covered by the original security proof, nor does security with this additional leakage look reducible to the original security assumption.

However, it is heuristically a hard problem to distinguish those pairs from uniform (an LWE-like problem with a rather unusual distribution), so one possible approach, which requires no change at all to the algorithm itself, is to redo the security proof with an additional, ad hoc hardness assumption. This is the main approach that we suggest in this paper. Although heuristically safe, it is rather unsatisfactory from a theoretical standpoint, so we additionally propose another approach:[1] compute the hash function not in terms of $\mathbf{r}$ itself, but of $f(\mathbf{r})$ where f is a statistically-hiding commitment scheme whose opening information is added to actual signatures, but not revealed in executions of the algorithm that do not pass the rejection sampling. Using a suitable f, $f(\mathbf{r})$ can be efficiently computed in masked form, and only the result needs to be unmasked. It is then clear that the leakage of $\big(f(\mathbf{r}), \mathbf{c}\big)$ is innocuous, and the modified scheme can be proved entirely with no additional hardness assumption. The downside of this approach is of course that the commitment key increases the size of the public key, the opening information increases the size of signatures, and the masked computation of the commitment itself takes a not insignificant amount of time. For practical purposes, we therefore recommend the heuristic approach.

Security of masking schemes with output-dependent probes. In order to prove the security of our masked implementation we see that we reveal some public value $\mathbf{r}$ or a commitment of it. Consequently, we must adapt the notion of security

[1] We are indebted to Vadim Lyubashevsky for suggesting this approach.

from the threshold probing model to account for public outputs; the idea here is not to state that public outputs do not leak relevant information, but rather that the masked implementation does not leak more information than the one that is released through public outputs. We capture this intuition by letting the simulator depend on the distribution of the public outputs. This extends the usual "non-interference" (NI) security notion to a new, more general notion of "non-interference with public outputs" (NIo).

Security proofs. The overall security guarantee for the masked implementation is established by proving the security of individual gadgets and asserting the security of their combination. For some gadgets, one establishes security in the usual threshold probing model, opening the possibility to resort to automated tools such as maskComp [4] to generate provably secure masked implementations. For other gadgets, the proofs of security are given by exhibiting a simulator, and checking its correctness manually. Finally, the main theorem is deduced from the proof of correctness and security in the threshold probing model with public outputs for the masked implementation, and from a modified proof of security for the GLP scheme.

Organization of the Paper. In Sect. 2, we describe the GLP signature scheme and the security assumption on which its security is based. In Sect. 3, we present the new security notions used in our proofs. Then, in Sect. 4, we describe how to mask the GLP algorithm at any masking order. Finally, in Sect. 5, we describe an implementation of this masking countermeasure, and suggest some possible efficiency improvements.

2 The GLP Signature Scheme

2.1 Parameters and Security

Notations. Throughout this paper, we will use the following notations: n is a power of 2, p is a prime number congruent to 1 modulo $2n$, $\mathcal{R}$ is the polynomial ring modulo x^n+1, $\mathcal{R} = \mathbb{Z}_p[x]/(x^n+1)$. The elements of $\mathcal{R}$ can be represented by polynomials of degree $n-1$ with coefficients in the range $[-\frac{p-1}{2}, \frac{p-1}{2}]$. For an integer k such that $0 < k \leq (p-1)/2$, we denote by $\mathcal{R}_k$ the elements of $\mathcal{R}$ with coefficients in the range $[-k, k]$. We write $\overset{\$}{\leftarrow} S$ for picking uniformly at random in a set S or $\overset{\$}{\leftarrow} \mathcal{D}$ for picking according to some distribution $\mathcal{D}$.

The key generation algorithm for the GLP signature scheme is as follows:

Algorithm 1. GLP key derivation

```
Result: Signing key $sk$, verification key $pk$
$\mathbf{s_1}, \mathbf{s_2} \overset{\$}{\leftarrow} \mathcal{R}_1$ //$\mathbf{s_1}$ and $\mathbf{s_2}$ have coefficients in $\{-1, 0, 1\}$
$\mathbf{a} \overset{\$}{\leftarrow} \mathcal{R}$
$\mathbf{t} \leftarrow \mathbf{a}\mathbf{s_1} + \mathbf{s_2}$
$sk \leftarrow (\mathbf{s_1}, \mathbf{s_2})$
$pk \leftarrow (\mathbf{a}, \mathbf{t})$
```

Given the verification key $pk = (\mathbf{a}, \mathbf{t})$, if an attacker can derive the signing key, he can be used to also solve a $\mathbf{DCK}_{p,n}$ problem defined in [21].

Definition 1. *The* $\mathbf{DCK}_{p,n}$ problem (Decisional Compact Knapsack problem) *is the problem of distinguishing between the uniform distribution over* $\mathcal{R} \times \mathcal{R}$ *and the distribution* $(\mathbf{a}, \mathbf{a}\mathbf{s}_1 + \mathbf{s}_2)$ *with* $\mathbf{s}_1$, $\mathbf{s}_2$ *uniformly random in* $\mathcal{R}_1$.

In the security proof of our variant of the signature scheme, we introduce a new computational problem.

Definition 2. *The* $\mathbf{R\text{-}DCK}_{p,n}$ problem (Rejected-Decisional Compact Knapsack problem) *is the problem of distinguishing between the uniform distribution over* $\mathcal{R} \times \mathcal{R} \times \mathcal{D}_\alpha^n$ *and the distribution* $(\mathbf{a}, \mathbf{a}\mathbf{y}_1 + \mathbf{y}_2, \mathbf{c})$ *where* $(\mathbf{a}, \mathbf{c}, \mathbf{y}_1, \mathbf{y}_2)$ *is uniformly sampled in* $\mathcal{R} \times \mathcal{D}_\alpha^n \times \mathcal{R}_k^2$*, conditioned by the event* $\mathbf{s}_1\mathbf{c} + \mathbf{y}_1 \notin \mathcal{R}_{k-\alpha}$ *or* $\mathbf{s}_2\mathbf{c} + \mathbf{y}_2 \notin \mathcal{R}_{k-\alpha}$.

As shown in the full version of this paper [5], assuming the hardness of $\mathbf{R\text{-}DCK}_{p,n}$ can be avoided entirely by computing the hash value $\mathbf{c}$ not in terms of $\mathbf{r} = \mathbf{a}\mathbf{y}_1 + \mathbf{y}_2$, but of a statistically hiding commitment thereof. This approach shows that masking can be done based on the exact same assumptions as the original scheme, but at some non-negligible cost in efficiency.

To obtain a scheme that more directly follows the original one and to keep the overhead reasonable, we propose to use $\mathbf{R\text{-}DCK}_{p,n}$ as an extra assumption, which we view as a pragmatic compromise. The assumption is admittedly somewhat artificial, but the same can be said of $\mathbf{DCK}_{p,n}$ itself to begin with, and heuristically, $\mathbf{R\text{-}DCK}_{p,n}$ is similar, except that it removes smaller (hence "easier") instances from the distribution: one expects that this makes distinguishing harder, even though one cannot really write down a reduction to formalize that intuition.

2.2 The Signature Scheme

This part describes the signature scheme introduced in [21]. Additional functions like transform and compress introduced in [21] can be used to shorten the size of the signatures. Note however that for masking purposes, we only need to consider the original, non-compressed algorithm of Güneysu et al., which we describe below. Indeed, signature compression does not affect our masking technique at all, because it only involves unmasked parts of the signature generation algorithm (the input of the hash function and the returned signature itself). As a result, although this paper only discusses the non-compressed scheme, we can directly apply our technique to the compressed GLP scheme with no change, and in fact this is what our proof-of-concept implementation in Sect. 5 actually does.

The signature scheme needs a particular cryptographic hash function, $H : \{0,1\}^* \rightarrow \mathcal{D}_\alpha^n$, where $\mathcal{D}_\alpha^n$ is the set of polynomials in $\mathcal{R}$ that have all zero coefficients except for at most $\alpha = 32$ coefficients that are in $\{-1, +1\}$ (or $\alpha = 16$ when using the updated parameters presented in [8]).

Let k be a security parameter. Algorithms 2 and 3 respectively describe the GLP signature and verification. Here is the soundness equation for the verification: $\mathbf{a}\mathbf{z}_1 + \mathbf{z}_2 - \mathbf{t}\mathbf{c} = \mathbf{a}\mathbf{y}_1 + \mathbf{y}_2$.

The parameter k controls the trade-off between the security and the runtime of the scheme. The smaller k gets, the more secure the scheme becomes and the shorter the signatures get but the time to sign will increase. The authors of the implementation of [21] suggest $k = 2^{14}$, $n = 512$ and $p = 8383489$ for ≈ 100 bits of security and $k = 2^{15}$, $n = 1024$ and $p = 16760833$ for > 256 bits of security.

2.3 Security Proof of the r-GLP Variant

As mentioned above, masking the hash function of the GLP signature directly has a prohibitive cost, and it is thus preferable to unmask the input $\mathbf{r} = \mathbf{a}\mathbf{y}_1 + \mathbf{y}_2$ to compute the hash value $\mathbf{c} = H(\mathbf{r}, \mathbf{m})$. Doing so allows a side-channel attacker to learn the pair $(\mathbf{r}, \mathbf{c})$ corresponding to rejected executions as well, and since that additional information is not available to the adversary in the original setting, we need to show that it does not affect the security of the scheme.

This stronger security requirement can be modeled as the unforgeability under chosen message attacks of a modified version of the GLP signature scheme in which the pair $(\mathbf{r}, \mathbf{c})$ is made public when a rejection occurs. We call this modified scheme $\mathbf{r}$-GLP, and describe it as Algorithm 4. The modification means that, in the EUF-CMA security game, the adversary gets access not only to correctly generated GLP signatures, but also to pairs $(\mathbf{r}, \mathbf{c})$ when rejection occurs, which is exactly the setting that arises as a result of unmasking the value $\mathbf{r}$. The following theorem, proved in the full version of this paper [5], states that the modified scheme is indeed secure, at least if we are willing to assume the hardness of the additional $\mathbf{DCK}_{p,n}$ assumption.

