Post-Quantum ID-based Ring Signatures from Symmetric-key Primitives

Maxime Buser[1], Joseph K. Liu[1], Ron Steinfeld[1], and Amin Sakzad[1]

Faculty of Information Technology, Monash University, Australia
{`maxime.buser,joseph.liu,ron.steinfeld,amin.sakzad`}`@monash.edu`

Abstract. Ring signatures and ID-based cryptography are considered promising in terms of application. A ring signature authenticates messages while the author of the message remains anonymous. ID-based cryptographic primitives suppress the need for certificates in public key infrastructures (PKI). In this work, we propose a generic construction for post-quantum ID-based ring signatures (IDRS) based on symmetric-key primitives from which we derive the first two constructions of IDRS. The first construction named PicRS utilizes the Picnic digital signature to ensure its security while the second construction XRS is motivated by the stateful digital signature XMSS instead of Picnic, allowing a signature size reduction. Both constructions have a competitive signature size when compared with state-of-the-art lattice-based IDRS. XRS can achieve a competitive signature size of 889KB for a ring of 4096 users while the fully stateless PicRS achieves a signature size of 1.900MB for a ring of 4096 users. In contrast, the shortest lattice-based IDRS achieves a signature size of 335MB for the same ring size.

Keywords: ID-based ring signature · Applied post-quantum cryptography · Symmetric-key primitives.

1 Introduction

Ring signatures [31] are currently considered one of the most valuable cryptographic primitives to ensure privacy. They allow a member of a group (i.e. ring) to anonymously sign a message on behalf of a group in a spontaneous manner. This spontaneity allows signers to form a group of their own choice and to generate an anonymous signature. According to their great promise in providing authenticity and anonymity, ring signatures have attracted a lot of interest from the research community.
ID-based Ring Signature (IDRS): ID-based cryptography [32] was introduced in 1984 to erase the need for certificates in public key infrastructures (PKI). ID-based cryptography utilizes a public key which is the identity of the user, for example, an identity can be an email address or a name. In this framework, a trusted third party named the private key generator (PKG) is required. PKG

uses the identity of a user and his private key to generate the secret signing key of the corresponding identity. The interest in combining both ID-based cryptography and ring signature is undeniable as proven by the works [4,15,21,24,16,35]. As explained by Chow et al. [16], the main advantage of ID-based Ring signature (IDRS) over "traditional" ring signatures in PKI is that IDRS provides better spontaneity. PKI's ring signatures can only form a ring with users that requested certificates for their public keys while in IDRS signers can form a ring using users' identities even if they did not request their secret signing keys to PKG. Additionally, IDRS may significantly reduce the communication overhead in sending the list of ring public keys to the verifier along with the signature for PKI's RS. The ring IDs may be much shorter and may even be implicitly known by the verifier (e.g. all employees of a certain organization/department).
Post-quantum IDRS: In 2016, the post-quantum (PQ) standardization process was launched by NIST and generated considerable attention from researchers. This also motivates to design post-quantum IDRS as this would provide primitives which ensure anonymity without requiring any certificates. There exist different PQ candidates to design quantum-safe cryptographic primitives. Lattice-based cryptography is currently the most investigated candidate due to its promise of flexibility. There are currently multiple lattice-based IDRS; the first one by Wang [33] and more recent works by Zhao et al. [36], Wei et al. [34] and Cao et al. [12].To the best of our knowledge, those are the only quantum-safe IDRS.
In this work, we will focus on another promising candidate: symmetric-key primitives, for example hash functions or block ciphers. These are old primitives providing the advantage of a well-studied and well-understood security. Another advantage is that the security of a symmetric-key-based protocol depends only on the integrated primitives and not on any assumed hard problem. This means that, if a symmetric-key primitive has been broken, it can simply be replaced and the design would still be secure. On the contrary, if the hardness assumption of lattice-based constructions has been broken, then all schemes relying on this assumption are not secure anymore. The recent design of zero-knowledge proof systems obtained from symmetric-key primitives opens up new directions. Zero-knowledge proof systems as ZKBoo [22], ZKB++ [13], KKW [28], ZK-STARK [6], Aurora [7] and Ligero++ [9] allow a user to prove the knowledge of a secret witness w such that $C(w) = 1$, where C is a public circuit similar to hash functions. For all these aforementioned reasons, our work studies PQ IDRS constructed based on symmetric-key primitives only.

1.1 Contributions

The contributions of this work can be presented in the following three parts:
Generic post-quantum ID-based ring signatures (Section 3): We designed a circuit C (see Section 3.1) to allow signers to execute a zero-knowledge proof that they own a witness w such that $C(w) = 1$. C is divided into two subcircuits: C_1 and C_2. C_1 proves the membership of the signer to the ring through an accumulator (see Definition 10) and C_2 proves the knowledge of a signing

Table 1: PicRS and XRS comparisons. PQ=post-quantum, $N=$ ring size, ✓=proven, (✓)= assumed, h=XMSS tree height

IDRS	PQ candidate	$\|\sigma\|$ (Asympt.)	Max. N	IDRS.Setup (Asympt.) time	|SID| (KB)	NIZK	PQ	H	(est.)$\|\sigma\|$(MB)		
									$N = 2^6$	$N = 2^{12}$	$N = 2^{20}$
PicRS	Symmetric	$\mathcal{O}(\log N)$	unli.	$\mathcal{O}(1)$	167	KKW	✓	LowMC	169.964	170.153	170.406
								SHA3	3619	3622	3626
						Ligero++	(✓)	SHA3	2.046	2.046	2.047
								MiMC	1.902	1.902	1.903
								Poseidon	1.898	1.899	1.900
XRS	Symmetric	$\mathcal{O}(\log N)$	2^{20}	$\mathcal{O}(2^h)$	4.899	KKW	✓	LowMC	12.487	12.680	12.930
								SHA3	332.300	335.266	339.211
						Ligero++	(✓)	SHA3	1.490	1.491	1.493
								MiMC	0.973	0.976	0.979
								Poseidon	0.885	0.889	0.893
[36]	Lattice	$\mathcal{O}(N)$	unli.	-	615000	-	✓	-	5	335	32243

secret key generated by the private key generator PKG. In our generic IDRS construction, the signing private key for the identity ID is a digital signature of ID generated by PKG, therefore C_2 proves the knowledge of a valid digital signature. Both C_1 and C_2 are linked through an "AND" logical gate ($C = C_1 \cdot C_2$). The generic circuit is illustrated in Fig. 3.

Applicable post-quantum ID-based ring signatures named PicRS and XRS from the generic construction (Section 4): We implemented sub-circuit C_1 with a Merkle Accumulator (Section 4.1) to prove the membership to the ring. sub-circuit C_2 can be initiated in two different ways:

(1) PKG uses Picnic digital signature [13], which means that a signer needs to prove the knowledge of a valid Picnic signature. We designed circuit Picnic.C presented in Section 4.2, Algorithm 1 and Fig. 5 to prove this statement.
(2) PKG uses the stateful digital signature XMSS [25], which means that a signer needs to prove the knowledge of a valid XMSS signature. We designed circuit XMSS.C presented in Section 4.2, Algorithm 2 and Fig. 6 to prove this statement.

Picnic.C allows to design the first IDRS PicRS, where a signer uses circuit C = PicRS.C = Merkle.$C(= C_1)\cdot$Picnic.$C(= C_2)$ to generate a signature. The second implementation named XRS ensues from circuit XMSS.C. In XRS, a signer uses circuit C = XRS.C = Merkle.$C(= C_1) \cdot$ XMSS.$C(= C_2)$ to generate a signature. While PicRS is stateless for the signer and PKG, XRS is still stateless for the signer but requires PKG to keep an updated state to the stateful nature of XMSS. The PicRS's PKG can generate an unlimited number of signing secret keys and therefore handle an unlimited number of users, while there is a cap for the maximum number of users in XRS (e.g. 2^{20} users).

Applicable constructions analysis and optimization (Section 5): We evaluate our constructions with two different zero-knowledge proof systems: KKW [28], Ligero++[9]. Each of them is tested with the standard hash function SHA3 and, additionally with the following non-standard hash functions to optimize the signature size: LowMC [3], MiMC [2] and Poseidon [23]. We optimize the complexity of circuit $\mathsf{Picnic}.C$ and $\mathsf{XMSS}.C$ by testing different parameters for Picnic and XMSS to achieve the best possible signature size. In theory, both schemes have a signature size that grows logarithmically ($\mathcal{O}(\log N)$) with the size of the ring represented by N. This is an improvement when compared with lattice-based IDRSs [33], [36], [34] or [12], whose signature size grows linearly ($\mathcal{O}(N)$) with the ring size, making them unsuitable for large rings. In practice, PicRS and XRS signature sizes are nearly constant because proving the knowledge of a valid Picnic (PicRS) or XMSS (XRS) signature is the bottleneck of both signature sizes. PicRS achieves a size of 1.900MB while XRS requires only 889KB for a ring of 4096 members. Table 1 demonstrates that these sizes are competitive when compared to the current state-of-the-art of lattice-based IDRS introduced by Zhao et al. [36], which is the only work proposing concrete parameters and allowing us to estimates their signature sizes.

1.2 Overview of techniques

At the heart of our generic IDRS, we utilize a non-interactive zero-knowledge proof system (NIZK) based on symmetric-key primitives, which allows us to prove the knowledge of a witness w such that, for a public circuit C, we have $C(w) = 1$. Traditional digital signatures, for example Picnic [13], run the NIZK on a circuit related to the underlying one-way function as in the original zero-knowledge proof based signature schemes like Picnic [13] New challenges arise in IDRS as distinct from a traditional digital signature, namely, the generated proof needs to include a part of the IDRS signature involves the "verification" circuit for the signing secret key generation (i.e. verification algorithm of a standard signature by the key generation authority). This means that optimising the size of the verification circuit for the underlying signature is critical for our IDRS signature size and we focused our efforts in this direction.

In our generic IDRS construction, a signer with an identity ID owns a witness that is the signing secret key SID. SID is a digital signature (see Definition 4), generated by PKG, using its ID as a message. A signer will use the NIZK proving procedure on circuit C to generate a signature of a message m. C is designed to prove that he knows a valid SID, in other words, a valid digital signature generated by PKG for an ID belonging to the ring L. Circuit C takes as inputs (i.e. the witness) the signer's identity ID, the corresponding signing secret key SID and the list of identities L (i.e. the ring). More formally, the signer will prove the knowledge of $(\mathsf{ID}, \mathsf{SID}, \mathsf{L})$ such that $C(\mathsf{ID}, \mathsf{SID}, \mathsf{L}) = 1$. C, summarized in Fig.3, is composed of two main sub-circuits named C_1 and C_2, which are used to prove the membership in the ring and to prove the knowledge of a valid digital signature.

We construct applicable constructions by defining and optimizing circuit C and both its sub-circuits C_1 and C_2. C_1 is implemented as $\mathsf{Merkle}.C$ which is constructed on top of a Merkle accumulator [18,10,28] and a multiplexer (see Section 4.1) to hide the position of the identity into the accumulator. C_2 can be implemented with the verification procedure of the Picnic digital signature or the verification algorithms of the stateful XMSS digital signature. XRS follows the same idea but, instead of having a valid picnic signature as SID, each user owns an XMSS signature. The proof of knowledge of a valid signature will be done through circuit $\mathsf{XMSS}.C(= C_2)$. As the stateful nature of XMSS, $\mathsf{XMSS}.C$ requires the use of multiplexers (see Equation 3) to hide the state and, so, provide anonymity.

1.3 Outline of the paper

This paper is structured as follows. Section 2 formally defines IDRS. Section 3 presents the generic construction. Section 4 introduces both possible instances PicRS and XRS. Section 5 concludes the paper with a full evaluation of the two applicable constructions. Appendix A defines the cryptographic primitives used in this work.

2 Definition of ID-based ring signature (IDRS)

We now formally define an ID-based ring signature. In IDRS, there is a private key generator (PKG), which is a trusted identity generating the signing secret keys of users. Only users who have received a signing secret key SID from PKG can generate a valid and anonymous signature.

Definition 1 (ID-based ring signature). *An ID-based ring signature is defined by the tuple of algorithms:* $\mathsf{IDRS} = (\mathsf{IDRS.Setup}, \mathsf{IDRS.KeyGen}, \mathsf{IDRS.Sign}, \mathsf{IDRS.Verify})$

– $(\mathsf{mpk}, \mathsf{msk}, \mathsf{param}) \leftarrow \mathsf{IDRS.Setup}(1^\lambda)$*: This algorithm takes as input the security parameters* λ*, it produces the master public key* mpk*, the master secret key* msk *and the public parameters* param*. This procedure is executed by the private key generator (PKG).*
– $\mathsf{SID} \leftarrow \mathsf{IDRS.KeyGen}(\mathsf{ID}, \mathsf{msk})$*: This algorithm takes as input an identity* $\mathsf{ID} \in \{0,1\}^*$ *and the master secret key* msk*, it outputs the signer's secret signing key* SID*. This procedure is executed by the private key generator PKG and the result is transmitted to the user with the identity* ID*.*
– $\sigma \leftarrow \mathsf{IDRS.Sign}(m, \mathsf{L}, \mathsf{ID}, \mathsf{SID}, \mathsf{mpk}, \mathsf{param})$*: This algorithm takes as input the message* m*, a list* L *of* N *identities, the identity of the signer* ID*, the signing secret key* SID *of the member* ID*, where* $\mathsf{ID} \in \mathsf{L}$*, the master public key* mpk *and the public parameters* param*. It outputs a ring signature* σ*.*
– $0/1 \leftarrow \mathsf{IDRS.Verify}(m, \mathsf{L}, \sigma, \mathsf{mpk}, \mathsf{param})$*: This algorithm takes as input a ring signature* σ*, a message* m*, the ring list* L*, the master public key* mpk *and the public parameters* param*. It outputs* 1 *if* σ *is valid and generated by one* $\mathsf{ID} \in \mathsf{L}$*,* 0 *otherwise.*

Security properties: A secure ID-based ring signature achieves correctness, unforgeability and anonymity. All security experiments presented in Fig. 1 and 2 are performed between an adversary $\mathcal{A}$ and a challenger $\mathcal{CH}$.