Theorem 1. *Let n, p, $\mathcal{R}$ and $\mathcal{D}^n_\alpha$ as defined in Sect. 2.1. Assuming the hardness of the $\boldsymbol{DCK}_{p,n}$ and $\boldsymbol{R\text{-}DCK}_{p,n}$ problems, the signature $\mathbf{r}$-GLP is EUF-CMA secure in the random oracle model.*

Remark 1. As mentioned previously, we can avoid the non-standard assumption $\mathbf{R\text{-}DCK}_{p,n}$ by hashing not $\mathbf{r}$ but $f(\mathbf{r})$ for some statistically hiding commitment f (which can itself be constructed under $\mathbf{DCK}_{p,n}$, or standard lattice assumptions). See the full version of this paper for details [5]. The downside of that

Algorithm 2. GLP signature

```
Data: $\mathbf{m}$, $pk$, $sk$
Result: Signature $\sigma$
$\mathbf{y}_1, \mathbf{y}_2 \xleftarrow{\$} \mathcal{R}_k$
$\mathbf{c} \leftarrow H(\mathbf{r} = \mathbf{a}\mathbf{y_1} + \mathbf{y_2}, \mathbf{m})$
$\mathbf{z}_1 \leftarrow \mathbf{s}_1\mathbf{c} + \mathbf{y}_1$
$\mathbf{z}_2 \leftarrow \mathbf{s}_2\mathbf{c} + \mathbf{y}_2$
if $\mathbf{z}_1$ or $\mathbf{z}_2 \notin \mathcal{R}_{k-\alpha}$ then
    restart
end
return $\sigma = (\mathbf{z}_1, \mathbf{z}_2, \mathbf{c})$
```

Algorithm 3. GLP verification

```
Data: $\mathbf{m}$, $\sigma$, $pk$
if $\mathbf{z}_1$, $\mathbf{z}_2 \in \mathcal{R}_{k-\alpha}$ and
$\mathbf{c} = H(\mathbf{a}\mathbf{z}_1 + \mathbf{z}_2 - \mathbf{t}\mathbf{c}, \mathbf{m})$ then
    accept
else
    reject
end
```

Algorithm 4. Tweaked signature with public $\mathbf{r}$

```
Data: $\mathbf{m}$, $pk = (\mathbf{a}, \mathbf{t})$, $sk = (\mathbf{s_1}, \mathbf{s_2})$
Result: Signature $\sigma$
$\mathbf{y_1} \xleftarrow{\$} \mathcal{R}_k$
$\mathbf{y_2} \xleftarrow{\$} \mathcal{R}_k$
$\mathbf{r} \leftarrow \mathbf{a}\mathbf{y_1} + \mathbf{y_2}$
$\mathbf{c} \leftarrow H(\mathbf{r}, \mathbf{m})$
$\mathbf{z_1} \leftarrow \mathbf{s_1}\mathbf{c} + \mathbf{y_1}$
$\mathbf{z_2} \leftarrow \mathbf{s_2}\mathbf{c} + \mathbf{y_2}$
if $\mathbf{z_1}$ or $\mathbf{z_2} \notin \mathcal{R}_{k-\alpha}$ then
    $(\mathbf{z_1}, \mathbf{z_2}) \leftarrow (\perp, \perp)$
end
return $\sigma = (\mathbf{z_1}, \mathbf{z_2}, \mathbf{c}, \mathbf{r})$
```

approach is that it has a non negligible overhead in terms of key size, signature size, and to a lesser extent signature generation time.

3 Threshold Probing Model with Public Outputs

In this section, we briefly review the definition of the threshold probing model, and introduce an extension to accommodate public outputs.

3.1 Threshold Probing Model

The threshold probing model introduced by Ishai, Sahai and Wagner considers implementations that operate over shared values [22].

Definition 3. *Let d be a masking order. A shared value is a $(d+1)$-tuple of values, typically integers or Booleans.*

A (u, v)-gadget is a probabilistic algorithm that takes as inputs u shared values, and returns distributions over v-tuples of shared values. (u, v)-gadgets are typically used to implement functions that take u inputs and produce v outputs.

Gadgets are typically written in pseudo-code, and induce a mapping from u-tuples of shared values (or equivalently $u(d+1)$-tuples of values) to a distribution over v-tuples of values, where the output tuple represents the joint distribution of the output shared values as well as all intermediate values computed during the execution of the gadget.

We now turn to the definition of probing security. Informally, an implementation is d-probing secure if and only if an adversary that can observe at most d intermediate values cannot recover information on secret inputs.

Definition 4. d-non-interference (d-NI): *A gadget is d-non-interfering if and only if every set of at most d intermediate variables can be perfectly simulated with at most d shares of each input.*

Definition 5. d-strong-non-interference (d-SNI): *A gadget is d-strongly non interfering if and only if every set of size $d_0 \leq d$ containing d_1 intermediate variables and $d_2 = d_0 - d_1$ returned values can be perfectly simulated with at most d_1 shares of each input.*

This notion of security is formulated in a simulation-based style. It is however possible to provide an equivalent notion as an information flow property in the style of programming language security and recent work on formal methods for proving security of masked implementations.

The maskComp tool. For certain composition proofs, we will use the maskComp tool from Barthe et al. [4]. It uses a type-based information flow analysis with cardinality constraints and ensures that the composition of gadgets is d-NI secure at arbitrary orders, by inserting refresh gadgets when required.

3.2 Threshold Probing Model with Public Outputs

The security analysis of our masked implementation of GLP requires an adaptation of the standard notion of security in the threshold probing model. Specifically, our implementation does not attempt to mask the computation of $H(\mathbf{r}, \mathbf{m})$ at line 2 of Algorithm 2; instead, it recovers $\mathbf{r}$ from its shares and then computes $H(\mathbf{r}, \mathbf{m})$. This optimization is important for the efficiency of the masked algorithm, in particular because it is not immediately clear whether one can mask the hash function H efficiently—note that this kind of optimization is also reminiscent of the method used to achieve efficient sorting algorithms in multi-party computations.

From a security perspective, recombining $\mathbf{r}$ in the algorithm is equivalent to making $\mathbf{r}$ a public output. In contrast with "return values", we will refer to "outputs" as values broadcast on a public channel during the execution of the masked algorithm. The side-channel attacker can therefore use outputs in attacks. Since the usual notions of NI and SNI security do not account for outputs in that sense, we need to extend those notions of security to support algorithms that provide such outputs. The idea here is not to state that public outputs do not leak relevant information, but rather that the masked implementation does not leak more information than the one that is released through public outputs. We capture this intuition by letting the simulator depend on the distribution of the public outputs.

Definition 6. *A gadget with public outputs is a gadget together with a distinguished subset of intermediate variables whose values are broadcast during execution.*

We now turn to the definition of probing security for gadgets with public outputs.

Definition 7. d-non-interference for gadgets with public outputs (d-NIo): *A gadget with public outputs X is d-NIo if and only if every set of at most d*

intermediate variables can be perfectly simulated with the public outputs and at most d shares of each input.

Again, it is possible to provide an equivalent notion as an information flow property in the style of programming language security.

Note that the use of public outputs induces a weaker notion of security.

Lemma 1. *Let G be a d-NI-gadget. Then G is d-NIo secure for every subset X of intermediate variables.*

Informally, the lemma states that a gadget that does not leak any information also does not leak more information than the one revealed by a subset of its intermediate variables. The lemma is useful to resort to automated tools for proving NI security of some gadgets used in the masked implementations of GLP. In particular, we will use the maskComp tool.

Since d-NIo security is weaker than d-NI security, we must justify that it delivers the required security guarantee. This is achieved by combining the proofs of security for the modified version of GLP with public outputs, and the proofs of correctness and security for the masked implementations of GLP.

4 Masked Algorithm

In this section, the whole GLP scheme is turned into a functionally equivalent scheme secure in the d-probing model with public outputs. Note that it suffices to mask the key derivation in the d-probing model and the signature in the d-probing model with public output $\boldsymbol{r}$, since the verification step does not manipulate sensitive data.

Remark 2. The masked version of GLP scheme with commitment has also been turned into a functionally equivalent scheme proved secure in the d-probing model with public output $\boldsymbol{r}$. Its masked version is a little more complex, it is detailed in the full version of this paper [5].

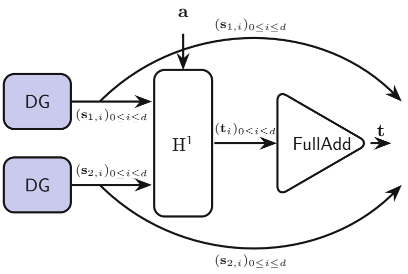
a
$(\mathbf{s}_{1,i})_{0\leq i\leq d}$
DG
$(\mathbf{s}_{1,i})_{0\leq i\leq d}$
H^1
$(\mathbf{t}_i)_{0\leq i\leq d}$
FullAdd
t
$(\mathbf{s}_{2,i})_{0\leq i\leq d}$
DG
$(\mathbf{s}_{2,i})_{0\leq i\leq d}$

Fig. 1. Composition of mKD (The blue gadgets will be proved d-NIo, the white ones will be proved d-NI) (Color figure online)

4.1 Overall Structure

For simplicity, we will show the masking on a single iteration version of the signature. The masking can be generalized by calling the masked signature again if it fails.

To ensure protection against d-th order attacks, we suggest a masking countermeasure with $d+1$ shares for the following sensitive data: $\mathbf{y}_1$, $\mathbf{y}_2$, $\mathbf{s}_1$ and $\mathbf{s}_2$. All the public variables are $(\mathbf{a}, \mathbf{t})$ (i.e., the public key), $\mathbf{m}$ (i.e., the message), $RejSp$ (i.e., the bit corresponding to the success of the rejection sampling), $(\mathbf{z}_1, \mathbf{z}_2, \mathbf{c})$ (i.e., the signature). As mentioned before, because of the need of $\boldsymbol{r}$ recombination, even if $\mathbf{r}$ is an intermediate value, it is considered as a public output.

Most operations carried out in the GLP signing algorithm are arithmetic operations modulo p, so we would like to use arithmetic masking. It means for example that $\mathbf{y}_1$ will be replaced by $\mathbf{y}_{1,0}, ... \mathbf{y}_{1,d} \in \mathcal{R}$ such that

$$\mathbf{y}_1 = \mathbf{y}_{1,0} + ... + \mathbf{y}_{1,d} \mod p.$$

The issue is that at some points of the algorithm, we need to perform operations that are better expressed using Boolean masking. Those parts will be extracted from both the key derivation and the signature to be protected individually and then securely composed. The different new blocks to achieve protection against d-th order attacks are depicted hereafter and represented in Figs. 1 and 2:

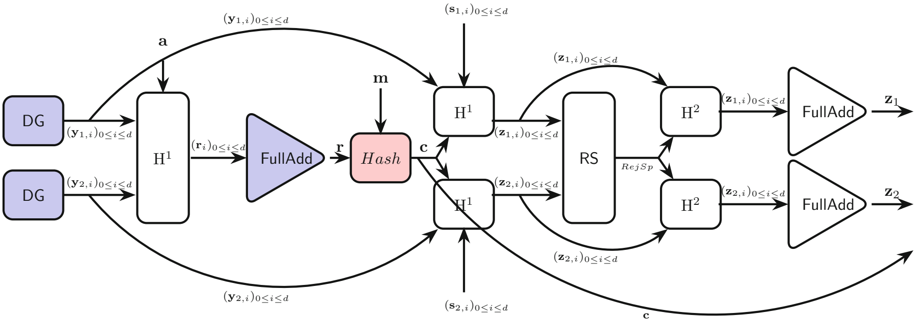
$(\mathbf{y}_{1,i})_{0\le i\le d}$
$\mathbf{a}$
$(\mathbf{s}_{1,i})_{0\le i\le d}$
$(\mathbf{z}_{1,i})_{0\le i\le d}$
$\mathbf{m}$
DG
DG
$(\mathbf{y}_{1,i})_{0\le i\le d}$
$(\mathbf{y}_{2,i})_{0\le i\le d}$
H^1
$(\mathbf{r}_i)_{0\le i\le d}$
FullAdd
$\mathbf{r}$
$Hash$
$\mathbf{c}$
H^1
H^1
$(\mathbf{z}_{1,i})_{0\le i\le d}$
$(\mathbf{z}_{2,i})_{0\le i\le d}$
RS
$RejSp$
H^2
H^2
$(\mathbf{z}_{1,i})_{0\le i\le d}$
$(\mathbf{z}_{2,i})_{0\le i\le d}$
FullAdd
FullAdd
$\mathbf{z}_1$
$\mathbf{z}_2$
$(\mathbf{z}_{2,i})_{0\le i\le d}$
$(\mathbf{y}_{2,i})_{0\le i\le d}$
$(\mathbf{s}_{2,i})_{0\le i\le d}$
$\mathbf{c}$

Fig. 2. Composition of mSign (The blue gadgets will be proved d-NIo, the white ones will be proved d-NI and the red one won't be protected) (Color figure online)

- Generation of the shared data (DG), masked version of line 1 in Algorithm 2 and line 1 in Algorithm 1, is a function to generate shares of $\mathbf{y}_1$, $\mathbf{y}_2$, $\mathbf{s}_1$ and $\mathbf{s}_2$. It will be described in Algorithm 7, decomposed and proved d-NIo secure by decomposition.
- Rejection Sampling (RS), masked version of line 5 in Algorithm 2, is a test to determine if $\mathbf{z}_1$ and $\mathbf{z}_2$ belong to the set $\mathcal{R}_{k-\alpha}$. It will be detailed in Algorithm 15 and proved d-NI secure by decomposition.

- Refresh and unmask (FullAdd) is a function that unmasks securely a variable by adding together its shares modulo p without leaking the partial sums. It will be described in Algorithm 16 and proved d-NIo secure and d-NI secure when used at the end.
- H^1 and H^2 are the elementary parts, masked versions of line 2, 3–4 and then 5–6 in Algorithm 2. H^1 is also the masked version of the instruction called in line 3 of the key derivation algorithm (Algorithm 1). They are made of arithmetic computations. They are depicted in Algorithms 17 and 18. They will be proved d-NI secure.
- Hash function, line 2 in Algorithm 2. As mentioned before, is left unmasked because it manipulates only public data.

Algorithm 6 shows a high level picture of mSign with all these blocks and Algorithm 5 shows mKD.

The proofs of dNI or d-NIo security will be given in the following subsection. Then, the composition will be proved in Sect. 4.3 to achieve global security in the d-probing model with public outputs. This yields the d-NIo security of the masked signature and masked key generation algorithms in Theorems 2 and 3, respectively. By combining these results with the refined analysis of the GLP signature in Theorem 1, one obtains the desired security guarantee, as discussed in Sect. 3.

4.2 Masked Gadgets

In this section each gadget will be described and proved d-NI or d-NIo secure. The difficulty is located in the gadgets containing Boolean/arithmetic conversions. In those gadgets (DG and RS) a detailed motivation and description has been made.

Data generation (DG). In the unmasked GLP signing algorithm, the coefficients of the "commitment" polynomials $\mathbf{y}_1$, $\mathbf{y}_2$ are sampled uniformly and independently from an integer interval of the form $[-k, k]$. In order to mask the

Algorithm 5. mKD

```
Result: Signing key $sk$, verification key $pk$
$(\mathbf{s}_{1,i})_{0\leq i\leq d} \leftarrow \mathsf{DG}(1, d)$
$(\mathbf{s}_{2,i})_{0\leq i\leq d} \leftarrow \mathsf{DG}(1, d)$
$\mathbf{a} \overset{\$}{\leftarrow} \mathcal{R}$
$(\mathbf{t}_i)_{0\leq i\leq d} \leftarrow \mathrm{H}^1(\mathbf{a}, (\mathbf{s}_{1,i})_{0\leq i\leq d}, (\mathbf{s}_{2,i})_{0\leq i\leq d})$
$\mathbf{t} \leftarrow \mathsf{FullAdd}((\mathbf{t}_i)_{0\leq i\leq d})$
$sk \leftarrow ((\mathbf{s}_{1,i})_{0\leq i\leq d}, (\mathbf{s}_{2,i})_{0\leq i\leq d})$
$pk \leftarrow (\mathbf{a}, \mathbf{t})$
return as public key $(\mathbf{a}, \mathbf{t})$
return as secret key $((\mathbf{s}_{1,i})_{0\leq i\leq d}, (\mathbf{s}_{2,i})_{0\leq i\leq d})$
```

Algorithm 6. mSign

```
Data: $m$, $pk = (\mathbf{a}, \mathbf{t})$, $sk = ((\mathbf{s}_{1,i})_{0 \le i \le d}, (\mathbf{s}_{2,i})_{0 \le i \le d})$
Result: Signature $\sigma$
$(\mathbf{y}_{1,i})_{0 \le i \le d} \leftarrow \mathsf{DG}(k, d)$
$(\mathbf{y}_{2,i})_{0 \le i \le d} \leftarrow \mathsf{DG}(k, d)$
$(\mathbf{r}_i)_{0 \le i \le d} \leftarrow \mathrm{H}^1(\mathbf{a}, (\mathbf{y}_{1,i})_{0 \le i \le d}, (\mathbf{y}_{2,i})_{0 \le i \le d})$
$\mathbf{r} \leftarrow \mathsf{FullAdd}((\mathbf{r}_i)_{0 \le i \le d})$
$\mathbf{c} \leftarrow hash(\mathbf{r}, \mathbf{m})$
$(\mathbf{z}_{1,i})_{0 \le i \le d} \leftarrow \mathrm{H}^1(\mathbf{c},\ (\mathbf{s}_{1,i})_{0 \le i \le d},\ (\mathbf{y}_{1,i})_{0 \le i \le d})$
$(\mathbf{z}_{2,i})_{0 \le i \le d} \leftarrow \mathrm{H}^1(\mathbf{c},\ (\boldsymbol{ts}_{2,i})_{0 \le i \le d},\ (\mathbf{y}_{2,i})_{0 \le i \le d})$
$RejSp \leftarrow \mathsf{RS}((\mathbf{z}_{1,i})_{0 \le i \le d}, (\mathbf{z}_{2,i})_{0 \le i \le d}, k - \alpha)$
$(\mathbf{z}_{1,i})_{0 \le i \le d} \leftarrow \mathrm{H}^2(RejSp, (\mathbf{z}_{1,i})_{0 \le i \le d})$
$(\mathbf{z}_{2,i})_{0 \le i \le d} \leftarrow \mathrm{H}^2(RejSp, (\mathbf{z}_{2,i})_{0 \le i \le d})$
$\mathbf{z}_1 \leftarrow \mathsf{FullAdd}((\mathbf{z}_{1,i})_{0 \le i \le d})$
$\mathbf{z}_2 \leftarrow \mathsf{FullAdd}((\mathbf{z}_{2,i})_{0 \le i \le d})$
return $\sigma = (\mathbf{z}_1, \mathbf{z}_2, \mathbf{c})$
```

Algorithm 7. Data Generation (DG)

```
Data: $k$ and $d$
Result: A uniformly random $\mathbf{y}$ integer in $\mathcal{R}_k$ in arithmetic masked form
        $(\mathbf{y}_i)_{0 \le i \le d}$.
$(\mathbf{y}_i)_{0 \le i \le d} \leftarrow \{0\}^d$
for $j = 1$ to $n$ do
    $(a_i)_{0 \le i \le d} \leftarrow \mathsf{RG}(k, d)$
    $(\mathbf{y}_i)_{0 \le i \le d} \leftarrow (\mathbf{y}_i + a_i x^j)_{0 \le i \le d}$
end
return $(\mathbf{y}_i)_{0 \le i \le d}$
```

signing algorithm, one would like to obtain those values in masked form, using order-d arithmetic masking modulo p. Note that since all of these coefficients are completely independent, the problem reduces to obtaining an order-d mod-p arithmetic masking of a single random integer in $[-k, k]$.

Accordingly, we will first create an algorithm called Random Generation (RG) which generates an order-d mod-p arithmetic masking of a single random integer in $[-k, k]$. Next, we will use RG in an algorithm called Data Generation (DG) which generates a sharing of a value in $\mathcal{R}_k$. DG is calling RG n times and is described in Algorithm 7. RG is described hereafter and will be given in Algorithm 14.

Let us now build RG. Carrying out this masked random sampling in arithmetic form directly and securely seems difficult. On the other hand, it is relatively easy to generate a Boolean masking of such a uniformly random value. We can then convert that Boolean masking to an arithmetic masking using Coron et al.'s higher-order Boolean-to-arithmetic masking conversion technique [13]. The technique has to be modified slightly to account for the fact that the modulus p of the arithmetic masking is not a power of two, but the overall structure of the algorithm remains the same. To obtain a better complexity, we also use the Kogge–Stone adder based addition circuit already considered in [12].

A more precise description of our approach is as follows. Let $K = 2k+1$, and w_0 be the smallest integer such that $2^{w_0} > K$. Denote also by w the bit size of the Boolean masking we are going to use; we should have $w > w_0 + 1$ and $2^w > 2p$. For GLP masking, a natural choice, particularly on a 32-bit architecture, would be $w = 32$.

Now the first step of the algorithm is to generate w_0-bit values $(x_i^0)_{0 \le i \le d}$ uniformly and independently at random, and apply a multiplication-based share refreshing algorithm $\mathsf{Refresh}$, as given in Algorithm 8, to obtain a fresh w-bit Boolean masking $(x_i)_{0 \le i \le d}$ of the same value x:

$$x = \bigoplus_{i=0}^{d} x_i^0 = \bigoplus_{i=0}^{d} x_i.$$

Note that x is then a uniform integer in $[0, 2^{w_0} - 1]$.

Algorithm 8. Multiplication-based refresh algorithm for Boolean masking ($\mathsf{Refresh}$)

```
Data: A Boolean masking $(x_i)_{0 \le i \le d}$ of some value $x$; the bit size $w$ of the
      returned masks
Result: An independent Boolean masking $(x'_i)_{0 \le i \le d}$ of $x$
$(x'_i)_{0 \le i \le d} \leftarrow (x_i)_{0 \le i \le d}$
for $i = 0$ to $d$ do
    for $j = i + 1$ to $d$ do
        pick a uniformly random $w$-bit value $r$
        $x'_i \leftarrow x'_i \oplus r$
        $x'_j \leftarrow x'_j \oplus r$
    end
end
return $(x'_i)_{0 \le i \le d}$
```

We then carry out a rejection sampling on x: if $x \ge K$, we restart the algorithm. If this step is passed successfully, x will thus be uniformly distributed in $[0, K-1] = [0, 2k]$. Of course, the test has to be carried out securely at order d. This can be done as follows: compute a random w-bit Boolean masking $(k_i)_{0 \le i \le d}$ of the constant $(-K)$ (the two's complement of K over w bits; equivalently, one can use $2^w - K$), and carry out the d-order secure addition $\mathsf{SecAdd}\big((x_i)_{0 \le i \le d}, (k_i)_{0 \le i \le d}\big)$, given in Algorithm 9 (where $\mathsf{Refresh}$ denotes the d-SNI multiplication-based refresh as proven in [4] and recalled in Algorithm 8). The result is a Boolean masking $(\delta_i)_{0 \le i \le d}$ of the difference $\delta = x - K$ in two's complement form. In particular, the most significant bit b of δ is 0 if and only if $x \ge K$. Since computing the most significant bit is an $\mathbb{F}_2$-linear operation,

Algorithm 9. Integer addition of Boolean maskings (SecAdd), as generated by the $\mathsf{maskComp}$ tool from the Kogge–Stone adder of [12]

```
Data: Boolean maskings $(x_i)_{0\le i\le d}$, $(y_i)_{0\le i\le d}$ of integers $x$, $y$; the bit size $w$ of
      the masks
Result: A Boolean masking $(z_i)_{0\le i\le d}$ of $x+y$
$(p_i)_{0\le i\le d} \leftarrow (x_i \oplus y_i)_{0\le i\le d}$
$(g_i)_{0\le i\le d} \leftarrow \mathsf{SecAnd}\big((x_i)_{0\le i\le d}, (y_i)_{0\le i\le d}, w\big)$
for $j = 1$ to $W := \lceil \log_2(w-1) \rceil - 1$ do
    $\mathrm{pow} \leftarrow 2^{j-1}$
    $(a_i)_{0\le i\le d} \leftarrow (g_i \ll \mathrm{pow})_{0\le i\le d}$
    $(a_i)_{0\le i\le d} \leftarrow \mathsf{SecAnd}\big((a_i)_{0\le i\le d}, (p_i)_{0\le i\le d}, w\big)$
    $(g_i)_{0\le i\le d} \leftarrow (g_i \oplus a_i)_{0\le i\le d}$
    $(a'_i)_{0\le i\le d} \leftarrow (p_i \ll \mathrm{pow})_{0\le i\le d}$
    $(a'_i)_{0\le i\le d} \leftarrow \mathsf{Refresh}\big((a_i)_{0\le i\le d}, w\big)$
    $(p_i)_{0\le i\le d} \leftarrow \mathsf{SecAnd}\big((p_i)_{0\le i\le d}, (a'_i)_{0\le i\le d}, w\big)$
end
$(a_i)_{0\le i\le d} \leftarrow (g_i \ll 2^W)_{0\le i\le d}$
$(a_i)_{0\le i\le d} \leftarrow \mathsf{SecAnd}\big((a_i)_{0\le i\le d}, (p_i)_{0\le i\le d}, w\big)$
$(g_i)_{0\le i\le d} \leftarrow (g_i \oplus a_i)_{0\le i\le d}$
$(z_i)_{0\le i\le d} \leftarrow \big(x_i \oplus y_i \oplus (g_i \ll 1)\big)_{0\le i\le d}$
return $(z_i)_{0\le i\le d}$
```

we can carry it out componentwise to obtain a masking $(b_i)_{0\le i\le d}$ of b with $b_i = \delta_i \gg (w-1)$. The resulting bit b is non-sensitive, so we can unmask it to check whether to carry out the rejection sampling.

After carrying out these steps, we have obtained a Boolean masking of a uniformly random integer in $[0, 2k]$. What we want is a *mod-p arithmetic* masking of a uniformly random integer in the interval $[-k, k]$, which is of the same length as $[0, 2k]$. If we can convert the Boolean masking to an arithmetic masking, it then suffices to subtract k from one of the shares and we obtain the desired result. To carry out the Boolean-to-arithmetic conversion itself, we essentially follow the approach of [13, Sect. 5], with a few changes to account for the fact that p is not a power of two.

The main change is that we need an algorithm for the secure addition modulo p of two values y, z in Boolean masked form $(y_i)_{0\le i\le d}$, $(z_i)_{0\le i\le d}$ (assuming that $y, z \in [0, p)$). Such an algorithm $\mathsf{SecAddModp}$ is easy to construct from SecAdd (see Algorithm 10 with SecAnd the d-order secure bitwise AND operation from [22,30] and recalled in Algorithm 11) and the comparison trick described earlier. More precisely, the approach is to first compute $(s_i)_{0\le i\le d} = \mathsf{SecAdd}\big((y_i)_{0\le i\le d}, (z_i)_{0\le i\le d}\big)$, which is a Boolean sharing of the sum $s = y + z$ without modular reduction, and then $(s'_i)_{0\le i\le d} = \mathsf{SecAdd}\big((s_i)_{0\le i\le d}, (p_i)_{0\le i\le d}\big)$ for a Boolean masking $(p_i)_{0\le i\le d}$ of the value $-p$ in two's complement form (or equivalently $2^w - p$). The result is a masking of $s' = s - p$ in two's complement form. In particular, we have $s \ge p$ if and only if the most significant bit b of s' is 0. Denote by r the desired modular addition $y + z \bmod p$. We thus have:

Algorithm 10. Mod-p addition of Boolean maskings ($\mathsf{SecAddModp}$)

```
Data: Boolean maskings $(x_i)_{0\le i\le d}$, $(y_i)_{0\le i\le d}$ of integers $x$, $y$; the bit size $w$ of
      the masks (with $2^w > 2p$)
Result: A Boolean masking $(z_i)_{0\le i\le d}$ of $x + y \bmod p$
$(p_i)_{0\le i\le d} \leftarrow \big(2^w - p, 0, \ldots, 0\big)$
$(s_i)_{0\le i\le d} \leftarrow \mathsf{SecAdd}\big((x_i)_{0\le i\le d}, (y_i)_{0\le i\le d}, w\big)$
$(s'_i)_{0\le i\le d} \leftarrow \mathsf{SecAdd}\big((s_i)_{0\le i\le d}, (p_i)_{0\le i\le d}, w\big)$
$(b_i)_{0\le i\le d} \leftarrow \big(s'_i \gg (w-1)\big)_{0\le i\le d}$
$(c_i)_{0\le i\le d} \leftarrow \mathsf{Refresh}\big((b_i)_{0\le i\le d}, w\big)$
$(z_i)_{0\le i\le d} \leftarrow \mathsf{SecAnd}\big((s_i)_{0\le i\le d}, (\widetilde{c_i})_{0\le i\le d}, w\big)$
$(c_i)_{0\le i\le d} \leftarrow \mathsf{Refresh}\big((b_i)_{0\le i\le d}, w\big)$
$(z_i)_{0\le i\le d} \leftarrow (z_i)_{0\le i\le d} \oplus \mathsf{SecAnd}\big((s'_i)_{0\le i\le d}, (\widetilde{\neg c_i})_{0\le i\le d}, w\big)$
return $(z_i)_{0\le i\le d}$
```

Algorithm 11. Bitwise AND of Boolean maskings (SecAnd) from [22,30]

```
Data: Boolean maskings $(x_i)_{0\le i\le d}$, $(y_i)_{0\le i\le d}$ of integers $x$, $y$; the bit size $w$ of
      the masks
Result: A Boolean masking $(r_i)_{0\le i\le d}$ of $x \wedge y$
$(r_i)_{0\le i\le d} \leftarrow (x_i \wedge y_i)_{0\le i\le d}$
for $i = 0$ to $d$ do
    for $j = i + 1$ to $d$ do
        pick a uniformly random $w$-bit value $z_{ij}$
        $z_{ji} \leftarrow (x_i \wedge y_j) \oplus z_{ij}$
        $z_{ji} \leftarrow z_{ji} \oplus (x_j \wedge y_i)$
        $r_i \leftarrow r_i \oplus z_{ij}$
        $r_j \leftarrow r_j \oplus z_{ji}$
    end
end
return $(r_i)_{0\le i\le d}$
```

$$r = \begin{cases} s & \text{if } b = 1; \\ s' & \text{if } b = 0. \end{cases}$$

As a result, we can obtain the masking of r in a secure way as:

$$(r_i)_{0\le i\le d} = \mathsf{SecAnd}\big((s_i)_{0\le i\le d}, (\widetilde{b_i})_{0\le i\le d}\big) \oplus \mathsf{SecAnd}\big((s'_i)_{0\le i\le d}, (\widetilde{\neg b_i})_{0\le i\le d}\big),$$

where we denote by $\widetilde{b}$ the extension of the bit b to the entire w-bit register (this is again an $\mathbb{F}_2$-linear operation that can be computed componentwise). This concludes the description of $\mathsf{SecAddModp}$.

Using $\mathsf{SecAddModp}$ instead of SecAdd in the algorithms of [13, Sect. 4], we also immediately obtain an algorithm $\mathsf{SecArithBoolModp}$ for converting a mod-p arithmetic masking $a = \sum_{i=0}^{d} a_i \bmod p$ of a value $a \in [0, p)$ into a Boolean masking $a = \bigoplus_{i=0}^{d} a'_i$ of the same value. The naive way of doing so (see Algorithm 12),

Algorithm 12. Secure conversion from mod-p arithmetic masking to Boolean masking (SecArithBoolModp); this is the simple version (cubic in the masking order)

```
Data: Arithmetic masking $(a_i)_{0\le i\le d}$ modulo $p$ of an integer $a$; the bit size $w$ of
      the returned masks (with $2^w > 2p$)
Result: A Boolean masking $(a'_i)_{0\le i\le d}$ of $a$
$(a'_i)_{0\le i\le d} \leftarrow (0,\dots,0)$
for $j = 0$ to $d$ do
    $(b_i)_{0\le i\le d} \leftarrow (a_j, 0, \dots, 0)$
    $(b_i)_{0\le i\le d} \leftarrow \mathsf{Refresh}((b_i)_{0\le i\le d}, w)$
    $(a'_i)_{0\le i\le d} \leftarrow \mathsf{SecAddModp}((a'_i)_{0\le i\le d}, (b_i)_{0\le i\le d}, w)$
end
return $(a'_i)_{0\le i\le d}$
```

Algorithm 13. Refresh-and-unmask algorithm for Boolean masking (FullXor) from [13]

```
Data: A Boolean masking $(x_i)_{0\le i\le d}$ of some value $x$; the bit size $w$ of the masks
Result: The value $x$
$(x'_i)_{0\le i\le d} \leftarrow \mathsf{FullRefresh}((x_i)_{0\le i\le d}, w)$
$x \leftarrow x'_0$ for $i = 1$ to $d$ do
    $x \leftarrow x \oplus x'_i$
end
return $x$
```

which is the counterpart of [13, Sect. 4.1], is to simply construct a Boolean masking of each of the shares a_i, and to iteratively apply SecAddModp to those masked values. This is simple and secure, but as noted by Coron et al., this approach has cubic complexity in the masking order d (because SecAdd and hence SecAddModp are quadratic). A more advanced, recursive approach allows to obtain quadratic complexity for the whole conversion: this is described in [13, Sect. 4.2], and directly applies to our setting.