Definition 2 (Unforgeability). *An* IDRS *achieves unforgeability if it is infeasible for an adversary* $\mathcal{A}$ *to generate a valid signature* σ *from the identities of the ring. An* IDRS *achieves unforgeability if and only if for a security parameter* λ *the advantage* $\mathsf{Adv}_{\mathcal{A}}^{Forge}$ *of an adversary* $\mathcal{A}$ *satisfies*

$$\mathsf{Adv}_{\mathcal{A}}^{Forge}(\lambda) = \Pr[\mathsf{Exp}_{\mathcal{A},\mathsf{IDRS}}^{Forge}(\lambda) = 1] < \mathsf{negl}(\lambda), \tag{1}$$

where the unforgeability experiment $\mathsf{Exp}_{\mathcal{A},\mathsf{IDRS}}^{Forge}(\lambda)$ *is presented in Fig. 1 on pg 6.*

$\mathsf{Exp}_{\mathcal{A},\mathsf{IDRS}}^{Forge}(\lambda)$

Setup:
This oracle is executed by the challenger $\mathcal{CH}$ who performs the setup algorithm $(\mathsf{mpk}, \mathsf{msk}, \mathsf{param}) \leftarrow \mathsf{IDRS.Setup}(1^{\lambda})$. PKG's public key mpk is then sent to the adversary $\mathcal{A}$. The sets $\mathcal{Q}, \mathcal{S}$ are initialized: $\mathcal{Q}, \mathcal{S} \leftarrow \emptyset$.
Query:
$\mathsf{SID} \leftarrow \mathcal{CO}(\mathsf{ID})$: This oracle corrupts a user. The adversary $\mathcal{A}$ sends the user's identity ID to the challenger $\mathcal{CH}$ which computes the corresponding secret key $\mathsf{SID} \leftarrow \mathsf{IDRS.KeyGen}(\mathsf{ID}, \mathsf{msk})$ and sends SID to $\mathcal{A}$. ID is added to the set $\mathcal{Q}$.
$\sigma \leftarrow \mathcal{SO}(m, \mathsf{L}, \mathsf{ID}, \mathsf{mpk}, \mathsf{param})$: This signing oracle starts with $\mathcal{A}$ sending to the challenger $\mathcal{CH}$ a list of user's identities L, message m and the identity of the signer $\mathsf{ID} \in \mathsf{L}$ and $\mathcal{CH}$ returns a valid signature σ for message m generated by the user ID ($\sigma \leftarrow \mathsf{IDRS.Sign}(m, \mathsf{L}, \mathsf{ID}, \mathsf{SID}, \mathsf{mpk}, \mathsf{param})$). The tuple (m, L) is added to the set $\mathcal{S}$.
Forgery:
$\mathcal{A}$ forges $(m^*, \mathsf{L}^*, \sigma^*)$.

```
    **if** $1 \leftarrow \mathsf{IDRS.Verify}(m^*, \mathsf{L}^*, \sigma^*, \mathsf{mpk}, \mathsf{param}) \wedge ((m^*, \mathsf{L}^*) \notin \mathcal{S}) \wedge (\mathsf{ID} \notin \mathcal{Q}$ for all $\mathsf{ID} \in \mathsf{L}^*)$
            **then return** 1
    **return** 0
```

Fig. 1: IDRS Unforgeability Game.

Definition 3 (Anonymity). *An* IDRS *for a ring* L*, a message* m *and a signature* $\sigma = \mathsf{IDRS.Sign}(m, \mathsf{L}, \mathsf{ID}, \mathsf{SID}, \mathsf{mpk}, \mathsf{param})$*, achieves anonymity if and only if for a security parameter* λ *the advantage* $\mathsf{Adv}_{\mathcal{A}}^{Anom}$ *of an adversary* $\mathcal{A}$ *satisfies*

$$\mathsf{Adv}_{\mathcal{A}}^{Anom}(\lambda) = |\Pr[\mathsf{Exp}_{\mathcal{A},\mathsf{IDRS}}^{Anom}(\lambda) = 1] - 1/N| < \mathsf{negl}(\lambda), \tag{2}$$

where the anonymity experiment $\mathsf{Exp}_{\mathcal{A},\mathsf{IDRS}}^{Anom}(\lambda)$ *is defined in Fig. 2 on pg 7 and* N *is number of ring members.*

3 Generic construction for ID-based ring signature from symmetric-key primitives

This section introduces our proposed generic construction of IDRS based on symmetric-key primitives. A key part of our proposal is that symmetric-key

$\mathsf{Exp}^{Anom}_{\mathcal{A},\mathsf{IDRS}}(\lambda)$

Setup:
This game is executed by the challenger $\mathcal{CH}$ who performs the setup algorithm $(\mathsf{mpk}, \mathsf{msk}, \mathsf{param}) \leftarrow \mathsf{IDRS.Setup}(1^\lambda)$. PKG's key pair $(\mathsf{mpk}, \mathsf{msk})$ is sent to the adversary $\mathcal{A}$.

Challenge:
$\sigma \leftarrow \mathcal{CHO}(m, \mathsf{L})$: This interactive oracle is initiated by the adversary $\mathcal{A}$, who sends a message m to be signed and a list of identities L to the challenger $\mathcal{CH}$. $\mathcal{CH}$ randomly picks $\mathsf{ID} \in \mathsf{L}$ and generates a valid signature σ for the selected ID ($\sigma \leftarrow \mathsf{IDRS.Sign}(m, \mathsf{L}, \mathsf{ID}, \mathsf{SID}, \mathsf{mpk}, \mathsf{param})$). $\mathcal{CH}$ sends σ to $\mathcal{A}$.

Output:
$\mathcal{A}$ chooses ID^* based on the received σ.

```
    if $\mathsf{ID}^* = \mathsf{ID}$
        then return $1$
    return $0$
```

Fig. 2: IDRS Anonymity Game.

based zero-knowledge proof systems (NIZK) give us the ability to prove the knowledge of an input (i.e. witness) w of a circuit C such that $C(w) = 1$. In an IDRS, the signer needs to demonstrate that (1) he owns a secret key generated by the central authority PKG and that (2) his identity belongs to the rings. This requires a circuit C that proves both statements.

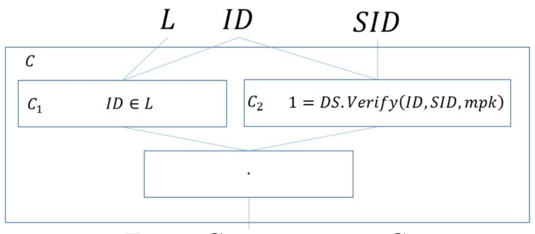


Fig. 3: Generic circuit C

The basic idea behind our generic IDRS is that the PKG possesses a digital signature (see Definition 4) key pair as master public mpk and private key msk (i.e. $\mathsf{mpk} = \mathsf{DS.pk}, \mathsf{msk} = \mathsf{DS.sk}$). Then, each user with an identity ID requests a signing key SID to PKG which generates a digital signature $\mathsf{DS}.\sigma$ taking the identity ID as the message or, in other words, PKG computes $\mathsf{SID} \leftarrow \mathsf{DS.Sign}(\mathsf{ID}, \mathsf{msk})$. To sign a message m, a user in possession of a signing secret key generated by PKG proves the knowledge through a NIZK (see Definition 6) of SID associated with an identify ID belonging to the ring L (i.e $\mathsf{ID} \in \mathsf{L}$). We designed a generic circuit C (see Fig.3) which proves the validity of both statements. C is divided into two sub-circuits" C_1 to prove $\mathsf{ID} \in \mathsf{L}$ and C_2 to prove $1 = \mathsf{DS.Verify}(\mathsf{ID}, \mathsf{SID}, \mathsf{mpk})$. C_1 and C_2 are associated with "AND" gate to form the overall circuit C.

3.1 Generic IDRS algorithms

We now formally define the algorithms for our generic construction of IDRS, which follows Definition 1.

$\underline{(\mathsf{mpk}, \mathsf{msk}, \mathsf{param}) \leftarrow \mathsf{IDRS.Setup}(1^\lambda)}$: This algorithm is executed by PKG and performs the following steps:

- $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{DS.KeyGen}(1^\lambda)$ (see Definition 4)
- $\mathsf{param} \leftarrow \mathsf{A.Gen}(1^\lambda)$ (See Definition 10)
- **Return** $(\mathsf{mpk}, \mathsf{msk})$

$\underline{\mathsf{SID} \leftarrow \mathsf{IDRS.KeyGen}(\mathsf{ID}, \mathsf{msk})}$: This algorithm is executed by PKG on the request of the user with the identity ID. It computes a digital signature $\mathsf{DS}.\sigma$ using the ID as the message. The digital signature $\mathsf{DS}.\sigma$ is transmitted to the user ID with and becomes his signing secret key SID. This procedure executes the following steps:

- $\mathsf{SID} \leftarrow \mathsf{DS.Sign}(\mathsf{ID}, \mathsf{msk})$ (see Definition 4)
- **Return** SID

$\underline{\sigma \leftarrow \mathsf{IDRS.Sign}(m, \mathsf{L}, \mathsf{ID}, \mathsf{SID}, \mathsf{mpk}, \mathsf{param})}$: This procedure takes as inputs the message m, the set of N identities L, in other words the ring, the signing secret key $\mathsf{SID} = \mathsf{DS}.\sigma$ and the master public key and the public parameters param which the initial public key of an empty accumulator. This executes the following steps:

- $(\mathsf{A_L}, \mathsf{A.pk}) \leftarrow \mathsf{A.Eval}(\mathsf{param}, \mathsf{L})$: This "accumulates" the set of identities belonging to the ring L. It returns an accumulator A and its updated public key $\mathsf{A.pk}$.
- $\mathsf{w_{ID}} \leftarrow \mathsf{A.WitGen}(\mathsf{A.pk}, \mathsf{A_L}, \mathsf{L}, \mathsf{ID})$: This returns the witness $\mathsf{w_{ID}}$ for the identity of the signer which will be used to prove that his ID is included in the accumulator $\mathsf{A_L}$.
- $\pi \leftarrow \mathsf{NIZK.Prove}((m, \mathsf{A.pk}, \mathsf{A_L}, \mathsf{mpk}), (\mathsf{SID}, \mathsf{ID}, \mathsf{w_{ID}}))$ (see Definition 6) The secret witness is $w = (\mathsf{SID}, \mathsf{ID}, \mathsf{w_{ID}})$, the public statement is composed of the message m, the accumulator public key $\mathsf{A.pk}$, the accumulator build on the ring $\mathsf{A_L}$ and PKG public key mpk ($x = (m, \mathsf{A.pk}, \mathsf{A_L}, \mathsf{mpk})$). The tuple $(x, w) \in R$ if and only if the following statements stand:
  (1) $1 = \mathsf{A.Verify}(\mathsf{A.pk}, \mathsf{A_L}, \mathsf{w_{ID}}, \mathsf{ID})$: This is equivalent to prove $\mathsf{ID} \in \mathsf{L}$, so it will be proven through sub-circuit C_1 (See Fig. 3).
  (2) $1 = \mathsf{DS.Verify}(\mathsf{ID}, \mathsf{SID}, \mathsf{mpk})$: This will be proven through sub-circuit C_2 (See Fig. 3).
  Both statements will be separately proven through the sub-circuits C_1 and C_2 which are linked together with a "AND" gate to form the whole circuit C ((See Fig. 3)) and assure that they are both valid. The message to be signed, m, is embedded by integrating it to the Fiat-Shamir transform [1] [20] to generate the challenge.

[1] Fiat-Shamir transform converts an interactive protocol into a non-interactive protocol. It generates the challenge c as an outputs of H ($c = H(r, m)$), where m is the message to be signed in IDRS, instead of receiving it from the verifier.

- $\sigma \leftarrow \pi$
- **Return** σ

$\underline{0/1 \leftarrow \mathsf{IDRS.Verify}(m, \mathsf{L}, \sigma, \mathsf{mpk}, \mathsf{param})}$: This algorithm takes as inputs the message m, the list of identities L and a ring signature σ. It verifies the validity of σ by executing the following steps:

- $\pi \leftarrow \sigma$
- $(\mathsf{A_L}, \mathsf{A.pk}) \leftarrow \mathsf{A.Eval}(\mathsf{param}, \mathsf{L})$: The verifier construct the accumulator for the set of identities belonging to the ring L. It returns an accumulator $\mathsf{A_L}$ and its updated public key $\mathsf{A.pk}$.
- **Return** $\mathsf{NIZK.Verify}((m, \mathsf{A.pk}, \mathsf{A_L}, \mathsf{mpk}), \pi)$: This returns 1 if the proof π is valid for circuit C.

3.2 Security analysis

The security of our generic IDRS depends on the symmetric-key primitives used, namely a EU-CMA secure digital signature scheme DS (see Definition 4), a secure accumulator A presented in Definition 10, a cryptographic hash function H (see Definition 11) and a secure NIZK NIZK (see Definition 6).

Theorem 1 (Unforgeability). *Let* IDRS *be the construction provided in Section 3.1 with a cryptographic hash function* H*, a EUC-CMA secure digital signature scheme* DS*, a secure accumulator* A *and a secure non-interactive zero-knowledge proof system* NIZK*. Then,* IDRS *achieves unforgeability based on Definition 2.*

Proof. Let V_i be the event that $\mathcal{A}$ wins the unforgeability experiment i and (m^*, L^*, σ^*) is a forgery generated by $\mathcal{A}$. In order to prove the unforgeability of IDRS, we distinguish fours cases in the attempt of signing the message m for the user with the identity ID who belongs to the ring L and please remind that $\sigma^* = \pi^*$:

Event V_1 : $\underline{\mathsf{Exp}^{Forge}_{\mathcal{A},\mathsf{IDRS}}(\lambda) = 1}$.