With both algorithms SecAddModp and SecArithBoolModp in hand, we can easily complete the description of our commitment generation algorithm by mimicking [13, Algorithm 6]. To convert the Boolean masking $(x_i)_{0\le i\le d}$ of x to a mod-p arithmetic masking, we first generate random integer shares $a_i \in [0,p)$, $1 \le i \le d$, uniformly at random. We then define $a'_i = -a_i \bmod p = p - a_i$ for $1 \le i \le d$ and $a'_0 = 0$. The tuple $(a'_i)_{0\le i\le d}$ is thus a mod-p arithmetic masking of the sum $a' = -\sum_{1\le i\le d} a_i \bmod p$. Using SecArithBoolModp, we convert this arithmetic masking to a Boolean masking $(y_i)_{0\le i\le d}$, so that $\bigoplus_{i=0}^{d} y_i = a'$. Now, let $(z_i)_{0\le i\le d} = \mathsf{SecAddModp}((x_i)_{0\le i\le d}, (y_i)_{0\le i\le d})$; this is a Boolean masking of:

$$z = (x + a') \bmod p = \left(x - \sum_{i=1}^{d} a_i\right) \bmod p.$$

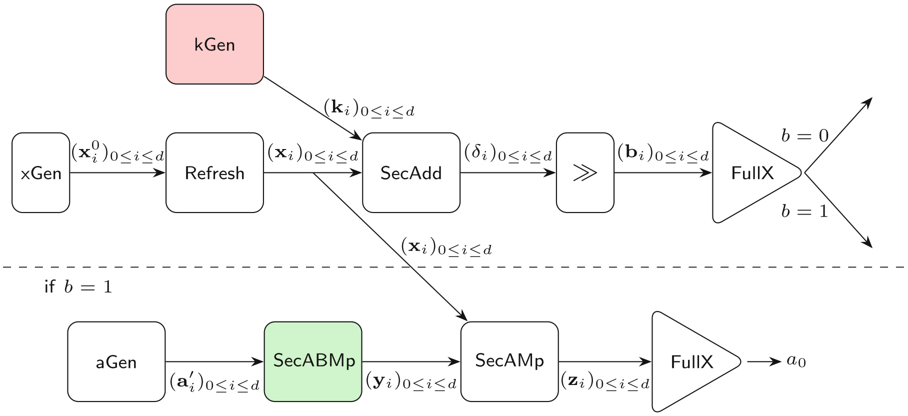
kGen
$(\mathbf{k}_i)_{0\le i\le d}$
xGen
$(\mathbf{x}_i^0)_{0\le i\le d}$
Refresh
$(\mathbf{x}_i)_{0\le i\le d}$
SecAdd
$(\delta_i)_{0\le i\le d}$
$\gg$
$(\mathbf{b}_i)_{0\le i\le d}$
FullX
$b=0$
$b=1$
$(\mathbf{x}_i)_{0\le i\le d}$
if $b=1$
aGen
$(\mathbf{a}_i')_{0\le i\le d}$
SecABMp
$(\mathbf{y}_i)_{0\le i\le d}$
SecAMp
$(\mathbf{z}_i)_{0\le i\le d}$
FullX
a_0

Fig. 3. Randomness Generation RG (The green (resp. white, red) gadgets will be proved *d*-SNI (resp. *d*-NI, unmasked)) (Color figure online)

Algorithm 14. Randomness generation (RG)

```
Data: $k$ and $d$
Result: A uniformly random $a$ integer in $[-k,k]$ in mod-$p$ arithmetic masked
        form $(a_i)_{0\le i\le d}$.
generate uniformly random $w_0$-bit values $(x_i^0)_{0\le i\le d}$
$(x_i)_{0\le i\le d} \leftarrow \mathsf{Refresh}\big((x_i^0)_{0\le i\le d}\big)$
initialize $(k_i)_{0\le i\le d}$ to a $w$-bit Boolean sharing of the two's complement value
  $-K = -2k-1$
$(\delta_i)_{0\le i\le d} \leftarrow \mathsf{SecAdd}\big((x_i)_{0\le i\le d}, (k_i)_{0\le i\le d}\big)$
$(b_i)_{0\le i\le d} \leftarrow (\delta_i)_{0\le i\le d} \gg (w-1)$
$b \leftarrow \mathsf{FullXor}\big((b_i)_{0\le i\le d}\big)$
output $b$
if $b=0$ then
    restart
end
generate uniform integers $(a_i)_{1\le i\le d}$ in $[0,p)$
$a_i' \leftarrow -a_i \bmod p$ for $i=1,\ldots,d$
$a_0' \leftarrow 0$
$(y_i)_{0\le i\le d} \leftarrow \mathsf{SecArithBoolModp}\big((a_i')_{0\le i\le d}\big)$
$(z_i)_{0\le i\le d} \leftarrow \mathsf{SecAddModp}\big((x_i)_{0\le i\le d}, (y_i)_{0\le i\le d}\big)$
$a_0 \leftarrow \mathsf{FullXor}\big((z_i)_{0\le i\le d}\big)$
return $(a_i)_{0\le i\le d}$
```

We then securely unmask this value using Coron et al. FullXor procedure, recalled in Algorithm 13, and set $a_0 = z - k \bmod p$. Then, we have:

$$\sum_{i=0}^{d} a_i \bmod p = z-k+\sum_{i=1}^{d} a_i \bmod p = x-k-\sum_{i=1}^{d} a_i+\sum_{i=1}^{d} a_i \bmod p = x-k \bmod p.$$

Thus, $(a_i)_{0 \leq i \leq d}$ is a correct mod-p arithmetic masking of a uniformly random value in $[-k, k]$ as required. The whole procedure is summarized in Algorithm 14 and described in Fig. 3 where xGen stands for the generation of x^0's shares, Refresh for the multiplication-based refreshing from [4,22], kGen for the generation of k's shares, $\gg$ for the right shift of δ's shares, FullX for FullXor, aGen for the generation of a's shares, SecABM for SecArithBoolModp, and SecAMp for SecAddModp.

The success probability of the rejection sampling step (the masked comparison to K) is $K/2^{w_0}$, and hence is at least $1/2$ by definition of w_0. Therefore, the expected number of runs required to complete is at most 2 (and in fact, a judicious choice of k, such as one less than a power of two, can make the success probability very close to 1). Since all the algorithms we rely on are at most in the masking order and (when using the masked Kogge–Stone adder of [12]) logarithmic in the size w of the Boolean shares, the overall complexity is thus $O(d^2 \log w)$.

Now that the randomness generation is decribed, each intermediate gadget will be proven either d-NI or d-NIo secure. Then, the global composition is proven d-NIo secure as well.

Lemma 2. *Gadget SecAdd is d-NI secure.*

Proof. Gadget SecAdd is built from the Kogge-Stone adder of [12] with secure AND and secure linear functions such as exponentiations and Boolean additions. As to ensure its security with the combination of these atomic masked functions, the tool maskComp was used to properly insert the mandatory d-SNI refreshings, denoted as Refresh in Algorithm 9. As deeply explained in its original paper, maskComp provides a formally proven d-NI secure implementation. □

Lemma 3. *Gadget SecAddModp is d-NI secure.*

Proof. Gadget SecAddModp is built from the gadget SecAdd and SecAnd and linear operations (like $\oplus$). We use the tool maskComp to generate automatically a verified implementation. Note that the tool automatically adds the two refreshs (line 5 and 7) and provides a formally proven d-NI secure implementation. □

Lemma 4. *Gadget SecArithBoolModp is d-SNI secure.*

Proof. A graphical representation of SecArithBoolModp is in Fig. 4. Let O be a set of observations performed by the attacker on the final returned value, let I_{A_j} be the set of internal observations made in step j in the gadget SecAddModp (line 5), and I_{R_j} be the set of internal observations made in the step j in the initialisation of b (line 3) or in the Refresh (line 4). Assuming that $|O| + \sum(|I_{A_j}| + |I_{R_j}|) \leq d$, the gadget is d-SNI secure, if we can build a simulator allowing to simulate all the internal and output observations made by the attacker using a set S of shares of a such that $|S| \leq \sum(|I_{A_j}| + |I_{R_j}|)$.

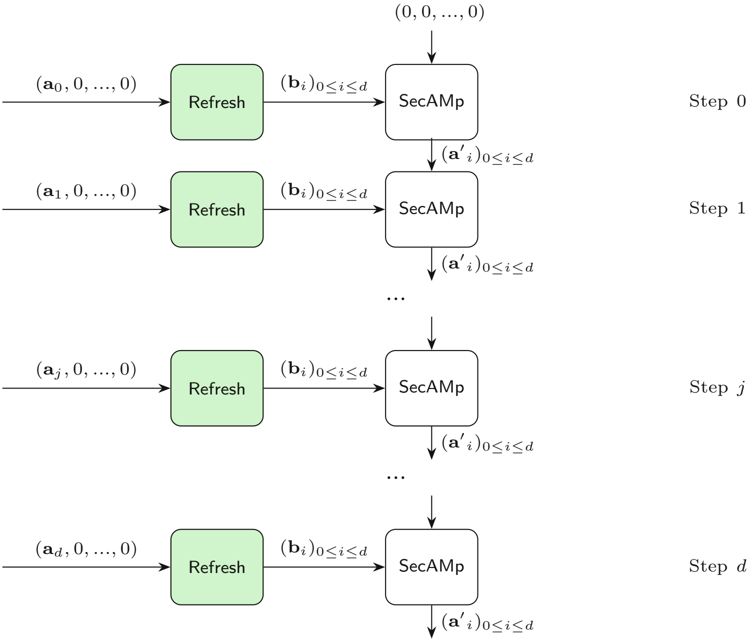
$(0, 0, \ldots, 0)$
$(\mathbf{a}_0, 0, \ldots, 0)$
Refresh
$(\mathbf{b}_i)_{0 \leq i \leq d}$
SecAMp
Step 0
$(\mathbf{a}'_i)_{0 \leq i \leq d}$
$(\mathbf{a}_1, 0, \ldots, 0)$
Refresh
$(\mathbf{b}_i)_{0 \leq i \leq d}$
SecAMp
Step 1
$(\mathbf{a}'_i)_{0 \leq i \leq d}$
...
$(\mathbf{a}_j, 0, \ldots, 0)$
Refresh
$(\mathbf{b}_i)_{0 \leq i \leq d}$
SecAMp
Step j
$(\mathbf{a}'_i)_{0 \leq i \leq d}$
...
$(\mathbf{a}_d, 0, \ldots, 0)$
Refresh
$(\mathbf{b}_i)_{0 \leq i \leq d}$
SecAMp
Step d
$(\mathbf{a}'_i)_{0 \leq i \leq d}$

Fig. 4. Graphical Representation of SecArithBoolModp (The green (resp. white) gadgets will be proved d-SNI (resp. d-NI)) (Color figure online)

At the last iteration (see Fig. 5), the set of observations $O \cup I_{A_d}$ can be simulated using a set $S_{a'_{d-1}}$ of shares of a' and $S_{b_{d-1}}$ of shares of b with $|S_{a'_{d-1}}| \leq |O| + |I_{A_d}|$ and $|S_{b_{d-1}}| \leq |O| + |I_{A_d}|$ (because the gadget SecAddModp is d-NI secure). Since the Refresh is d-SNI secure, the sets $S_{b_{d-1}}$ and I_{R_d} can be simulated using a set $S_{b'_{d-1}}$ of input share with $|S_{b'_{d-1}}| \leq |I_{R_d}|$. If I_{R_d} is not empty, then $S_{b'_{d-1}}$ may contain a_d, so we add a_d to S. For each iteration of the loop this process can be repeated. At the very first iteration, several shares of a' may be necessary to simulate the set of observations. However, there are all initialized to 0, nothing is added to S.

At the end we can conclude that the full algorithm can be simulated using the set S of input shares. Furthermore we have $|S| \leq \sum |I_{R_j}|$ (since a_j is added

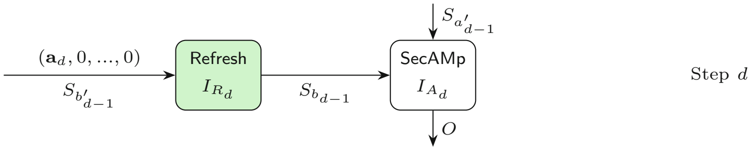

Fig. 5. Last step of SecArithBoolModp with probes (The green (resp. white) gadgets will be proved d-SNI (resp. d-NI)) (Color figure online)

in S only if I_{R_j} is not empty), so we can conclude that $|S| \leq \sum |I_{A_j}| + |I_{R_j}|$ which concludes the proof. □

Lemma 5. *Gadget* RG *is d-NIo secure with public output b.*

Proof. Here we need to ensure that the returned shares of a cannot be revealed to the attacker through a d-order side-channel attack. Since xGen and aGen are just random generation of shares, the idea is to prove that any set of $t \leq d$ observations on RG including these inputs can be perfectly simulated with at most t shares of x and t shares of a.

Gadget RG is built with no cycle. In this case, from the composition results of [4], it is enough to prove that each sub-gadget is d-NI to achieve global security. In our case, it is enough to prove that each sub-gadget is d-NIo with the knowledge of b to achieve global security.

From Lemmas 2, 4, and 3, SecAdd, SecArithBoolModp, and SecAddModp are d-NI secure. $\gg$ is trivially d-NI secure as well since it applies a linear function, gadget FullRefresh is d-SNI secure thus d-NI secure by definition, and gadget kGen is generating shares of a non sensitive value.

At this point, both gadgets FullXor have to be analyzed to achieve the expected overall security. We start with the gadget computing b. After its execution, b is broadcasted. Since b have to be public, its knowledge does not impact the security but because of this output, the security of RG will be d-NIo with public output b and not d-NI. FullXor is composed of a d-SNI secure refreshing (made of $d+1$ linear refreshing) of the shares and of a Boolean addition of these resulting shares. The attacker is not able to observe intermediate variable of all the linear refreshings (since he only has $\delta \leq d$ available observations), thus we consider that the i^{th} refreshing is left unobserved. As a consequence, all the previous observations involve only one b's share and all the following observations are independent from b's share except for their sum. That is, FullXor is d-NI secure. As for its second instance to compute a_0, FullXor is still d-NI secure but a_0 is not revealed after its execution. While the attacker is able to observe its value, it is not returned for free. All the $\delta_0 \leq d$ observations made by the attacker of this last instance of FullXor can be perfectly simulated with a_0 (for the observations performed after the unobserved linear refreshing) and at most $\delta_0 - 1$ shares of z (for the observations made before the unobserved linear refreshing). □

Remark 3. The knowledge of b (ie. the success of the randomness generation) is not sensitive and we decided to consider it as a public output. To simplify the notation when we report the security on the whole scheme, we will omit b in the public outputs.

Lemma 6. *Gadget* DG *is d-NIo secure with public output b.*

Proof. From Lemma 5, Gadget DG is d-NIo secure since it only consists in the linear application of Gadget RG to build the polynomial coefficients. □

Algorithm 15. Rejection sampling (RS)

```
Data: The $4n$ values $a^{(j)}$ to check, in mod-$p$ arithmetic masked representation
      $(a_i^{(j)})_{0 \le i \le d}$.
Result: The bit $r$ equal to 1 if all values satisfy that $a^{(j)} + k - \alpha \ge 0$, and 0
        otherwise.
initialize $(r_i)_{0 \le i \le d}$ as a single-bit Boolean masking of 1
initialize $(p_i)_{0 \le i \le d}$ as a $w$-bit Boolean masking of $-p$
initialize $(p'_i)_{0 \le i \le d}$ as a $w$-bit Boolean masking of $-(p+1)/2$
initialize $(k'_i)_{0 \le i \le d}$ as a $w$-bit Boolean masking of $k - \alpha$
for $j = 1$ to $4n$ do
    $(a'_i)_{0 \le i \le d} \leftarrow \mathsf{SecArithBoolModp}\big((a_i^{(j)})_{0 \le i \le d}\big)$
    $(\delta_i)_{0 \le i \le d} \leftarrow \mathsf{SecAdd}\big((a'_i)_{0 \le i \le d}, (p'_i)_{0 \le i \le d}\big)$
    $(b_i)_{0 \le i \le d} \leftarrow (\delta_i)_{0 \le i \le d} \gg (w-1)$
    $(s_i)_{0 \le i \le d} \leftarrow \mathsf{SecAdd}\big((a'_i)_{0 \le i \le d}, (p_i)_{0 \le i \le d}\big)$
    $(c_i)_{0 \le i \le d} \leftarrow \mathsf{Refresh}\big((b_i)_{0 \le i \le d}\big)$
    $(a'_i)_{0 \le i \le d} \leftarrow \mathsf{SecAnd}\big((a'_i)_{0 \le i \le d}, (\widetilde{c_i})_{0 \le i \le d}\big)$
    $(c_i)_{0 \le i \le d} \leftarrow \mathsf{Refresh}\big((b_i)_{0 \le i \le d}\big)$
    $(a'_i)_{0 \le i \le d} \leftarrow (a'_i)_{0 \le i \le d} \oplus \mathsf{SecAnd}\big((s_i)_{0 \le i \le d}, \neg(\widetilde{c_i})_{0 \le i \le d}\big)$
    $(\delta_i)_{0 \le i \le d} \leftarrow \mathsf{SecAdd}\big((a'_i)_{0 \le i \le d}, (k'_i)_{0 \le i \le d}\big)$
    $(b_i)_{0 \le i \le d} \leftarrow (\delta_i)_{0 \le i \le d} \gg (w-1)$
    $(r_i)_{0 \le i \le d} \leftarrow \mathsf{SecAnd}\big((r_i)_{0 \le i \le d}, \neg(b_i)_{0 \le i \le d}\big)$
end
$r \leftarrow \mathsf{FullXor}\big((r_i)_{0 \le i \le d}\big)$
return $r$
```

Rejection Sampling (RS). Right before the rejection sampling step of the masked signing algorithm, the candidate signature polynomials $\mathbf{z}_1$ and $\mathbf{z}_2$ have been obtained as sums of $d+1$ shares modulo p, and we want to check whether the coefficients in $\mathbb{Z}/p\mathbb{Z}$ represented by those shares are all in the interval $[-k+\alpha, k-\alpha]$. Again, carrying out this check using mod-p arithmetic masking seems difficult, so we again resort to Boolean masking.

For each coefficient $z_{i,j}$ of $\mathbf{z}_1$ and $\mathbf{z}_2$, one can trivially obtain a mod-p arithmetic masked representation of both $z_{i,j}$ and $-z_{i,j}$, and the goal is to check whether those values, when unmasked modulo p in the interval $[(-p+1)/2, (p-1)/2)]$, are all greater than $-k+\alpha$.

Let a be one of those values, and $a = a_0 + \cdots + a_d \bmod p$ its masked representation. Using SecArithBoolModp as above, we can convert this mod-p arithmetic masking to a w-bit Boolean masking $(a'_i)_{0 \le i \le d}$. From this masking, we first want to obtain a masking of the centered representative of $a \bmod p$, i.e. the value a'' such that:

$$a'' = \begin{cases} a & \text{if } a \le (p-1)/2, \\ a - q & \text{otherwise.} \end{cases}$$

This can be done using a similar approach as the one taken for randomness generation: compute a Boolean masking $(b_i)_{0 \le i \le d}$ of the most significant bit $a - (p+1)/2$ (which is 1 in the first case and 0 in the second case), and a Boolean masking $(s_i)_{0\leq i\leq d}$ of the sum $a-q$. Then, a Boolean masking of $(a''_i)_{0\leq i\leq d}$ is obtained as:

$$(a''_i)_{0\leq i\leq d} = \mathsf{SecAnd}\big((a'_i)_{0\leq i\leq d}, (\widetilde{b_i})_{0\leq i\leq d}\big) \oplus \mathsf{SecAnd}\big((s_i)_{0\leq i\leq d}, \neg(\widetilde{b_i})_{0\leq i\leq d}\big).$$

Finally, once this Boolean masking is obtained, it suffices to add $k-\omega$ to it and check the most significant bit to obtain the desired test.

We cannot directly unmask that final bit, but we can compute it in masked form for all the $4n$ values to be tested, and apply SecAnd iteratively on all of these values to compute a Boolean masked representation of the bit equal to 1 if all the coefficients are in the required intervals, and 0 otherwise. This bit can be safely unmasked, and is the output of our procedure. The whole algorithm is summarized as Algorithm 15.

Since both SecArithBoolModp and SecAnd have quadratic complexity in the masking order (and SecArithBoolModp has logarithmic complexity in the size w of the Boolean shares), the overall complexity of this algorithm is $O(nd^2 \log w)$.

Lemma 7. *Gadget RS is d-NI secure.*

Proof. From Lemmas 2 and 4, Gadgets SecArithBoolModp and SecAdd are d-NI secure. Gadget SecAnd is d-SNI secure from [4,30] and $\gg$ is linear, thus trivially d-NI secure as well.

As done for Gadget SecAdd, the tool maskComp was called to generate a d-NI circuit from the initial sequence of gadgets. It inserted gadgets Refresh (as shown in Algorithm 15) at specific locations so that the overall circuit is formally proven to be d-NI secure. □

Refresh and Unmask (FullAdd). This part provides a computation of the sensitive value as the sum of all its shares. It is a gadget with public output because the final value is returned and also output. This output is useful when FullAdd is used to recombine the intermediate value r.