Event V_1' : $\underline{V_1 \text{ happens and } ((\mathsf{ID}^*, \mathsf{SID}^*, \mathsf{w}^*_{\mathsf{ID}}) \leftarrow \mathsf{NIZK.Ext}(\mathsf{crs}, t, (m^*, \mathsf{A.pk}^*, \mathsf{A}^*_{\mathsf{L}}, \mathsf{mpk}), \pi^*) \text{ such that } ((m^*, \mathsf{A.pk}^*, \mathsf{A}^*_{\mathsf{L}}, \mathsf{mpk}), (\mathsf{ID}^*, \mathsf{SID}^*, \mathsf{w}^*_{\mathsf{ID}})) \in R}$: By the simulation extractability property of NIZK (see Definition 9), we have $\Pr[V_1'] = \Pr[V_1] - \mathsf{negl}(\lambda)$. We divide this event into two disjoint sub-events $v'_{1.1}$ and $v'_{1.2}$:

$v'_{1.1}$: $\underline{\mathsf{ID}^* \in \mathsf{L}^*}$: This event has a negligible probability to happen as this requires breaking the unforgeability of the digital signature scheme DS as $\mathsf{ID}^* \notin \mathcal{Q}$ (see Fig. 1). $\mathcal{A}$ executes the extractor (see Definition 9) on a previous signature σ, which gives him a witness SID^*. This event will happen if and only if $\mathcal{A}$ can construct his own SID without interacting with PKG, which means that he needs to break the digital signature DS used by PKG with the help of the extracted witness. However, as DS achieves EU-CMA security (see Definition 5), we can conclude that $\Pr[v'_{1.1}] \leq \mathsf{Adv}^{EU-CMA}_{\mathcal{A}} < \mathsf{negl}(\lambda)$.

$v'_{1.2}$: $\underline{\mathsf{ID}^* \notin \mathsf{L}^*}$: This event has a negligible probability to happen due to the collision-freeness property of the accumulator A. In this case, the extractor run on the $\mathcal{A}$'s forgery produces a valid witness $\mathsf{w}^*_{\mathsf{ID}}$ for an extracted identity ID^* not included in the ring L^*, which has been accumulated into $\mathsf{A}_{\mathsf{L}^*}$, i.e. $(\mathsf{A}_{\mathsf{L}^*}, \mathsf{A.pk}^*) = \mathsf{A.Eval}(\mathsf{param}, \mathsf{L}^*)$ but $\mathsf{A.Verify}(\mathsf{A.pk}^*, \mathsf{A}_{\mathsf{L}^*}, \mathsf{w}^*_{\mathsf{ID}}, \mathsf{ID}^*) = 1$. Therefore, if this event happens, we break the collision-freeness property of A (see Definition 10) Therefore, we can conclude that $\Pr[v'_{1.2}] < \mathsf{negl}(\lambda)$.

Hence $\Pr[V'_1] = \Pr[v'_{1.1}] + \Pr[v'_{1.2}] < \mathsf{negl}(\lambda)$, so we have $\Pr[V_1] < \mathsf{negl}(\lambda)$.

Overall, we can conclude $\Pr[\mathsf{Exp}^{Forge}_{\mathcal{A},\mathsf{IDRS}}(\lambda) = 1] = \Pr[V_1] < \mathsf{negl}(\lambda)$.

□

Theorem 2 (Anonymity). *Let* IDRS *be the construction provided in Section 3.1 with a cryptographic hash function* H*, a EUC-CMA secure digital signature scheme* DS*, a secure accumulator* A *and a secure non-interactive zero-knowledge proof system* NIZK*. Then,* IDRS *achieves anonymity based on Definition 3.*

Anonymity. Anonymity is directly stemmed from the zero-knowledge property of NIZK. We use a game-based approach to show that IDRS provides anonymity. Let V_i be the event that the adversary $\mathcal{A}$ wins the GAME_i.
GAME_1: The original anonymity experiment $\mathsf{Exp}^{Anom}_{\mathcal{A},\mathsf{IDRS}}$ (Fig. 3) is run by $\mathcal{A}$.
GAME_2: Same as the previous game but the proof π generated with NIZK on circuit C is replaced with the outputs of its simulator NIZK.Sim (see Definition 8). This is computationally indistinguishable from the previous game due to zero-knowledge properties of NIZK. Therefore, we can conclude that $|\Pr[V_2] - \Pr[V_1]| = \mathsf{Adv}^{\mathsf{zk}}_{\mathcal{A},\mathsf{NIZK}} < \mathsf{negl}(\lambda)$. This concludes the Anonymity proof. □

4 IDRS: applicable constructions

This section introduces possible applicable constructions of the generic IDRS presented in Section 3. Section 4.1 starts with a presentation of the practical construction of sub-circuit C_1 and discusses its security while Section 4.2 is dedicated to presenting possible constructions of sub-circuit C_2 and also discusses their security. The section ends with the summary of two possible implementations of the generic IDRS. We assume that hash functions H output a string of 2λ bits where λ is the post-quantum security level.

4.1 Sub-circuit C_1

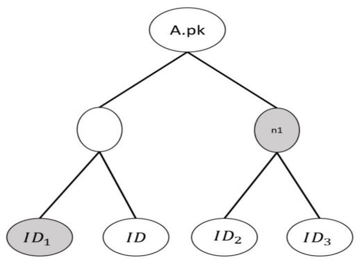
Sub-circuit C_1 (see Fig.3) aims to prove the first statement $1 = \mathsf{A.Verify}(\mathsf{A.pk}, \mathsf{A}_{\mathsf{L}}, \mathsf{w}_{\mathsf{ID}}, \mathsf{ID})$ which is equivalent to prove $\mathsf{ID} \in \mathsf{L}$ or, in other words, that the identity ID of the signer belongs to the ring L. As previously stated, we use an accumulator to prove the membership to the ring. Applying NIZK based on symmetric-key primitives requires that the verification of a valid witness, $\mathsf{A.Verify}$ algorithm, can be expressed as a one-way circuit. This leads us to circuit $\mathsf{Merkle}.C$, derived from the Merkle accumulator.

Merkle accumulator and circuit $\mathsf{Merkle}.C$: The Merkle accumulator respects Definition 10 and "accumulates" the set of identities L as a Merkle tree [30]. Each identity of the ring L is a leaf in a Merkle tree. A Merkle tree is a binary tree in which each internal node is the hash of both its children. The accumulator's public key $\mathsf{A.pk}$ is the tree root as illustrated in Fig.4a.

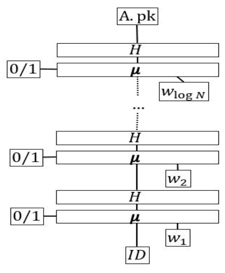
The membership proof consists of demonstrating the knowledge of the path from the leaf associated with the signer identity to the root of the tree. This can be represented as a circuit $\mathsf{Merkle}.C(w_{\mathsf{ID}}, \mathsf{ID}) = \mathsf{A.pk}$, where $\mathsf{A.pk}$ is the Merkle root and w_{ID} is the authentication path for the identity ID composed of internal nodes of the Merkle accumulator/tree and $\log N$ bits, which indicate the direction of the path. $\mathsf{Merkle}.C$ is formally presented in Algorithm 2 and is composed of $\log N$ calls of hash function H, where N is the ring size. Additionally, at each level of the Merkle accumulator, $\mathsf{Merkle}.C$ goes through a multiplexer μ which is defined as follows:

$$\mu(x, y, b) = \begin{cases} (x, y) & \text{if } b = 0, \\ (y, x) & \text{if } b = 1. \end{cases} \tag{3}$$

μ orders the inputs of H depending on the path coming from left or right in the tree. μ hides the path's direction from ID to the root $\mathsf{A.pk}$, hence ensuring anonymity to our IDRS. μ can be written as a circuit $\mu(x, y, b) = (\bar{b} \cdot x + b \cdot y, \bar{b} \cdot y + b \cdot x)$. Fig. 4a depicts an example of a Merkle accumulator and $\mathsf{Merkle}.C$: the corresponding witness for ID is $\mathsf{w}_{\mathsf{ID}} = ((\mathsf{w}_1, 1), (\mathsf{w}_2, 0))$, where the $\mathsf{Merkle}.C(\mathsf{w}_{\mathsf{ID}}, \mathsf{ID}) = H(\mu(H(\mu(\mathsf{ID}, \mathsf{w}_1, 1)), \mathsf{w}_2, 0))$. Without the multiplexer, the position of the ID would be identifiable from the path meaning that the anonymity could not be provided. $\mathsf{Merkle}.C$ is presented Algorithm 2 on pg. 16. In conclusion, $\mathsf{Merkle}.C$ plays the role of sub-circuit C_1 in our general circuit (see



(a) Example: Merkle accumulator with $N = 4$, $n1 = H(\mathsf{ID}_2, \mathsf{ID}_3)$



(b) $\mathsf{Merkle}.C$ representation

Fig. 4: Merkle Accumulator and circuit $\mathsf{Merkle}.C$

Fig. 3) to prove that the membership to the ring can be replaced in application by circuit $\mathsf{Merkle}.C$ presented in this section. The complexity of $\mathsf{Merkle}.C$ grows logarithmically with the number of identities in the ring.

Security and one-wayness $\mathsf{Merkle}.C$: As the security presented in Section 3.2 shows, we require a secure accumulator A, in our case, a secure Merkle accumulator which comes from the properties of the hash function H (see Definition 11). The collision-freeness of the accumulator comes directly from the collision resistance property of H. This means that it is computationally infeasible to construct the same accumulator from two different sets of ring members. $\mathsf{Merkle}.C$'s one-wayness ensues from the one-wayness of the hash function H (see Definition 11). This circuit is a Merkle tree with a public root. It should be computationally infeasible to recover any of the leaves from the root and this is achieved because the root is an output of the hash function H, which is a one-way function as presented in Definition 11.

4.2 Sub-circuit C_2

C_2 (see Fig.3) aims to prove the validity of signing secret key SID, namely to prove that $1 = \mathsf{DS.Verify}(\mathsf{ID}, \mathsf{SID}, \mathsf{mpk})$. Therefore, the verification procedure of the digital signature scheme is to be expressed as a one-way circuit. The current state-of-the-art of digital signatures based on symmetric-key primitives gives us two possible digital signatures: The stateless Picnic signature (see Appendix A.1) and the stateful XMSS signature (see Appendix A.2), which meet the requirements to fulfil Theorem 1 and 2.Both schemes were chosen because they are considered standard post-quantum signature (XMSS) or as alternative candidate (Picnic) by the NIST standardization process. It is important that it exist other alternatives [1]. We first present circuit $\mathsf{Picnic}.C$, which aims to prove the knowledge of a valid Picnic signature, and then circuit $\mathsf{XMSS}.C$, which aims to prove the knowledge of a valid XMSS signature.

Circuit $\mathsf{Picnic}.C$: The goal of this circuit is to prove that the signer possesses a valid Picnic signature generated by PKG. Picnic and a detailed description of its parameters are presented in Appendix A.1. More formally, the signer proves the knowledge of SID such that $1 = \mathsf{Picnic.Verify}(\mathsf{ID}, \mathsf{SID}, \mathsf{mpk})$ with the $\mathsf{NIZK.Prove}$ procedure and circuit $\mathsf{Picnic}.C$. Circuit $\mathsf{Picnic}.C$ takes as inputs $\mathsf{SID} = (\mathcal{C}, \mathcal{P}, \{\mathsf{seed}^*_j, h'_j\}_{j \notin \mathcal{C}}, \{\{state_{j,i}, r_{i,j}\}_{i \neq p_j}, com_{j,p_j}, \{\hat{z}_{j,a}\}, msgs_{j,p_j}\}_{j \in \mathcal{C}})$ and the signer identity ID. To facilitate the explanation of the circuit, we present a high-level picture of $\mathsf{Picnic}.C$ in Fig. 5 and we divide it into four sub-circuits $\mathsf{Picnic}.c_1, \mathsf{Picnic}.c_2, \mathsf{Picnic}.c_3$ and $\mathsf{Picnic}.c_4$. Each sub-circuit executes a specific step of the Picnic verification procedure (see Definition 12) as presented below:

Sub-circuit $\mathsf{Picnic}.c_1$ needs to be executed as the first part of the $\mathsf{Picnic.Verify}$ algorithm, which computes the commitment of $n-1$ parties for the τ executions of reveal in the Picnic signature. It starts for every $j \in \mathcal{C}$ and $i \neq p_j$ computes $com_{j,i} := H(state_{j,i}, r_{j,i})$: This is $(n-1)\tau$ executions of the hash function H. Then it computes the value $h_j := H(com_{j,1}, \ldots, com_{j,n})$. At end of the sub-circuit, we have τ elements (recall that $|\mathcal{C}| = \tau$) of 2λ-bits which will be used as inputs in sub-circuit $\mathsf{Picnic}.c_4$. The detailed construction of $\mathsf{Picnic}.c_1$ is presented in Algorithm 2.

Sub-circuit $\mathsf{Picnic}.c_2$ is executed on the second part of the Picnic verification procedure, namely for $j \notin \mathcal{C}$, using seed_j^* to compute h_j as the signer in the first step of $\mathsf{Picnic.Sign}$ algorithm. This creates commitments of the MPC-in-the-head execution that are not executed in the online phase (see Definition 12). According to [27], this is used to generate a binary tree similar to GGM [29] construction with n leaves $\mathsf{seed}_{j,i}$. This sub-circuit can be represented by:

– For all $j \notin \mathcal{C}$ (goes for $M - \tau$ iterations)
  1. Compute n $\mathsf{seed}_j^{(i)}$ (one per party in MPC) from seed_j^* given in the SID by expending seed_j^* into binary tree of n leaves due to the circuit $\mathsf{Tree}(\mathsf{seed}_\mathsf{j}^*, \mathsf{n}) \rightarrow \{\mathsf{seed}_j^{(i)}\}_{i=1}^n$ (see Table 2).
  2. Generate $state_{j,i}$ and $r_{j,i}$ from the circuit $\mathsf{ComputeState}(\mathsf{seed}_j^{(i)})$ by computing n (one tuple per party) calls of H (see Table 2).
  3. For $i \in [n]$: $com_{j,i} := H(state_{j,i}, r_{j,i})$:
  4. Then compute the result of the commitment based $h_j := H(com_{j,1}, \ldots, com_{j,n})$.