Before summing, the sharing is given as input for FullRefresh [10, Algorithm 4], which is made of a succession of $d+1$ linear refresh operations. Those linear refreshing modify the sharing by adding randoms elements to each share while keeping constant the value of the sum. Their number is strictly superior to d which is useful to consider that any share or strictly partial sum of shares at the output of the final linear refreshing is independent from the original sharing. Then, the following partial sums do not give any information about the original sharing which is dependent of the sensitive values. The whole algorithm, given in Algorithm 16 has a quadratic complexity in d.

Lemma 8. *Gadget FullAdd is d-NIo secure with public output r.*

Proof. Let $\delta \leq d$ be the number of observations made by the attacker. We use a combination of $d+1$ linear refresh operations. That is, there is at least one of the linear refreshing (we call it the i^{th} refreshing) which is not observed by the attacker. For all the $\delta_1 \leq \delta$ observations preceding the i^{th} refreshing in FullAdd,

Algorithm 16. FullAdd

```
Data: $(\mathbf{r}_i)_{0\leq i\leq d}$
Result: $\mathbf{r}$
if $(\mathbf{r}_i)_{0\leq i\leq d} = \perp$ then
    return $\perp$
end
$(\mathbf{r}_i)_{0\leq i\leq d} \leftarrow$ FullRefresh $((\mathbf{r}_i)_{0\leq i\leq d})$
$\mathbf{r} \leftarrow (\mathbf{r}_0 + ... + \mathbf{r}_d)$
output $(\mathbf{r})$
return $(\mathbf{r})$
```

Algorithm 17. H^1

```
Data: $\mathbf{a}$, $(\mathbf{y}_{1,i})_{0\leq i\leq d}$
for $0 \leq i \leq d$ do
    $\mathbf{r}_i \leftarrow \mathbf{a}\mathbf{y}_{1,i} + \mathbf{y}_{2,i}$
end
return $(\mathbf{r}_i)_{0\leq i\leq d}$
```

Algorithm 18. H^2

```
Data: $RejSp$, $(\mathbf{z}_{1,i})_{0\leq i\leq d}$
if $RejSp = 0$ then
    $(\mathbf{z}_{1,i})_{0\leq i\leq d} \leftarrow \perp$
end
return $(\mathbf{z}_{1,i})_{0\leq i\leq d}$
```

they can be perfectly simulated with at most δ_1 shares of $\mathbf{r}$ since each one of them involves at most one $\mathbf{r}_i$. As for the observations performed after the i^{th} refreshing, each one of them is independent from the $\mathbf{r}_i$ inside the refresh mask and each intermediate sum of the unmask part is independent of the $\mathbf{r}_i$ as well with the knowledge of $\mathbf{r}$. Then, during the sum computation, all the $\mathbf{r}_i$ can be simulated with fresh random that sum to $\mathbf{r}$ (the public output). Thus, at most δ shares of $\mathbf{r}$ and $\mathbf{r}$ itself are enough to simulate further probes. □

Remark 4. When FullAdd is used at the very end of the whole algorithm (mKD or mSign), the public outputs are also among the returned values. Then, in those cases, it can be considered as d-NI.

Transition Parts. The elementary parts H^1 and H^2 are quite easy to build since they perform only linear operations on the input data. A masked implementation only performs these linear operations on each share to securely compute the returned shares. H^1 and H^2 are described in Algorithms 17 and 18.

Lemma 9. *Gadget* H^2 *and* H^1 *are d-NI secure.*

The straightforward proof is given in the full version of this paper.

Hash Function. The hash function does not manipulate any sensitive data. Thus, it is left unmasked.

4.3 Proofs of Composition

Theorem 2. *The masked GLP sign in Algorithm 6 is d-NIo secure with public output* $\{r, b\}$.

Proof. From Lemmas 6, 7 and 9, Algorithms DG, RS, H^1, and H^2 are all d-NI. From Lemma 8, FullAdd is d-NIo secure.

Let us assume that an attacker has access to $\delta \leq d$ observations on the whole signature scheme. Then, we want to prove that all these δ observations can be perfectly simulated with at most δ shares of each secret among $\mathbf{y}_1$, $\mathbf{y}_2$, $\mathbf{s}_1$ and $\mathbf{s}_2$ and the public variables. With such a result, the signature scheme is then secure in the d-probing model since no set of at most d observations would give information on the secret values.

In the following, we consider the following distribution of the attacker's δ observations: δ_1 (resp. δ_2) on the instance of DG that produces shares of $\mathbf{y}_1$ (resp. $\boldsymbol{y}_2$), δ_3 on H^1, δ_4 on FullAdd of $\mathbf{r}$, δ_5 (resp. δ_6) on H^1 which produces $\mathbf{z}_1$ (resp. $\mathbf{z}_2$), δ_7 on the instance of RS, δ_8 (resp. δ_9) on H^2 applied on $\mathbf{z}_1$ (resp. $\mathbf{z}_2$), and δ_{10} (resp. δ_{11}) on FullAdd of $\mathbf{z}_1$ (resp. $\mathbf{z}_2$). Some other observations can be made on the *Hash* function, their number won't matter during the proof. Finally, we have $\sum_{i=1}^{11} \delta_i \leq \sum_{i=1}^{11} + \delta_{Hash} \leq \delta$.

Now, we build the proof from right to left as follows.

Both last FullAdd blocks in the very end of mSign are d-NI secure, then all the observations performed during the execution of FullAdd on $\mathbf{z}_1$ (resp. $\mathbf{z}_2$) can be perfectly simulated with at most δ_{10} (resp. δ_{11}) shares of $\mathbf{z}_1$ (resp. $\mathbf{z}_2$).

H^2 is d-NI secure, then all the observations from the call of H^2 on $\mathbf{z}_1$ (resp. $\mathbf{z}_2$) can be perfectly simulated with $\delta_8 + \delta_{10}$ (resp. $\delta_9 + \delta_{11}$) shares of the sensitive input $\mathbf{z}_1$ (resp. $\mathbf{z}_2$). The inputs $\mathbf{z}_1$ and $\mathbf{z}_2$ do not come from RS which do not act on them. They are directly taken from the returned values of H^1.

RS is d-NI secure and do not return any sensitive element, then all the observations performed in gadget RS can be perfectly simulated with at most δ_7 shares of $\mathbf{z}_1$ and $\mathbf{z}_2$. So, after H^1, the observations can be simulated with $\delta_7 + (\delta_8 + \delta_{10})$ shares of $\mathbf{z}_1$ and $\delta_7 + (\delta_9 + \delta_{11})$ shares of $\mathbf{z}_2$.

H^1 is d-NI secure as well, thus all the observations from the call of H^1 on $\mathbf{y}_1$ can be perfectly simulated with $\delta_5 + \delta_7 + \delta_8 + \delta_{10} \leq \delta$ shares of $\mathbf{y}_1$ and $\mathbf{s}_1$. Respectively, on $\mathbf{y}_2$, the observations can be perfectly simulated from $\delta_6 + \delta_7 + \delta_9 + \delta_{11} \leq \delta$ shares of $\mathbf{y}_2$ and $\mathbf{s}_2$.

The left FullAdd gadget is d-NIo secure and do not return any sensitive element, then all the observations performed in this gadget can be perfectly simulated with at most δ_4 shares of $\mathbf{r}$.

The left H^1 gadget is d-NI secure, thus all the observations from its call can be perfectly simulated with at most $\delta_3 + \delta_4$ shares of each one of the inputs $\mathbf{y}_1$ and $\mathbf{y}_2$.

DG is also d-NI secure, thus we need to ensure that the number of reported observations does not exceed δ. At the end of DG, the simulation relies on $(\delta_3 + \delta_4) + (\delta_5 + \delta_7 + \delta_8 + \delta_{10}) \leq \delta$ shares of $\mathbf{y}_1$ and $(\delta_3 + \delta_4) + (\delta_6 + \delta_7 + \delta_9 + \delta_{11}) \leq \delta$ shares of $\mathbf{y}_2$. With the additional δ_1 (resp. δ_2) observations performed on the first (resp. the second) instance of DG, the number of observations remains below δ which is sufficient to ensure security of the whole scheme in the d-probing model. □

Theorem 3. *The masked GLP key derivation in Algorithm 5 is d-NIo secure with public output b.*

Proof. From Lemmas 6 and 9, Algorithms DG, H^1 are all d-NI. From Lemma 8, $\mathsf{FullAdd}$ is d-NIo secure.

Here too, let us assume that an attacker has access to $\delta \leq d$ observations on the whole signature scheme. Then, we want to prove that all these δ observations can be perfectly simulated with at most δ shares of each secret among $\mathbf{s}_1$ and $\mathbf{s}_2$.

We now consider the following distribution of the attacker's δ observations: δ_1 (resp. δ_2) on the instance of DG that produces shares of $\mathbf{s}_1$ (resp. $\mathbf{s}_2$), δ_3 on H^1, and δ_4 on $\mathsf{FullAdd}$, such that $\sum_{i=1}^{4} \delta_i = \delta$.

Now, we build the proof from right to left: $\mathsf{FullAdd}$ is used at the very end of mKD, so it is d-NI secure. Thus, all the observations from the call of $\mathsf{FullAdd}$ can be perfectly simulated with $\delta_4 \leq \delta$ sensitive shares of the input $\mathbf{t}$.

H^1 is d-NI, thus all the observations from its call can be perfectly simulated with at most $\delta_3 + \delta_4 \leq \delta$ shares of each one of the inputs $\mathbf{s}_1$ and $\mathbf{s}_2$.

DG is d-NIo, thus we need to ensure that the number of reported observations does not exceed δ. At the end of DG, the simulation relies on $(\delta_3+\delta_4) \leq \delta$ shares of $\mathbf{s}_1$ and $\mathbf{s}_2$. With the additional δ_1 (resp. δ_2) observations performed on the first (resp. the second) instance of DG, the number of observations on each block remains below δ. All the observations can thus be perfectly simulated with the only knowledge of the outputs, that is, the key derivation algorithm is this d-NIo secure. □

5 Implementation of the Countermeasure

We have carried out a completely unoptimized implementation of our masking countermeasure based on a recent, public domain implementation of the GLP signature scheme called GLYPH [8,9]. The GLYPH scheme actually features a revised set of parameters supposedly achieving a greater level of security (namely, $n = 1024$, $p = 59393$, $k = 16383$ and $\alpha = 16$), as well as a modified technique for signature compression. We do not claim to vouch for those changes, but stress that, for our purposes, they are essentially irrelevant. Indeed, the overhead of our countermeasure only depends on the masking order d, the bit size w of Boolean masks (which should be chosen as $w = 32$ both for GLYPH and the original GLP parameters) and the degree n of the ring $\mathcal{R}$ (which is the same in GLYPH as in the high-security GLP parameters). Therefore, our results on GLYPH should carry over to a more straightforward implementation of GLP as well.