  Circuit $\mathsf{Picnic}.c_2$ generates $M-\tau$ outputs of 2λ bits (see detailed construction in Algorithm 1).

Sub-circuit $\mathsf{Picnic}.c_3$ aims to prove the knowledge of τ valid MPC-in-the-head computation of circuit C_{PKG} (this circuit is used to perform the Picnic signature). For each $j \in \mathcal{C}$ run an execution of circuit C_{PKG} among the parties $\{P_i\}_{i \neq p_j}$ using $\{state_{i,j}\}_{i \neq p_j}$, $\{\hat{z}_{j,a}\}$, and $msgs_{j,p_j}$; this yields $msgs_{j,p_j}$ and an output bit b. Check if $b = 1$. Then compute $h'_j = H(\{\hat{z}_{j,a}\}, msgs_{j,1}, \ldots, msgs_{j,n})$. The aim of the sub-circuit is to prove the knowledge of a valid MPC execution. As previously explained, for each "AND" gates of circuit C_{PKG} used by PKG, and for each party P_i, $\mathsf{Picnic}.c_3$ needs to perform the following MPC-in-the-head over the circuit $(\hat{z}_a \cdot [\lambda_b] \oplus \hat{z}_b \cdot [\lambda_a] \oplus [\lambda_c] \oplus [\lambda_\gamma] \oplus \hat{z}_a \cdot \hat{z}_b)$ for each $n-1$ parties for all $|\mathcal{C}| = \tau$ simulations (see Algorithm 1). After the MPC simulation, it checks that the output y is equal to mpk and computes $h'_j = H(\hat{z}_{j,a}, msgs_{j,1}, \ldots, msgs_{j,n})$.

Sub-circuit $\mathsf{Picnic}.c_4$ takes as input all elements output by the other circuits $\mathsf{Picnic}.c_1$, $\mathsf{Picnic}.c_2$ and $\mathsf{Picnic}.c_3$ and outputs 1 if the final equality $(\mathcal{C}, \mathcal{P}) \overset{?}{=} G(\mathsf{ID}, h_1, h'_1, \ldots, h_M, h'_M)$ holds.

The final circuit $\mathsf{Picnic}.C$ can be represented as $\mathsf{Picnic}.C = \mathsf{Picnic}.c_4(\mathsf{Picnic}.c_3(\mathsf{SID}), \mathsf{Picnic}.c_2(\mathsf{SID}), \mathsf{Picnic}.c_1(\mathsf{SID}), \mathsf{ID})$. The detailed circuit is presented in Algorithm 1 and illustrated in Fig. 5.

One-wayness and security of $\mathsf{Picnic}.C$: The unforgeability of Picnic have been proven in [13,28] and therefore provides the desired security according to Theorem 1 and 2. The one-wayness of $\mathsf{Picnic}.C$ ensues from the one-wayness of the four sub-circuit presented in Algorithm 1. The one-wayness of $\mathsf{Picnic}.c_1$ depends on the one-wayness of the cryptographic hash function H. Indeed, each step involves only the calls of the hash function H therefore it is computationally infeasible to invert $\mathsf{Picnic}.c_1$. $\mathsf{Picnic}.c_2$ provides also one-wayness for the same

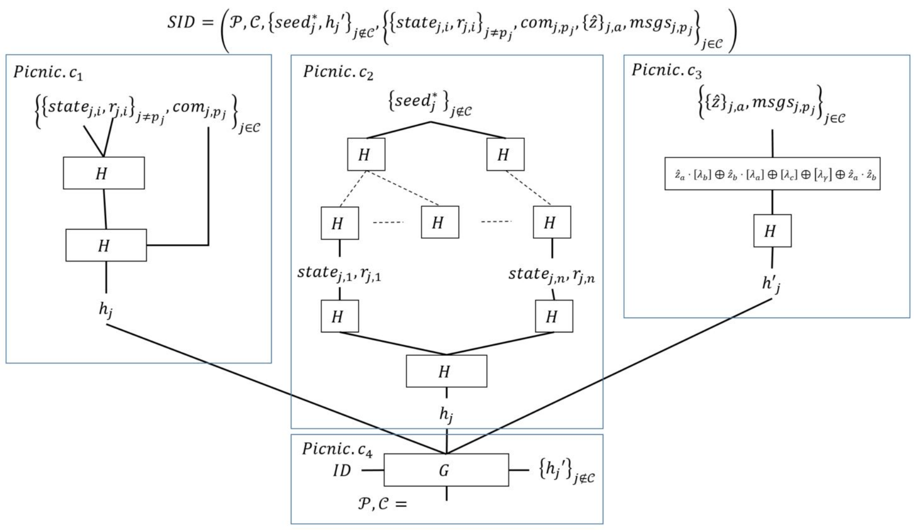

$SID = \left(\mathcal{P}, \mathcal{C}, \{seed_j^*, h_j'\}_{j \notin \mathcal{C}}, \left\{\{state_{j,i}, r_{j,i}\}_{j \neq p_j}, com_{j,p_j}, \{\hat{z}\}_{j,a}, msgs_{j,p_j}\right\}_{j \in \mathcal{C}}\right)$
$Picnic.c_1$
$\left\{\{state_{j,i}, r_{j,i}\}_{j \neq p_j}, com_{j,p_j}\right\}_{j \in \mathcal{C}}$
H
H
h_j
$Picnic.c_2$
$\{seed_j^*\}_{j \notin \mathcal{C}}$
H
$state_{j,1}, r_{j,1}$
$state_{j,n}, r_{j,n}$
h_j
$Picnic.c_3$
$\left\{\{\hat{z}\}_{j,a}, msgs_{j,p_j}\right\}_{j \in \mathcal{C}}$
$\hat{z}_a \cdot [\lambda_b] \oplus \hat{z}_b \cdot [\lambda_a] \oplus [\lambda_c] \oplus [\lambda_\gamma] \oplus \hat{z}_a \cdot \hat{z}_b$
h'_j
$Picnic.c_4$
ID
G
$\{h_j'\}_{j \notin \mathcal{C}}$
$\mathcal{P}, \mathcal{C} =$

Fig. 5: Circuit Picnic.C

Algorithm 1 Picnic.C circuit, for parameters see Appendix A.1 and Table 2

$\mathsf{SID} = (\mathcal{C}, \mathcal{P}, \{\mathsf{seed}_j^*, h_j'\}_{j \notin \mathcal{C}}, \{\{state_{j,i}\}_{i \neq p_j}, com_{j,p_j}, \{\hat{z}_{j,a}\}, msgs_{j,p_j}\}_{j \in \mathcal{C}})$

```
Picnic.$c_1$
Input: SID
Output: $\{h_j\}_{j \in \mathcal{C}}$
 1: for $j \in \mathcal{C}$ do
 2:     $h_j = H(H(state_{j,1}, r_{j,1}), \ldots, com_{j,p_j}, .., H(state_{j,n}, r_{j,n}))$
 3: end for
 4: return $\{h_j\}_{j \in \mathcal{C}}$
Picnic.$c_2$
Input: SID
Output: $\{h_j\}_{j \notin \mathcal{C}}$
 1: for $j \notin \mathcal{C}$ do
 2:     $\{\mathsf{seed}_j^{(i)}\}_{i=1}^n \leftarrow \mathsf{Tree}(\mathsf{seed}_j^*, n)$ (See Table 2)
 3:     $(state_{i,j}, r_{i,j}) \leftarrow \mathsf{ComputeState}(\mathsf{seed})$ (See Table 2)
 4:     $h_j = H(H(state_{j,1}, r_{j,1}), \ldots, H(state_{j,n}, r_{j,n}))$
 5: end for
 6: return $\{h_j\}_{j \notin \mathcal{C}}$
Picnic.$c_3$
Input: SID
Output: $b$
 1: for $j \in \mathcal{C}$ do
 2:     Execute $C$, where $C$ is the circuit used for the Picnic signature.
 3:     for all $\{P_i\}_{i \neq p_j}$ do
 4:         for $x \in [\mathsf{AND}]$ do
 5:             $(\hat{z}_a \cdot [\lambda_b] \oplus \hat{z}_b \cdot [\lambda_a] \oplus [\lambda_c] \oplus [\lambda_\gamma] \oplus \hat{z}_a \cdot \hat{z}_b)^{(j,i,x)}$ (see Fig. 5)
 6:         end for
 7:     end for
 8:     $h_j' = H(\{\hat{z}_j\}, msgs_{j,1}, \ldots, msgs_{j,n})$
 9: end for
10: return $\{h_j'\}_{j \in \mathcal{C}}$
Picnic.$c_4$
Input: $\{h_j\}_{j \in \mathcal{C}}, \{h_j'\}_{j \in C}, \{h_j, h_j'\}_{j \notin \mathcal{C}}, \mathsf{ID}$
Output: $b$
 1: $b = (\mathcal{C}, \mathcal{P}) \overset{?}{=} G(\mathsf{ID}, h_1, h_1', \ldots, h_M, h_M')$
 2: return $b$
Picnic.$C$
Input: SID, ID
Output: 0/1
 1: $b = \mathsf{Picnic}.c_4(\mathsf{Picnic}.c_1(\mathsf{SID}), \mathsf{Picnic}.c_2(\mathsf{SID}), \mathsf{Picnic}.c_3(\mathsf{SID}), \mathsf{ID})$
 2: return $b$
```

reason. Picnic.c_3 one-wayness depends on the one-wayness of the following equation $(\hat{z}_a \cdot [\lambda_b] \oplus \hat{z}_b \cdot [\lambda_a] \oplus [\lambda_c] \oplus [\lambda_\gamma] \oplus \hat{z}_a \cdot \hat{z}_b)$ on which the NIZK is executed. This six inputs equation has a 50% chance to output 0, which makes it one-way. Picnic.c_4's one-wayness follows directly from the one-wayness of hash function G. This important to notice that in Picnic, G computes the challenge (Fiat-Shamir transform) and is modelled as random oracle. Hence, producing NIZKs for calls to a random oracle which is not possible, but in practice G is initialized as an hash function which makes PicRS possible.

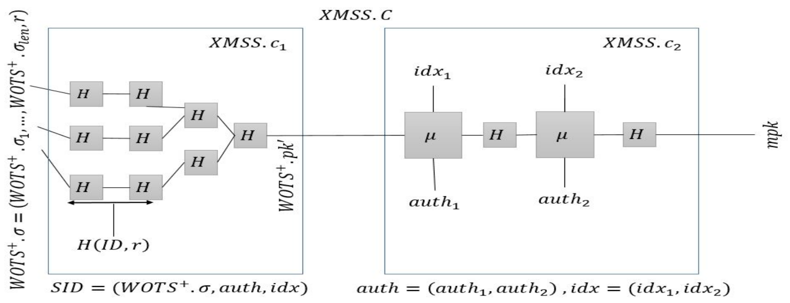

XMSS.C
XMSS.c_1
XMSS.c_2
$WOTS^+.\sigma = (WOTS^+.\sigma_1, \ldots, WOTS^+.\sigma_{len}, r)$
H
H(ID, r)
$WOTS^+.pk'$
idx_1
idx_2
μ
$auth_1$
$auth_2$
mpk
SID = (WOTS$^+$.σ, auth, idx)
auth = (auth$_1$, auth$_2$) , idx = (idx$_1$, idx$_2$)

Fig. 6: XMSS.C

XMSS.C **circuit:** We define circuit XMSS.C that will be used to prove the knowledge of a valid SID such that $1 = $ XMSS.Verify(ID, SID, mpk) (see Definition 14). We divided circuit XMSS.C into two sub-circuits XMSS.c_1 and XMSS.c_2 presented in Algorithm 2 on pg. 16. In order to facilitate the explanation, we also provide a high-level representation of XMSS.C in Fig. 8.

Sub-circuit XMSS.c_1 executes a WOTS$^+$ signatures verification procedure WOTS$^+$.pk$'$ = WOTS$^+$.Verify(ID, WOTS$^+$.σ) (see Definition 13). It outputs WOTS$^+$.pk$'$.

Sub-circuit XMSS.c_2 computes XMSS.pk$'$ from WOTS$^+$.pk$'$ (output of circuit XMSS.c_1) with the help of the authentication path auth (the grey nodes in Fig. 8). This corresponds to h evaluation of the hash function H. This takes the outputs of circuit XMSS.c_1, WOTS$^+$.pk$'$, and using the element auth as chronological outputs to the hash function H and idx (which indicates the order of the inputs of every hash function H). The idx indicates the position of the WOTS$^+$.pk in the XMSS tree. In order to provide anonymity, at each level of the path verification go through a multiplexer gate is defined in 3 (the same as the one used for the Merkle accumulator presented in Section 4.1). The multiplexer hides the position of the WOTS$^+$.pk in PKG's XMSS tree and so provides unlinkability between two signatures generated by the same signer. For example, in Fig. 8, circuit would be XMSS.c_2(WOTS$^+$.pk$'_6$, auth, idx) $=$ $H(\mu(H(\mu(H(\mu(\mathsf{WOTS}^+.\mathsf{pk}_6, \mathsf{auth}_1, 0)), n1, 1)), n2, 1))$

One-wayness and security of XMSS.C**:** XMSS.C is composed of two sub-circuits presented in Algorithm 2. The one-wayness of XMSS.C ensues directly from the unforgeability of XMSS used by PKG and of the one-wayness of the hash function. Indeed according to [25], the security of XMSS depends on the one-wayness of the hash function, it is computationally infeasible to compute the WOTS^+ public key from the XMSS tree root which is mpk. As it is computationally infeasible to invert $\mathsf{XMSS}.c_2$ from mpk and also infeasible $\mathsf{XMSS}.c_2$. The information leaked by the stateful nature of XMSS is hidden by integrating the multiplexer μ to circuit $\mathsf{XMSS}.c_2$. Because of the multiplexer, the position of the WOTS^+ signature in PKG's XMSS tree is hidden and avoids the possibility to link two signatures generated by the same signer.

Algorithm 2 $\mathsf{XMSS}.C$ circuit, for parameters see Table 2, Definition 13 and 14

$\mathsf{SID} = (\mathsf{WOTS}^+.\sigma, \mathsf{idx}, \mathsf{auth})$

```
XMSS.c_1
Input: $\mathsf{ID}, \mathsf{WOTS}^+.\sigma$
Output: $\mathsf{WOTS}^+.\mathsf{pk}'$
1: $md = (md_1, \ldots, md_{\mathsf{len}_1}) \leftarrow H(\mathsf{ID}, r)$
2: $c = (md_1, \ldots, md_{\mathsf{len}_2}) \leftarrow \sum_{i=1}^{\mathsf{len}_1}(\mathsf{Wint} - 1 - md_i)$
3: $md = (md||c)$
4: for $1 \leq i \leq \mathsf{len}$ do
5:     $\mathsf{WOTS}^+.\mathsf{pk}'_i = H^{\mathsf{Wint}-md_i}(\mathsf{WOTS}^+.\sigma_i)$
6: end for
7: $\mathsf{WOTS}^+.\mathsf{pk}' = \mathsf{MT}(\{\mathsf{WOTS}^+.\mathsf{pk}'_i\}_{i=1}^{\mathsf{len}})$ (see Table 2)
8: return $\mathsf{WOTS}^+.\mathsf{pk}'$

XMSS.c_2
Input: $\mathsf{WOTS}^+.\mathsf{pk}', \mathsf{idx}, \mathsf{auth}$
Output: $b$
1: $h_1 \leftarrow H(\mu(\mathsf{WOTS}^+.\mathsf{pk}, \mathsf{auth}_1, \mathsf{idx}_1))$ {Where $\mu$ is a multiplexer as described in Section 4.1 and presented in 3}
2: for $1 \leq i \leq h$ do
3:     $hash_i = H(\mu(h_{i-1}, \mathsf{auth}_i, \mathsf{idx}_i))$
4: end for
5: return $hash_h == \mathsf{mpk}$

XMSS.C
Input: SID
Output: 0/1
1: $b = \mathsf{XMSS}.c_2(\mathsf{XMSS}.c_1(\mathsf{SID}))$
2: return $b$

Merkle.C
Input: $\mathsf{w}_{\mathsf{ID}} = (\mathsf{w}_i, b_i)_{i=1}^{\log N}, \mathsf{A.pk}$
Output: 0/1
1: $a_1 = H(\mu(\mathsf{ID}, \mathsf{w}_1, b_1))$
2: for $i = 2$ to $i = \log N$ do
3:     $a_i = H(\mu(a_{i-1}, \mathsf{w}_i, b_i))$
4: end for
5: return $\mathsf{A.pk} = a_{\log N}$
```

4.3 Applicable post-quantum IDRSs from symmetric-key primitives

This section introduced three possible circuits: $\mathsf{Merkle}.C$, which can take the role of C_1, $\mathsf{Picnic}.C$ and $\mathsf{XMSS}.C$, which can both be implemented as C_2. From these circuits, we can implemented two different IDRSs.

Post-quantum IDRS from Picnic named PicRS**:** PicRS follows the generic construction presented in Section 3.1 and uses Picnic as digital signature DS. This means that the master public and private keys mpk and msk are set to $\mathsf{msk} = \mathsf{Picnic.sk}$ and $\mathsf{mpk} = \mathsf{Picnic.pk}$ and each user with an identity ID owns a signing secret key $\mathsf{SID} = \mathsf{Picnic}.\sigma$ such that $1 = \mathsf{Picnic.Verify}(\mathsf{ID}, \mathsf{SID}, \mathsf{mpk})$.

Table 2: Additional Functions and parameters summary

$r \leftarrow \mathsf{MT}(\mathcal{S})$	This circuit constructs a Merkle root r of binary tree from a set of leaves $\mathcal{S}$. This circuit executes H for $\lvert\mathcal{S}\rvert - 1$ times.
$\{leaf_i\}_{i=1}^{2^{leaves}} \leftarrow \mathsf{Tree}(seed, depth)$	This circuit takes as input a seed $seed$ and an integer $depth$ indicating the number of the binary tree leaves. This procedure calls $2 \cdot (leaves - 1)$ executions of H.
$(state, r) \leftarrow \mathsf{ComputeState}(seed)$	This circuit takes as input a $seed$ and outputs the tuple $(state, r)$. This circuit executes one H.
n	Number of MPC parties for Picnic signature, this indicates the number of parties that will perform the MPC-in-the-head procedure.
M	Number of MPC instanciation, which indicates the number of time the number of time MPC-in-the-head procedure needs to be executed in the Picnic
τ	number of element in the set $\mathcal{C}$, which correspond to the number of MPC execution provided in a Picnic Signature
idx	Index indicating a leave position in a XMSS/Merkle tree
AND	number of multiplication/"AND" gates
auth	authentication path in a XMSS tree
$(\mathsf{mpk}, \mathsf{msk})$	master public key & master secret key
Wint	Winternitz parameter of WOTS^+, which indicates the number of hash between WOTS^+ secret key and a public key
len	WOTS^+ parameters $\mathsf{len} = \mathsf{len}_1 + \mathsf{len}_2$ where $\mathsf{len}_1 = \left\lceil \frac{\lvert m\rvert}{\log(\mathsf{Wint})} \right\rceil$ and $\mathsf{len}_2 = \left\lfloor \frac{\log(\mathsf{len}_1(\mathsf{Wint}-1))}{\log(\mathsf{Wint})} \right\rfloor + 1$, this indicates the number of element a WOTS^+ signature is composed
ID	user's identity
SID	signing secret key of user ID
h	height of the XMSS tree
$\cdot$	the multiplication/"AND" gates in circuits
$+$	the addition/"OR" gates in circuits

In PicRS, the general circuit C (see Fig. 3) named $\mathsf{PicRS}.C$ is where the signer executes the NIZK proof. We have circuit $C = \mathsf{PicRS}.C$ where $C = \mathsf{PicRS}.C = \mathsf{Merkle}.C \cdot \mathsf{Picnic}.C$. PicRS security comes from the EU-CMA security of Picnic proven in [14], from the properties of the Merkle accumulator and from the one-wayness of circuits $\mathsf{Merkle}.C$ and $\mathsf{Picnic}.C$ discussed in Section 4.3.

Post-quantum IDRS from Picnic XMSS named XRS: The idea of the XMSS-based IDRS construction is similar to the PicRS construction but the Picnic digital signature is replaced with the XMSS signature. This means that in XRS, each signer executes NIZK on circuit $C = \mathsf{XRS}.C = \mathsf{Merkle}.C \cdot \mathsf{XMSS}.C$, where $\mathsf{XMSS}.C$ will prove the knowledge of a valid XMSS signature (i.e. $1 = \mathsf{XMSS.Verify}(\mathsf{ID}, \mathsf{SID}, \mathsf{mpk})$). XRS security comes from the EU-CMA security of XMSS proven in [25], from the properties of the Merkle accumulator and from the one-wayness of circuits $\mathsf{Merkle}.C$ and $\mathsf{XMSS}.C$ discussed in Section 4.3.

5 Evaluation

This section analyzes the signature sizes of both applicable constructions PicRS and XRS for a post-quantum security level of $\lambda = 128$ bits. We evaluate both constructions using two different non-interactive zero-knowledge proof systems (NIZK): KKW [28] and Ligero++ [9]. We chose to work with these NIZKs because, on the one hand, KKW is considered the current state-of-the-art of symmetric-key based NIZK. It has been analyzed in the QROM model, it is considered by NIST as an alternate candidate for the standardization process [1] and provides all the security properties that our construction requires. On the other hand, Ligero++ has been less studied and its post-quantum security is only assumed, but according to the literature it achieves the most competitive proof size for large circuits when compared to other works as Aurora [7] or ZK-STARK [6], which could be promising NIZK's alternative. KKW is optimized to work on binary circuits while Ligero++ is optimized for arithmetic circuits (see Table 3a). We implemented our schemes using different hash functions which are either considered as binary or as arithmetic circuits (See Table 3b). We use the standard hash function SHA3 but we also tested our constructions with other hash functions which have an optimized complexity that decreases the overall size of the circuit and of the signature. The complexity of all circuits are expressed in terms of "number of H executions" and we consider an execution of the compression function $H(x, y) = z$ with $x, y, z \in \{0, 1\}^{2\lambda}$. If H has n inputs, the number of counted execution is $n - 1$.

In the rest of this section, we discuss the complexity of circuit $\mathsf{Merkle}.C$. We then start the discussion on optimizing circuit $\mathsf{Picnic}.C$ and $\mathsf{XMSS}.C$ in order to have the shortest signature possible for PicRS and XRS. We express the complexity in terms of number of hash executions. The total complexity of our circuits can be computed with the help of Table 3b depending on the used H. We conclude the paper with a comparison between both schemes and a comparison with the current state-of-the-art of post-quantum IDRS constructed with lattices and some final recommendations.

Table 3: NIZKs and Hash functions

(a) NIZKs and their associated type of circuit. :$|C|$= circuit size/complexity, $|\mathsf{AND}|$= number of multiplication gates in circuit C

NIZK	Type	Proof Size (Asympt.)
KKW [28]	Binary	$\mathcal{O}(\lvert\mathsf{AND}\rvert)$
Ligero++ [9]	Arithmetic	$\mathcal{O}(\log^2(\lvert C\rvert))$

(b) Hash functions' complexity, A=arithmetic, B=binary

Hash	Type	Complexity	
		Mult.	Add.
SHA3	A/B	38400	422400
LowMC [3]	B	1374	30134110
Poseidon [23]	A	600	
MiMC [2]	A	1293	646
μ (Equ. 3)	A/B	1024	512

Merkle.C complexity The complexity of Merkle.C, which is used to "accumulate" all identities in the ring L increases logarithmically ($\mathcal{O}(\log N)$) with the size of the ring. This ensues from the Merkle tree structure of the accumulator (see Section 4.1). The complexity of circuit Merkle.C can be expressed as $\underline{|\mathsf{Merkle}.C| = \log N \cdot (|H| + |\mu|)}$, where $|H|$ is the complexity of the hash function H, $|\mu|$ is the complexity of the multiplexer (see Equation 3), and N the ring size. Merkle.C is implemented in both constructions, therefore this discussion is valid for both PicRS and XRS.

5.1 PicRS signature's size

The signature size of PicRS depends on the complexity of the circuit $\mathsf{PicRS}.C = \mathsf{Merkle}.C \cdot \mathsf{Picnic}.C$ (see Section 4.3). To analyze the PicRS signature size, we need to investigate circuit Picnic.C described in Section 4.2 and in Algorithm 1. We optimize the complexity circuit Picnic.C by testing different parameters, proposed in their last paper [27], for the Picnic digital signature used by PKG. We express the complexity of Picnic.C in function of executions of hash functions H and G, of number of multiplication and addition gates. We consider that an execution of H has two inputs and therefore the number of calls of H grows linearly with the number of inputs. Table 4 demonstrates that Picnic.C achieves its optimal circuit size for the Picnic scheme with the parameters $n = 3$, $\tau = 438$, and $M = 438$. This comes principally from the fact that such an instance does not need to execute sub-circuit Picnic.c_2 as $M = \tau$. Therefore, for the rest of the analysis of PicRS, we use Picnic parameters ($n = 3$, $\tau = 438 = M$) as the digital signature for PKG. Table 1 shows the signature sizes of PicRS for different ring sizes with different hash functions and NIZKs.

Table 4: Picnic.C's complexity for different Picnic parameters n, M, and τ. It shows the number of executions for the hash function H and G, the multiplication (Mult.) and addition (Add.) gates

Picnic			Circuits complexity									
n	M	τ	Picnic.c_1	Picnic.c_2	Picnic.c_3			Picnic.c_4	Picnic.C			
			H	H	H	Mult.	Add.	G	H	G	Mult.	Add.
3	438	438	1752	0	5256	2680560	244831488	1	7008	1	2680560	244831488
16	604	68	2040	1673392	4352	3121200	285077760	1	1679784	1	3121200	285077760
64	803	50	6300	2495442	12800	9639000	880387200	1	2514542	1	9639000	880387200

5.2 XRS signature's size

XRS signature size depends on the complexity of circuit XRS.C composed of sub-circuits XMSS.C and Merkle.C (see Section 4.3) on which the NIZK.Prove algorithm is executed. In this part, we focus on the specific sub-circuit XMSS.C and its internal sub-circuit XMSS.c_1, which is composed of $\mathsf{len} \cdot \mathsf{Wint} + \mathsf{len} - 1$ executions of the hash function H and the second sub-circuit XMSS.c_2 consisting of h calls to H and the multiplexer μ. Table 5 presents the complexity of circuit XMSS.C with different parameters for the XMSS scheme used by PKG to generate all secret signing keys. We used parameters proposed by XMSS's original paper [25]. Our results demonstrates that XMSS should be implemented with the parameters $\mathsf{Wint} = 4$, $\mathsf{len} = 133$, and $h = 20$ to optimize the complexity of XMSS. It is important to note that 2^h is the maximum number of SID that can be generated by PKG in this case. The detailed reason of setting $h = 20$ is presented in Section 5.3.

After optimizing the complexity of the XMSS.C, we evaluate the signature size of XRS for different group sizes and with different NIZKs and hash functions. The details of XRS features and results are presented in Table 1.

Table 5: XMSS.C circuit sizes for each sub-circuit for different Wint and len.