Implementation results on a single core of an Intel Core i7-3770 CPU are provided in Table 1. In particular, we see that the overhead of our countermeasure with 2, 3 and 4 shares (secure in the d-probing model for $d = 1, 2, 3$ respectively) is around 15×, 30× and 73×. In view of the complete lack of optimizations

Table 1. Implementation results. Timings are provided for 100 executions of the signing and verification algorithms, on one core of an Intel Core i7-3770 CPU-based desktop machine.

Number of shares $(d+1)$	Unprotected	2	3	4	5	6
Total CPU time (s)	0.540	8.15	16.4	39.5	62.1	111
Masking overhead	—	×15	×30	×73	×115	×206

of this implementation, we believe that those results are quite promising. The memory overhead is linear in the masking order, so quite reasonable in practice (all masked values are simply represented as a vector of shares).

For future work, we mention several ways in which our implementation could be sped up:

- For simplicity, we use a version of SecArithBoolModp with cubic complexity in the masking order, as in [13, Sect. 4.1]. Adapting the quadratic algorithm of [13, Sect. 4.2] should provide a significant speed-up. Moreover, for small values of d, Coron's most recent algorithm [11] should be considerably faster. However, the technique from [11] unfortunately has an overhead exponential in the masking order, so it is not suitable for our purpose of masking GLP at any order.
- Several of our algorithms call SecAdd on two masked values one of which is actually a public constant. One could use a faster SecAddConst procedure that only protect the secret operand instead.
- Our algorithms are generic, and do not take advantage of the special shape of k for example. In the case of GLYPH, a comparison to $k = 2^{14} - 1$ could be greatly simplified.
- One key way in which masking affects the efficiency of GLP signing is in the computation of the product $\boldsymbol{a} \cdot \boldsymbol{y}_1$. This product is normally carried out using a number-theoretic transform (NTT), with $O(n \log n)$ complexity. However, the NTT is not linear, and is thus inconvenient to use when $\boldsymbol{y}_1$ is masked. In our implementation, we use the schoolbook $O(n^2)$ polynomial multiplication instead. However, one could consider other approaches: either use a faster linear algorithm, like Karatsuba or Toom–Cook, or try and mask the NTT itself.
- Many other more technical improvements are also possible: for example, we have made no attempt to reduce the number of unnecessary array copies.

6 Conclusion

In this paper, we have described a provably secure masking of the GLP lattice-based signature scheme, as well as a proof-of-concept implementation thereof. The security proof itself involved a number of new techniques in the realm of masking countermeasures. Our method should apply almost identically to other lattice-based Fiat–Shamir type signature schemes using uniform distributions in intervals (as opposed to Gaussian distributions). This includes the Bai–Galbraith signature scheme [2], as well as the recently proposed Dilithium signature [16].

We have mostly ignored the issue of signature compression, which is an important one in all of these constructions, GLP included. However, it is easy to see that compression can be securely applied completely separately from our countermeasure: this is because it only affects already generated signatures (which are non-sensitive) as well as the input to the hash function (which is already unmasked in our technique).

On the other hand, extending our approach to schemes using Gaussian distributions appears to be really difficult: neither Boolean masking nor arithmetic masking with uniform masks seems particularly well-suited to address the problem. One way to tackle the problem might be to consider masking with non-uniform noise, and only achieving statistically close instead of perfect simulatability. Developing such a framework, however, is certainly a formidable challenge.

Masking hash-and-sign type signatures in using GPV lattice trapdoors is probably even harder, as they involve Gaussian sampling not only in $\mathbb{Z}$ but on arbitrary sublattices of $\mathbb{Z}^n$, with variable centers. It seems unlikely that a masked GPV signature scheme can achieve a reasonable level of efficiency.

Finally, while we have used the maskComp tool to securely instantiate the masked versions of some of the gadgets we use in our construction, it would be interesting to leverage recent advances in verification [3] and synthesis [4] of masked implementations in a more systematic way in the lattice-based setting. Even for verification, the sheer size of the algorithms involved poses significant challenges in terms of scalability; however, automated tool support would be invaluable for the further development of masking in the postquantum setting.

Acknowledgements. We are indebted to Vadim Lyubashevsky for fruitful discussions, and to the reviewers of EUROCRYPT for their useful comments. We acknowledge the support of the French Programme d'Investissement d'Avenir under national project RISQ. This work is also partially supported by the European Union PROMETHEUS project (Horizon 2020 Research and Innovation Program, grant 780701) and ONR Grant N000141512750.

References

1. Abdalla, M., An, J.H., Bellare, M., Namprempre, C.: From identification to signatures via the fiat-shamir transform: minimizing assumptions for security and forward-security. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 418–433. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-46035-7_28
2. Bai, S., Galbraith, S.D.: An improved compression technique for signatures based on learning with errors. In: Benaloh, J. (ed.) CT-RSA 2014. LNCS, vol. 8366, pp. 28–47. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-04852-9_2
3. Barthe, G., Belaïd, S., Dupressoir, F., Fouque, P.-A., Grégoire, B., Strub, P.-Y.: Verified proofs of higher-order masking. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9056, pp. 457–485. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46800-5_18

4. Barthe, G., Belaïd, S., Dupressoir, F., Fouque, P.-A., Grégoire, B., Strub, P.-Y., Zucchini, R.: Strong non-interference and type-directed higher-order masking. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 16, pp. 116–129. ACM Press, October 2016
5. Barthe, G., Belaïd, S., Espitau, T., Fouque, P.-A., Grégoire, B., Rossi, M., Tibouchi, M.: Masking the GLP lattice-based signature scheme at any order. Cryptology ePrint Archive (2018). http://eprint.iacr.org/. Full version of this paper
6. Bindel, N., Buchmann, J.A., Krämer, J.: Lattice-based signature schemes and their sensitivity to fault attacks. In: Maurine, P., Tunstall, M. (eds.) FDTC 2016, pp. 63–77. IEEE Computer Society (2016)
7. Chari, S., Rao, J.R., Rohatgi, P.: Template attacks. In: Kaliski Jr., B.S., Koç, K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 13–28. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36400-5_3
8. Chopra, A.: GLYPH: a new insantiation of the GLP digital signature scheme. Cryptology ePrint Archive, Report 2017/766 (2017). http://eprint.iacr.org/2017/766
9. Chopra, A.: Software implementation of GLYPH. GitHub repository (2017). https://github.com/quantumsafelattices/glyph
10. Coron, J.-S.: Higher order masking of look-up tables. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 441–458. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-55220-5_25
11. Coron, J.-S.: High-order conversion from Boolean to arithmetic masking. Cryptology ePrint Archive, Report 2017/252 (2017). http://eprint.iacr.org/2017/252
12. Coron, J.-S., Großschädl, J., Tibouchi, M., Vadnala, P.K.: Conversion from arithmetic to Boolean masking with logarithmic complexity. In: Leander, G. (ed.) FSE 2015. LNCS, vol. 9054, pp. 130–149. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48116-5_7
13. Coron, J.-S., Großschädl, J., Vadnala, P.K.: Secure conversion between Boolean and arithmetic masking of any order. In: Batina, L., Robshaw, M. (eds.) CHES 2014. LNCS, vol. 8731, pp. 188–205. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44709-3_11
14. Duc, A., Dziembowski, S., Faust, S.: Unifying leakage models: from probing attacks to noisy leakage. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 423–440. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-55220-5_24
15. Ducas, L., Durmus, A., Lepoint, T., Lyubashevsky, V.: Lattice signatures and bimodal Gaussians. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8042, pp. 40–56. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40041-4_3
16. Ducas, L., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., Stehle, D.: CRYSTALS - dilithium: digital signatures from module lattices. Cryptology ePrint Archive, Report 2017/633 (2017). http://eprint.iacr.org/2017/633
17. Espitau, T., Fouque, P.-A., Gérard, B., Tibouchi, M.: Loop-abort faults on lattice-based Fiat-Shamir and hash-and-sign signatures. In: Avanzi, R., Heys, H. (eds.) SAC 2016. LNCS, vol. 10532, pp. 140–158. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69453-5_8
18. Espitau, T., Fouque, P.-A., Gérard, B., Tibouchi, M.: Side-channel attacks on BLISS lattice-based signatures: exploiting branch tracing against strongSwan and electromagnetic emanations in microcontrollers. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 17, pp. 1857–1874. ACM Press, October/November 2017

19. Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In: Ladner, R.E., Dwork, C. (eds.) 40th ACM STOC, pp. 197–206. ACM Press, May 2008
20. Groot Bruinderink, L., Hülsing, A., Lange, T., Yarom, Y.: Flush, gauss, and reload – a cache attack on the BLISS lattice-based signature scheme. In: Gierlichs, B., Poschmann, A.Y. (eds.) CHES 2016. LNCS, vol. 9813, pp. 323–345. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53140-2_16
21. Güneysu, T., Lyubashevsky, V., Pöppelmann, T.: Practical lattice-based cryptography: a signature scheme for embedded systems. In: Prouff, E., Schaumont, P. (eds.) CHES 2012. LNCS, vol. 7428, pp. 530–547. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33027-8_31
22. Ishai, Y., Sahai, A., Wagner, D.: Private circuits: securing hardware against probing attacks. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 463–481. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45146-4_27
23. Lyubashevsky, V.: Fiat-Shamir with aborts: applications to lattice and factoring-based signatures. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 598–616. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-10366-7_35
24. Lyubashevsky, V.: Lattice signatures without trapdoors. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 738–755. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29011-4_43
25. Oder, T., Schneider, T., Pöppelmann, T., Güneysu, T.: Practical CCA2-secure and masked ring-LWE implementation. Cryptology ePrint Archive, Report 2016/1109 (2016). http://eprint.iacr.org/2016/1109
26. Pessl, P., Bruinderink, L.G., Yarom, Y.: To BLISS-B or not to be: attacking strongSwan's implementation of post-quantum signatures. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 17, pp. 1843–1855. ACM Press, October/November 2017
27. Pöppelmann, T., Ducas, L., Güneysu, T.: Enhanced lattice-based signatures on reconfigurable hardware. In: Batina, L., Robshaw, M. (eds.) CHES 2014. LNCS, vol. 8731, pp. 353–370. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44709-3_20
28. Reparaz, O., de Clercq, R., Roy, S.S., Vercauteren, F., Verbauwhede, I.: Additively homomorphic ring-LWE masking. In: Takagi, T. (ed.) PQCrypto 2016. LNCS, vol. 9606, pp. 233–244. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-29360-8_15
29. Reparaz, O., Sinha Roy, S., Vercauteren, F., Verbauwhede, I.: A masked ring-LWE implementation. In: Güneysu, T., Handschuh, H. (eds.) CHES 2015. LNCS, vol. 9293, pp. 683–702. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48324-4_34
30. Rivain, M., Prouff, E.: Provably secure higher-order masking of AES. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 413–427. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15031-9_28