XMSS			Circuits complexity		
Wint	len	h	$\lvert$XMSS.$c_1\rvert$	$\lvert$XMSS.$c_2\rvert$	$\lvert$XMSS.$C\rvert$
4	133	20	$665 \cdot \lvert H\rvert$		$\mathbf{685 \cdot \lvert H\rvert + 20 \cdot \lvert\mu\rvert}$
16	67	20	$1139 \cdot \lvert H\rvert$	$20 \cdot (\lvert H\rvert + \mu)$	$1159 \cdot \lvert H\rvert + 20 \cdot \lvert\mu\rvert$
64	44	20	$2880 \cdot \lvert H\rvert$		$2900 \cdot \lvert H\rvert + 20 \cdot \lvert\mu\rvert$

5.3 PicRS vs XRS

Choice of NIZK: As presented in Table 1, our implementations using Ligero++ as a NIZK are the best options when targeting a signature size optimization. It works better for large circuits as its proof size grows logarithmically with the circuit size while KKW grows linearly with the number of multiplication gates in the circuit. For this reason, our Ligero++-based implementations can use the standard hash function SHA3 and still achieve a decent signature size while our KKW-based implementation requires a specifically designed hash function (LowMC) to be competitive. To the best of our knowledge, KKW is optimized for binary circuits similar to LowMC while Poseidon and MiMC are arithmetic circuits that work over larger finite fields, which makes them unpractical for KKW. However, KKW has been submitted to the NIST standardization process [1] and therefore gives stronger security guarantees than the other Ligero++. Ligero++ was only published recently and its post-quantum security has been assumed as it relies on known post-quantum paradigms, but it has not been proven.

Signature Size: Table 1 summarizes the performance of both schemes in terms of signature size. XRS clearly outperforms PicRS due to the lower complexity of circuit XMSS.C (see Table 5) used in XRS compared to Picnic.C (see Table 4) implemented in PicRS. Even if in theory both signature sizes should increase logarithmically with the ring size, we observe that both schemes provide a nearly constant signature size because the circuit complexity of PicRS.C and XRS.C depends mainly on Picnic.C and XMSS.C. Picnic.C represent 99% of PicRS.C complexity and XMSS.C 95% of XRS.C's complexity for the largest ring $N = 2^{20}$. Because of Ligero++ proof size that increases only logarithmically with the circuit size while KKW's one increases linearly with the number of multiplication gates, PicRS and XRS implemented with Ligero++ have a signature size "more constant" than the ones implemented with KKW (see Table 1). A possible optimization for XRS could be replacing the WOTS$^+$ scheme by a few-time signature scheme named FORS [8] in the XMSS scheme used by PKG, which should further reduce the signature size for XRS. However, this assumption requires a formal security analysis.

PKG characteristic: In PicRS, PKG enjoys the stateless feature of Picnic and therefore does not need to update his secret key after a signature as it is required for XRS. The main advantage of PicRS over XRS is that PKG can theoretically generate an infinite number of signing secret keys, so can handle an infinite number of users, while XRS is limited to 2^h users. Our implementation showed in Table 1 sets $h = 20$ due to the computation complexity of generating a XMSS tree (e. g. IDRS.Setup algorithm) which is grows exponentially with h.

Comparison with lattice-based IDRS: Table 1 also highlights the competitiveness of XRS and PicRS when it comes to signature sizes compared with lattice-based IDRS. It is important to highlight that none of the lattice-based works gave a precise signature size. We estimated the signature size of Zhao et.al. [36] work according to their formula. We fixed their parameters to $n = 1000$ (n is their security parameters for the short integer solution problem (SIS)), $q = 2^{40}$, $w = 3$ and $k = 41$. Their estimated size is presented in Table 1. Regardless of the difference of the actual signature size, XRS and PicRS enjoy a nearly constant signature while all current state-of-the-art of lattice constructions [36], [12],[33], [34] have a signature size increasing linearly with the ring size N. Therefore, all of our implementations shown in Table 1 are more suitable for large rings than the lattice-based IDRS. Investigating the traditional state-of-the-art lattice-based ring signature designed by Esgin et al. [19] could be a promising future work to improve the competitiveness of lattice-based IDRS.

Final recommendations and conclusion: Table 1 shows that XRS implemented with Ligero++ is our most promising construction when an optimized signature size is desired. It achieves a competitive signature size with hash functions Poseidon, MiMC and even with the standard hash SHA3. ZK-STARK [6] and Aurora [7] could be a alternative to Ligero++, they both achieve a proof size slightly larger than Ligero++, but are still competitive. As illustrated in Table 1, XRS outperforms PicRS with a smaller signature size and a smaller SID that comes from the difference in size between XMSS and Picnic. It is also important to highlight that our possible constructions have been evaluated theoretically and it would be interesting to investigate the applicability with an implementation. According to KKW [28] Ligero++[9] original papers the circuit's complexity influences running and the memory complexity of the signing and verification algorithms. This increases the advantage of XRS over PicRS. Therefore, our final recommendation would be to use XRS implemented either with Poseidon and Ligero++ to achieve the best compromise between proof size and security or with KKW combined with LowMC to ensure post-quantum security.

References

1. G. Alagic, J. Alperin-Sheriff, D. Apon, D. Cooper, Q. Dang, J. Kelsey, Y.-K. Liu, C. Miller, D. Moody, R. Peralta, et al. Status report on the second round of the nist post-quantum cryptography standardization process. *NIST, Tech. Rep., July*, 2020.
2. M. Albrecht, L. Grassi, C. Rechberger, A. Roy, and T. Tiessen. Mimc: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In *ASIACRYPT*, pages 191–219. Springer, 2016.
3. M. R. Albrecht, C. Rechberger, T. Schneider, T. Tiessen, and M. Zohner. Ciphers for mpc and fhe. In *EUROCRYPT*, pages 430–454. Springer, 2015.
4. M. H. Au, J. K. Liu, T. H. Yuen, and D. S. Wong. Id-based ring signature scheme secure in the standard model. In *International Workshop on Security*, pages 1–16. Springer, 2006.
5. M. Backes, L. Hanzlik, and J. Schneider-Bensch. Membership privacy for fully dynamic group signatures. In *ACM CCS 2019*, pages 2181–2198, 2019.
6. E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev. Scalable, transparent, and post-quantum secure computational integrity. *IACR Cryptol. ePrint Arch.*, 2018:46, 2018.
7. E. Ben-Sasson, A. Chiesa, M. Riabzev, N. Spooner, M. Virza, and N. P. Ward. Aurora: Transparent succinct arguments for r1cs. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 103–128. Springer, 2019.
8. D. J. Bernstein, A. Hülsing, S. Kölbl, R. Niederhagen, J. Rijneveld, and P. Schwabe. The sphincs+ signature framework. In *ACM CCS 2019*, pages 2129–2146, 2019.
9. R. Bhadauria, Z. Fang, C. Hazay, M. Venkitasubramaniam, T. Xie, and Y. Zhang. Ligero++: a new optimized sublinear iop. In *ACM CSS*, pages 2025–2038, 2020.
10. D. Boneh, S. Eskandarian, and B. Fisch. Post-quantum epid signatures from symmetric primitives. In *Cryptographers' Track at the RSA Conference*, pages 251–271. Springer, 2019.
11. J. Buchmann, E. Dahmen, and A. Hülsing. Xmss-a practical forward secure signature scheme based on minimal security assumptions. In *International Workshop on Post-Quantum Cryptography*, pages 117–129. Springer, 2011.

12. C. Cao, L. You, and G. Hu. Fuzzy identity-based ring signature from lattices. *Security and Communication Networks*, 2021, 2021.
13. M. Chase, D. Derler, S. Goldfeder, C. Orlandi, S. Ramacher, C. Rechberger, D. Slamanig, and G. Zaverucha. Post-quantum zero-knowledge and signatures from symmetric-key primitives. In *ACM CCS*, pages 1825–1842. ACM, 2017.
14. M. Chase, D. Derler, S. Goldfeder, C. Orlandi, S. Ramacher, C. Rechberger, D. Slamanig, and G. Zaverucha. Picnic: Post quantum signatures. https://github.com/microsoft/Picnic, 2020.
15. S. S. Chow, R. W. Lui, L. C. Hui, and S.-M. Yiu. Identity based ring signature: Why, how and what next. In *European Public Key Infrastructure Workshop*, pages 144–161. Springer, 2005.
16. S. S. Chow, S.-M. Yiu, and L. C. Hui. Efficient identity based ring signature. In *ACNS*, pages 499–512. Springer, 2005.
17. C. D. de Saint Guilhem, L. De Meyer, E. Orsini, and N. P. Smart. Bbq: using aes in picnic signatures. In *International Conference on Selected Areas in Cryptography*, pages 669–692. Springer, 2019.
18. D. Derler, C. Hanser, and D. Slamanig. Revisiting cryptographic accumulators, additional properties and relations to other primitives. In *Cryptographers' track at the rsa conference*, pages 127–144. Springer, 2015.
19. M. F. Esgin, R. K. Zhao, R. Steinfeld, J. K. Liu, and D. Liu. Matrict: Efficient, scalable and post-quantum blockchain confidential transactions protocol. In *ACM CCS 2019*, pages 567–584, 2019.
20. A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology—CRYPTO'86*, pages 186–194. Springer, 1986.
21. C. Gamage, B. Gras, B. Crispo, and A. S. Tanenbaum. An identity-based ring signature scheme with enhanced privacy. In *2006 Securecomm and Workshops*, pages 1–5. IEEE, 2006.
22. I. Giacomelli, J. Madsen, and C. Orlandi. Zkboo: Faster zero-knowledge for boolean circuits. In *USENIX Security*, pages 1069–1083, 2016.
23. L. Grassi, D. Khovratovich, C. Rechberger, A. Roy, and M. Schofnegger. Poseidon: A new hash function for zero-knowledge proof systems. In *Usenix Security*, 2021.
24. J. Herranz and G. Sáez. New identity-based ring signature schemes. In *International Conference on Information and Communications Security*, pages 27–39. Springer, 2004.
25. A. Hülsing, D. Butin, S. Gazdag, and A. Mohaisen. Xmss: Extended hash-based signatures. In *Crypto Forum Research Group Internet-Draft.(2015). draft-irtf-cfrg-xmss-hash-based-signatures-01*, 2015.
26. Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Zero-knowledge proofs from secure multiparty computation. *SIAM Journal on Computing*, 39(3):1121–1152, 2009.
27. D. Kales and G. Zaverucha. Improving the performance of the picnic signature scheme. *IACR Cryptol. ePrint Arch.*, 2020:427, 2020.
28. J. Katz, V. Kolesnikov, and X. Wang. Improved non-interactive zero knowledge with applications to post-quantum signatures. In *ACM CCS 2018*, pages 525–537, 2018.
29. B. Lynn. Ggm constructions. https://crypto.stanford.edu/pbc/notes/crypto/ggm.html, 2020.
30. R. C. Merkle. A certified digital signature. In *Conference on the Theory and Application of Cryptology*, pages 218–238. Springer, 1989.

31. R. L. Rivest, A. Shamir, and Y. Tauman. How to leak a secret. In *ASIACRYPT*, pages 552–565. Springer, 2001.
32. A. Shamir. Identity-based cryptosystems and signature schemes. In *Workshop on the theory and application of cryptographic techniques*, pages 47–53. Springer, 1984.
33. J. Wang. Identity-based ring signature from lattice basis delegation. *Beijing: Tsinghua University*, 2008.
34. B. Wei, Y. Du, H. Zhang, F. Zhang, H. Tian, and C. Gao. Identity based threshold ring signature from lattices. In *International Conference on Network and System Security*, pages 233–245. Springer, 2015.
35. F. Zhang and K. Kim. Id-based blind signature and ring signature from pairings. In *ASIACRYPT*, pages 533–547. Springer, 2002.
36. G. Zhao and M. Tian. A simpler construction of identity-based ring signatures from lattices. In *International Conference on Provable Security*, pages 277–291. Springer, 2018.

A Definitions

Definition 4 (Digital signature). *A digital signature scheme* DS *is composed by the following algorithms:*

$(\mathsf{DS.pk}, \mathsf{DS.sk}) \leftarrow \mathsf{DS.KeyGen}(1^\lambda)$*: This takes as input the security parameter* λ *and outputs the keypair* $(\mathsf{DS.pk},\ \mathsf{DS.sk})$.

$\mathsf{DS}.\sigma \leftarrow \mathsf{DS.Sign}(m, \mathsf{DS.sk})$*: This takes as inputs a message* m *to be signed and a secret key* $\mathsf{DS.sk}$*. It outputs a valid digital signature* $\mathsf{DS}.\sigma$*.*

$0/1 \leftarrow \mathsf{DS.Verify}(m, \mathsf{DS}.\sigma, \mathsf{DS.pk})$*: This takes as inputs the signed message* m*, a digital signature* $\mathsf{DS}.\sigma$*, and the public key* $\mathsf{DS.pk}$*. It outputs* 1 *if* $\mathsf{DS}.\sigma$ *is valid and* 0*, otherwise.*

Definition 5 (EU-CMA). *A digital signature scheme* DS *reaches existential unforgeability under adaptive chosen message attacks (EU-CMA) security if and only if for a security parameter* λ *and an integer* $q = polynomial(\lambda)$ *the advantage* $\mathsf{Adv}_{\mathcal{A}}^{EU\text{-}CMA}$ *of an adversary* $\mathcal{A}$ *satisfies*

$$\mathsf{Adv}_{\mathcal{A}}^{EU\text{-}CMA}(\lambda) = \Pr[\mathsf{Exp}_{\mathcal{A},\mathsf{DS}}^{EU\text{-}CMA}(\lambda) = 1] < \mathsf{negl}(\lambda), \tag{4}$$

where $\mathsf{Exp}_{\mathcal{A},\mathsf{DS}}^{EU\text{-}CMA}(\lambda)$ *is defined in Fig. 7.*

```
$\mathsf{Exp}_{\mathcal{A},\mathsf{DS}}^{EU\text{-}CMA}(\lambda)$ :
$(\mathsf{DS.pk}, \mathsf{DS.sk}) \leftarrow \mathsf{DS.KeyGen}(1^\lambda)$
$(m', \mathsf{DS}.\sigma) \leftarrow \mathcal{A}^{\mathsf{DS.Sign(DS.sk)}}(\mathsf{DS.pk})$
The set $\mathcal{Q}$ contains $q$ pairs $(m', \mathsf{DS}.\sigma)$ generated by $\mathsf{DS.Sign}$.
**if** $1 \leftarrow \mathsf{DS.Verify}(\mathsf{m'}, \mathsf{DS}.\sigma', \mathsf{DS.pk})$ and $m' \notin \mathcal{Q}$
    **then return** $1$
```

Fig. 7: EU-CMA security experiment

Definition 6 (Non-interactive zero-knowledge proof system (NIZK)). *Non-interactive zero-knowledge proof system (NIZK) [5] aims to prove that a public statement x and a private witness w belong to a defined relation R (i.e. $(x, w) \in R$). We also let $\mathcal{L}_R = \{x | \exists w$ s.t. $(x, w) \in R\}$. A* NIZK *consists of the following three algorithms:*

- $\mathsf{crs} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$: *This generates the common reference string* crs *from the security parameters λ.*
- $\pi \leftarrow \mathsf{NIZK.Prove}(\mathsf{crs}, x, w)$*: This generates a proof π for the common reference string* crs*, the statement x and the witness w that satisfies the relation R (to be more specific, we have $(x, w) \in R$).*
- $0/1 \leftarrow \mathsf{NIZK.Verify}(\mathsf{crs}, x, \pi)$*: This returns 1 if the proof π based on the common reference string* crs *and the public statement x is valid, 0 otherwise.*

Remark 1. In this paper, we omit the use of the common reference string crs.

Definition 7 (Completeness). *A proof system* NIZK *achieves completeness if for an adversary $\mathcal{A}$, the advantage* $\mathsf{Adv}^{\mathsf{Comp}}_{\mathcal{A},\mathsf{NIZK}}(\lambda)$ *is*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{Comp}}_{\mathcal{A},\mathsf{NIZK}}(\lambda) = \Pr[&\mathsf{crs} \leftarrow \mathsf{NIZK.Setup}(1^\lambda); (x, \pi) \xleftarrow{\$} \mathcal{A}(\mathsf{crs}) : \\ &\mathsf{NIZK.Verify}(\mathsf{crs}, x, \pi) = 0 \wedge x \in \mathcal{L}_R] \leq \mathsf{negl}(\lambda).\end{aligned} \tag{5}$$

Definition 8 (Zero-knowledge). *The verifier learns nothing but the validity of the statement. A* NIZK *is said to be zero-knowledge if the advanatage of $\mathcal{A}$* $\mathsf{Adv}^{\mathsf{zk}}_{\mathcal{A},\mathsf{NIZK}}(\lambda)$ *is:*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{zk}}_{\mathcal{A},\mathsf{NIZK}}(\lambda) = |\Pr[&\mathsf{crs} \leftarrow \mathsf{NIZK.Setup}(1^\lambda) : \mathcal{A}^{\mathsf{NIZK.Prove}(\mathsf{crs},x,w)} = 1] - \\ &\Pr[(\mathsf{crs}^*, \pi^*) \leftarrow \mathsf{NIZK.Sim}(1^\lambda, x) : \mathcal{A}^{(x,\mathsf{crs}^*,\pi^*)} = 1]| \leq \mathsf{negl}(\lambda),\end{aligned} \tag{6}$$

where $(\mathsf{crs}^*, \pi^*) \leftarrow \mathsf{NIZK.Sim}(1^\lambda, x)$ *is a simulator that takes as input the security parameter λ and the statement x and outputs a common reference string* crs^* *and a simulated proof π^*.*

Definition 9 (Simulation extractability). *A proof system* NIZK *satisfies simulation extractability if there is algorithms* S *and* $\mathsf{NIZK.Sim}$ *satisfying the zero-knowledge definition and an extractor $\mathcal{E}$ such that:*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{SimE}}_{\mathcal{A},\mathsf{NIZK}}(\lambda) = \Pr[&(x, \pi) \leftarrow \mathcal{A}^{\mathsf{S},\mathsf{NIZK.Sim}}(1^\lambda); w \leftarrow \mathsf{NIZK.Ext}(\mathsf{crs}, t, x, \pi) : \\ &\mathsf{NIZK.Verify}(x, \pi) = 1 \wedge (x, \pi) \notin \mathcal{Q} \wedge (x, w) \notin R] \leq \mathsf{negl}(\lambda)\end{aligned} \tag{7}$$

where $\mathcal{E} = ((\mathsf{crs}, t) \leftarrow \mathsf{NIZK.ExtGen}(1^\lambda, t), w \leftarrow \mathsf{NIZK.Ext}(\mathsf{crs}, t, x, \pi)$ *is a extractor, $\pi \leftarrow$* $\mathsf{S}(t, x)$*, t is a state and $\mathcal{Q}$ is the list of queries done by $\mathcal{A}$ to* $\mathsf{NIZK.Sim}$.

Definition 10 (Accumulator). *An accumulator [10]* A *is defined by the following algorithms:*

$\mathsf{A.pk} \leftarrow \mathsf{A.Gen}(1^\lambda)$: *The setup algorithm takes as input the security parameter* λ *and outputs the public key* $\mathsf{A.pk}$.

$(\mathsf{A}_\mathcal{X}, \mathsf{A.pk}) \leftarrow \mathsf{A.Eval}(\mathsf{A.pk}, \mathcal{X})$: *The evaluation algorithm takes as inputs the public key* $\mathsf{A.pk}$ *and the set* $\mathcal{X}$ *and outputs the accumulator* $\mathsf{A}_\mathcal{X}$ *and an updated public key* $\mathsf{A.pk}$ *for the new accumulated set* $\mathcal{X}$.

$\mathsf{w}_{x_i} / \perp \leftarrow \mathsf{A.WitGen}(\mathsf{A.pk}, \mathsf{A}_\mathcal{X}, \mathcal{X}, x_i)$: *The witness generation algorithm takes as inputs the public key* $\mathsf{A.pk}$*, the accumulator* $\mathsf{A}_\mathcal{X}$*, the set* $\mathcal{X}$*, and an element* x_i*. It outputs the witness* w_{x_i} *if* $x_i \in \mathcal{X}$ *and* $\perp$ *otherwise.*

$0/1 \leftarrow \mathsf{A.Verify}(\mathsf{A.pk}, \mathsf{A}_\mathcal{X}, \mathsf{w}_{x_i}, x_i)$: *The verification algorithm takes as inputs the public key* $\mathsf{A.pk}$*, the accumulator* $\mathsf{A}_\mathcal{X}$*, the witness* w_{x_i}*, and the element* x_i*. It outputs* 1 *if* w_{x_i} *is a valid witness for* $x_i \in \mathcal{X}$ *and* 0 *otherwise.*

Similar to [10], we assume that an accumulator A *achieves correctness, soundness and collision-freeness as follows:*

- *Correctness: For all* $x \in \mathcal{X}$ *with* $\mathsf{A.pk} \leftarrow \mathsf{A.Gen}(1^\lambda)$*,* $(\mathsf{A}_\mathcal{X}, \mathsf{A.pk}) \leftarrow \mathsf{A.Eval}(\mathsf{A.pk}, \mathcal{X})$*, and* $\mathsf{w}_x \leftarrow \mathsf{A.WitGen}(\mathsf{A.pk}, \mathsf{A}_\mathcal{X}, \mathcal{X}, x)$*, we have that* $\Pr[0 = \mathsf{A.Verify}(\mathsf{A.pk}, \mathsf{A}_\mathcal{X}, \mathsf{w}_x, x)] < \mathsf{negl}(\lambda)$.
- *Soundness: For all* $x \notin \mathcal{X}$ *with* $\mathsf{A.pk} \leftarrow \mathsf{A.Gen}(1^\lambda)$*,* $(\mathsf{A}_\mathcal{X}, \mathsf{A.pk}) \leftarrow \mathsf{A.Eval}(\mathsf{A.pk}, \mathcal{X})$*, and* $\mathsf{w}_x \leftarrow \mathsf{A.WitGen}(\mathsf{A.pk}, \mathsf{A}_\mathcal{X}, \mathcal{X}, x)$*, we have that* $\Pr[1 = \mathsf{A.Verify}(\mathsf{A.pk}, \mathsf{A}_\mathcal{X}, \mathsf{w}_x, x)] < \mathsf{negl}(\lambda)$.
- *Collision-freeness: An accumulator achieves collision-freeness if for an adversary* $\mathcal{A}$ *we have:*

$$\begin{aligned} &\Pr[\mathsf{A.Verify}(\mathsf{A.pk}^*, \mathsf{A}_{\mathcal{X}^*}, \mathsf{w}^*_{x_i}, x^*_i) = 1 \wedge x^*_i \notin \mathcal{X}^* | \mathsf{A.pk} \leftarrow \mathsf{A.Gen}(1^\lambda), \\ &(\mathcal{X}^*, \mathsf{w}^*_{x_i}, x^*_i) \leftarrow \mathcal{A}(\mathsf{A.pk}), (\mathsf{A}_{\mathcal{X}^*}, \mathsf{A.pk}^*) \leftarrow \mathsf{A.Eval}(\mathsf{A.pk}, \mathcal{X}^*)] < \mathsf{negl}(\lambda). \end{aligned} \tag{8}$$

Definition 11 (Cryptographic Hash function). *A cryptographic hash function*

$$H : \{0,1\}^* \to \{0,1\}^{2\lambda} \tag{9}$$

takes as input a message a *of any length and outputs the hash value* b *of length* 2λ *bits. A cryptographic hash function fulfills the three following properties:*

- *Pre-image resistance (one-wayness): given a hash value* b*, where* $b = H(a)$ *for a uniformly random* $a \in \{0,1\}^*$ *it is computationally infeasible (in polynomial-time) to find* a *such that* $b = H(a)$.
- *Second Pre-image Resistance: knowing a pair* $(a_0, H(a_0))$ *for a uniformly random* $a_0 \in \{0,1\}^*$ *it is computationally infeasible to find another input* $a_1 \in \{0,1\}^*$ *such that* $H(a_1) = H(a_0)$.
- *Collision Resistance: it is computationally infeasible to find two different inputs* a_0 *and* a_1 *such that* $a_0 \neq a_1$ *resulting with the same hash value* $b = H(a_0) = H(a_1)$

A.1 The Picnic signature

Picnic applies the MPC-in-the-head paradigm described in [26], and [13]. In MPC-in-the-head, the prover simulates a MPC computation between n parties of a circuit C composed of "AND" and "XOR" logical gates. The n parties, shares the "XORED" bit b, which is denoted by $[b] = \bigoplus_{i=1}^{n} b_i$. The proof will reveal the views of $n-1$ parties, named opened parties, and the messages broadcasted by the last party, unopened party. The verifier will execute the MPC-in-the-head protocol based on those views and the messages of the unopened party. It is important to know that it requires $(n-1)-$privacy which means knowing the view of $n-1$ parties does not reveal anything about the last party. In Picnic, each wire a has a value z_a and a party P_i owns a mask λ_a for the wire a which allow to compute the masked value $\hat{z}_a = z_a \oplus \lambda_a$. A party P_i owns a seed $\mathsf{seed}^{(i)} \in \{0,1\}^{2\lambda}$, which is used to compute his mask $\lambda_{a,i}$ for each single wire in C. The party P_n owns an additional value aux_n, which is a correction value of AND bits, where AND is the number of "AND" gates in C. In order to achieve acceptable soundness, the proof contains multiple parallel executions of the MPC-in-the-head simulation. Picnic is defined following parameters:

- n: the number of MPC parties,
- M: the number of parallel executions of the MPC-in-the-head protocol,
- AND: the number of "AND" gates in C,
- τ: number of online executions revealed in the proof ($\tau \leq M$), and
- $M-\tau$: number of preprocessing computations revealed.

In the prepropressing, each party generates AND triples $\{([\lambda_a], [\lambda_b], [\lambda_c])\}$ such that $[\lambda_c] = [\lambda_a] \cdot [\lambda_b]$.
All λ_a, λ_b and λ_c can be generated from a party-owned seed $\mathsf{seed}^{(i)}$. Moreover, the correction value aux_n is part of the $state_n$ of the P_n in order to ensure the validity of . This part is done M times. Then the online part starts. The parties then evaluate circuit C in a gate-by-gate fashion in chronological. The only type of gates that requires communication is the "AND" gates and the following procedure is done:

- Each party locally computes and broadcasts $[s] = \hat{z}_a \cdot [\lambda_b] \oplus \hat{z}_b \cdot [\lambda_a] \oplus [\lambda_c] \oplus [\lambda_\gamma]$, where $\lambda_\gamma = \lambda_a \oplus \lambda_b$. Then computes $\hat{z}_\gamma = \hat{z}_a \cdot \hat{z}_b \oplus s$.

At the end of circuit C, each party broadcast its share and then the outputs is reconstructed. The online part is done only over τ randomly selected prepossessed executions. The Picnic signature size can be computed from the following formula [28,17]:

$$4\lambda + \tau \log(M/\tau) 6\lambda + \tau(2\lambda \log(n) + 2\mathsf{AND} + |w| + 4\lambda). \tag{10}$$

Definition 12 (Picnic). *The Picnic signature follows Definition 4 and therefore is defined by the following algorithms:*

$(\mathsf{Picnic.pk}, \mathsf{Picnic.sk}) \leftarrow \mathsf{Picnic.KeyGen}(1^\lambda)$*: The key generation algorithm initiates the circuit C. It samples w, computes $y \leftarrow C(w)$, and lets* $\mathsf{Picnic.sk} = w$ *and* $\mathsf{Picnic.pk} = y$.

$\mathsf{Picnic}.\sigma \leftarrow \mathsf{Picnic.Sign}(m, \mathsf{Picnic.sk})$*: This is simply an execution of* $\mathsf{KKW.Prove}$ *algorithm and uses the message m to generate the challenge with the help of the Fiat-Shamir transform. The signing algorithm has the following steps:*

1. *for $j \in [M]$*
   - *(a) Sets* $\mathsf{seed}_j^* \xleftarrow{\$} \{0,1\}^{2\lambda}$*, to generate* $\mathsf{seed}_j^{(1)}, \ldots, \mathsf{seed}_j^{(n)}$ *(one for each party) and $r_{j,1}, \ldots, r_{j,n}$, set* $\mathsf{seed}_j^{(i)} = state_{j,i}$ *for $1 \leq i \leq n-1$ and* $state_{j,n} = \mathsf{seed}_j^{(n)} || aux_j$ *where aux_j is the correction value for the j-th iteration and is computed as follow.*
     - *For $y \in [\mathsf{AND}]$*
       - *i. The party P_i uses* $\mathsf{seed}_j^{(i)}$ *to sample $\lambda^{(i)}_{a_{j,y}}, \lambda^{(i)}_{b_{j,y}}$ and $\lambda^{(i)}_{c_{j,y}}$*
       - *ii. Compute $[\lambda_{a_{j,y}}]$ and $[\lambda_{b_{j,y}}]$,*
       - *iii. Compute $\delta_{j,y} = [\lambda_{a_{j,y}}] \cdot [\lambda_{b_{j,y}}] - [\lambda_{c_{j,y}}]$. Let $aux_j^{(n)} = (\delta_{j,y})_{y \in [\mathsf{AND}]}$.*
   - *(b) For $i \in [n]$: $com_{j,i} = H(state_{j,i}, r_{j,i})$*
   - *(c) $h_j = H(com_{j,1}, \ldots, com_{j,n})$.*
   - *(d) This procedure starts by computing the masked $\hat{z}_{j,a}$ from the witness $w =$* $\mathsf{Picnic.sk}$ *and the λ_a computed in step (1a) inputs for each wire a. Then compute the Circuit C by proceeding through the gates in order. For each party P_i the messages $msgs_{j,i}$ are generated for the iteration j, $msgs_{j,p_j}$ are actually the communication for each "AND" gate.*
   - *(e) $h'_j = H(\hat{z}_{j,a}, msgs_{j,1}, \ldots, msgs_{j,n})$.*
2. *$(\mathcal{C}, \mathcal{P}) = G(m, h_1, h'_1, \ldots, h_M, h'_M)$, with $\mathcal{C} \subseteq [M]$ composed τ elements and $\mathcal{P}$ representing the set of unopened parties, one per parallel execution. This last part is the only difference between KKW and Picnic. In KKW, the challenge is generated thanks the preprocessing step, while in Picnic, the message m to be signed is included in the Fiat-Shamir transform. G is a hash function with an output size of $\tau \cdot \log M + \tau \log N$.*
3. ***Return*** $\mathsf{Picnic}.\sigma = (\mathcal{C}, \mathcal{P}, \{\mathsf{seed}_j^*, h'_j\}_{j \notin \mathcal{C}}, \{\{state_{j,i}, r_{j,i}\}_{i \neq p_j}, com_{j,p_j}, \{\hat{z}_{j,a}\}, msgs_{j,p_j}\}_{j \in \mathcal{C}})$.

$0/1 \leftarrow \mathsf{Picnic.Verify}(m, \mathsf{Picnic}.\sigma, \mathsf{Picnic.pk})$*: This algorithm executes the* KKW *verification procedure. It return 1 if the result of* $\mathsf{KKW.Verify}$ *is 1 and 0 otherwise, where* $\mathsf{Picnic}.\sigma = (\mathcal{C}, \mathcal{P}, \{\mathsf{seed}_j^*, h'_j\}_{j \notin \mathcal{C}}, \{\{state_{j,i}, r_{j,i}\}_{i \neq p_j}, com_{j,p_j}, \{\hat{z}_{j,a}\}, msgs_{j,p_j}\}_{j \in \mathcal{C}})$ *this algorithms does:*

1. *For $j \in \mathcal{C}$ and $i \in [n]$ with $i \neq p_j$, set $com_{j,i} := H(state_{j,i}, r_{j,i})$ and then compute the value $h_j := H(com_{j,1}, \ldots, com_{j,n})$.*
2. *For $j \notin \mathcal{C}$ used* seed_j^* *to compute h_j as the signer would (step 1 (a) to (c) of* $\mathsf{Picnic.Sign}$ *algorithm)*
3. *For each $j \in \mathcal{C}$ run an execution of C among the parties $\{P_i\}_{i \neq p_j}$ using $\{state_{i,j}\}_{i \neq p_j}$, $\{\hat{z}_{j,a}\}$, and $msgs_{j,p_j}$; this yields $msgs_{j,p_j}$*

and an output bit b. Check that $b = 1$. *Then compute* $h'_j = H(\{\hat{z}_{j,a}\}, msgs_{j,1}, \ldots, msgs_{j,n})$.

4. *Return* $(\mathcal{C}, \mathcal{P} == G(m, h_1, h'_1, \ldots, h_M, h'_M))$.

A.2 The stateful digital signature XMSS

The XMSS [25] is a digital signature relying on Merkle tree [30] and properties of a cryptographic hash function H. XMSS achieves EU-CMA (see Definition 5) as presented in [25]. A signer can produce at most 2^h signatures. As XMSS is stateful as the signer needs to update his key after each signature. At the heart of XMSS, there is a one-time signature scheme named WOTS^+, in this work we only consider WOTS^+ in XMSS and therefore, WOTS^+ signature never needs to be verified.

Definition 13 (WOTS^+). WOTS^+ *is a one-time signature scheme with the parameters* Wint *and* $\mathsf{len} = \mathsf{len}_1 + \mathsf{len}_2$ *where* $\mathsf{len}_1 = \left\lceil \frac{|m|}{\log(\mathsf{Wint})} \right\rceil$ *and* $\mathsf{len}_2 = \left\lfloor \frac{\log(\mathsf{len}_1(\mathsf{Wint}-1))}{\log(\mathsf{Wint})} \right\rfloor + 1$. WOTS^+ *defined by the following algorithms:*

$(\mathsf{WOTS}^+.\mathsf{pk}, \mathsf{WOTS}^+.\mathsf{sk}) \leftarrow \mathsf{WOTS}^+.\mathsf{KeyGen}(1^\lambda)$*: This algorithm computes for* $i \in [\mathsf{len}]$*:* $\mathsf{WOTS}^+.\mathsf{sk}_i \xleftarrow{\$} \{0,1\}^{2\lambda}$ *;* $\mathsf{WOTS}^+.\mathsf{pk}_i \leftarrow H^{\mathsf{Wint}}(\mathsf{WOTS}^+.\mathsf{sk}_i)$*, then* $\mathsf{WOTS}^+.\mathsf{pk} = \mathsf{MT}(\{\mathsf{WOTS}^+.\mathsf{pk}_i\}_{i=1}^{\mathsf{len}})$ *(this reduces* $\mathsf{WOTS}^+.\mathsf{pk}$ *in a* 2λ *bits elements, see Table 2).*

$\mathsf{WOTS}^+.\sigma \leftarrow \mathsf{WOTS}^+.\mathsf{Sign}(m, \mathsf{WOTS}^+.\mathsf{sk})$*: This generates a signature* $\mathsf{WOTS}^+.\sigma = (\mathsf{WOTS}^+.\sigma_1, \ldots, \mathsf{WOTS}^+.\sigma_{\mathsf{len}}, r) = (H^{md_1}(\mathsf{WOTS}^+.\mathsf{sk}_1), \ldots, H^{md_{\mathsf{len}}}(\mathsf{WOTS}^+.\mathsf{sk}_{\mathsf{len}}), r)$. *Where md is compute with the following step:* $r \xleftarrow{\$} \{0,1\}^\lambda$, $md = (md_1, \ldots, md_{\mathsf{len}_1}) \leftarrow H(\mathsf{ID}, r)$, $c = (md_1, \ldots, md_{\mathsf{len}_2}) \leftarrow \sum_{i=1}^{\mathsf{len}_1}(\mathsf{Wint} - 1 - md_i)$ *and* $md = md||c$

$\mathsf{WOTS}^+.\mathsf{pk}' \leftarrow \mathsf{WOTS}^+.\mathsf{Verify}(m, \mathsf{WOTS}^+.\sigma, \mathsf{WOTS}^+.\mathsf{pk})$*: In the context of the XMSS,* WOTS^+ *signature is not verified. This procedure outputs the corresponding key* $\mathsf{WOTS}^+.\mathsf{pk}'$ *based on the signature* $\mathsf{WOTS}^+.\sigma$ *with the following procedure.* $(\mathsf{WOTS}^+.\mathsf{pk}'_1, \ldots, \mathsf{WOTS}^+.\mathsf{pk}'_{len})$
$= (H^{\mathsf{Wint}-md_1}(\mathsf{WOTS}^+.\sigma_1), \ldots, H^{\mathsf{Wint}-md_{\mathsf{len}}}(\mathsf{WOTS}^+.\sigma_{\mathsf{len}}))$.
Where Where md is compute with the following step: $md = (md_1, \ldots, md_{\mathsf{len}_1}) \leftarrow H(\mathsf{ID}, r)$, $c = (md_1, \ldots, md_{\mathsf{len}_2}) \leftarrow \sum_{i=1}^{\mathsf{len}_1}(\mathsf{Wint} - 1 - md_i)$ *and* $md = md||c$
Then it returns $\mathsf{WOTS}^+.\mathsf{pk}' = \mathsf{MT}(\{\mathsf{WOTS}^+.\mathsf{pk}'_i\}_{i=1}^{\mathsf{len}})$

In XMSS, each leaf is a WOTS^+ key pair which will be used by the signer in chronological order. An XMSS signature is composed of a WOTS^+ signature and an authentication path which are the internal nodes of a binary tree. The authentication path allows one to compute the tree root from the WOTS^+ signature. For simplification reasons, we omit the use of bitmask in the XMSS tree construction.

Definition 14 (XMSS). XMSS *[11] is a stateful digital signature defined by the following three polynomial-time algorithms:*

$(\mathsf{XMSS.pk}, \mathsf{XMSS.sk}) \leftarrow \mathsf{XMSS.KeyGen}(1^\lambda)$*: This starts by generating* 2^h WOTS^+ *key pairs. Then a binary tree is built using the* $\mathsf{WOTS}^+.\mathsf{pk}$ *as leaves and outputs the root* $\mathsf{XMSS.pk}$*. An example is presented in see Fig. 8.*

$\mathsf{XMSS}.\sigma \leftarrow \mathsf{XMSS.Sign}(m, \mathsf{XMSS.sk})$*: This takes a message* m *to be signed, the secret key* $\mathsf{XMSS.sk}$ *and returns* $\mathsf{XMSS}.\sigma$ *following the procedure:*

1. $\mathsf{WOTS}^+.\sigma \leftarrow \mathsf{WOTS}^+.\mathsf{Sign}(m, \mathsf{WOTS}^+.\mathsf{sk}_{\mathsf{idx}})$,
2. $\mathsf{XMSS}.\sigma = (\mathsf{WOTS}^+.\sigma, \mathsf{idx}, \mathsf{auth})$*, where the index* idx *indicates the position of the* WOTS^+ *keys in the tree and* auth *is the authentication path from the* WOTS^+ *to the root (the grey nodes in Fig. 8), and*
3. $\mathsf{idx} = \mathsf{idx} + 1$*. Update the index* idx *such that the next* WOTS^+ *key pair will be used for the next signature.*

$1/0 \leftarrow \mathsf{XMSS.Verify}(m, \mathsf{XMSS}.\sigma, \mathsf{XMSS.pk})$*: This takes as input a message* m*, a signature* $\mathsf{XMSS}.\sigma$ *and the public key* $\mathsf{XMSS.pk}$*. It outputs* 1 *if* $\mathsf{XMSS}.\sigma$ *is valid and* 0 *otherwise. This procedure executes the following steps:*

1. $\mathsf{WOTS}^+.\mathsf{pk}' = \mathsf{WOTS}^+.\mathsf{Verify}(m, \mathsf{WOTS}^+.\sigma)$ *(see Definition 13),*
2. *Compute* $\mathsf{XMSS.pk}'$ *from* $\mathsf{WOTS}^+.\mathsf{pk}'$ *with the help of the authentication path* auth *(the grey nodes in Fig. 8). This corresponds to* h *evaluations of the hash function* H*, and*
3. *Return* $\mathsf{XMSS.pk} == \mathsf{XMSS.pk}'$*.*

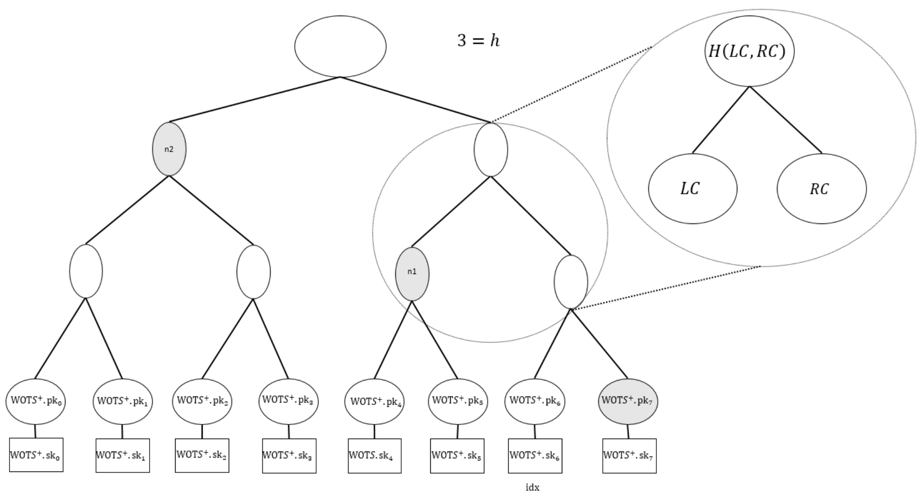


Fig. 8: XMSS tree