New Code-Based Cryptographic Accumulator and Fully Dynamic Group Signature [1]

Edoukou Berenger Ayebie & El Mamoun Souidi
emsouidi@gmail.com

Mohammed V University in Rabat

[1] accepted by *Designs, Codes and Cryptography, 2022*

Accumulator

– A cryptographic accumulator allows accumulating a finite set of elements into a single value that is called Accumulated Value (a public value).

Accumulator

– A cryptographic accumulator allows accumulating a finite set of elements into a single value that is called Accumulated Value (a public value).

– For each element, it is possible to generate a short membership proof called Witness, which attest the belonging of the element to the set.

Accumulator

– A cryptographic accumulator allows accumulating a finite set of elements into a single value that is called Accumulated Value (a public value).

– For each element, it is possible to generate a short membership proof called Witness, which attest the belonging of the element to the set.

– For a cryptographic accumulator to be secure, it should not be possible to find a valid witness for an element not belonging to the set.

Accumulator

– A cryptographic accumulator allows accumulating a finite set of elements into a single value that is called Accumulated Value (a public value).

– For each element, it is possible to generate a short membership proof called Witness, which attest the belonging of the element to the set.

– For a cryptographic accumulator to be secure, it should not be possible to find a valid witness for an element not belonging to the set.

– This notion was first introduced by Benaloh and De Mare in 1993.

Example

To accumulate a finite set $X = \{p_i\}$ of big prime integers,

Example

To accumulate a finite set $X = \{p_i\}$ of big prime integers, we compute the product

$$p = \prod_{i \in X} p_i$$

Example

To accumulate a finite set $X = \{p_i\}$ of big prime integers, we compute the product

$$p = \prod_{i \in X} p_i$$

What is the witness ?

Example

To accumulate a finite set $X = \{p_i\}$ of big prime integers, we compute the product

$$p = \prod_{i \in X} p_i$$

What is the witness ? $q_i = \frac{p}{p_i}$.

Example

To accumulate a finite set $X = \{p_i\}$ of big prime integers, we compute the product

$$p = \prod_{i \in X} p_i$$

What is the witness ? $q_i = \frac{p}{p_i}$.

Then everyone can check the membership of p_i by checking if $p_i q_i = p$ which is an easy computation.

Example

To accumulate a finite set $X = \{p_i\}$ of big prime integers, we compute the product

$$p = \prod_{i \in X} p_i$$

What is the witness ? $q_i = \frac{p}{p_i}$.

Then everyone can check the membership of p_i by checking if $p_i q_i = p$ which is an easy computation.

But the factorization of p to extract p_i is "hard" for usual computers but not for the hypothetical quantum computer.

Accumulator Applications

Cryptographic accumulators are very useful in many privacy-preserving technologies like

Accumulator Applications

Cryptographic accumulators are very useful in many privacy-preserving technologies like
anonymous credential systems

Accumulator Applications

Cryptographic accumulators are very useful in many privacy-preserving technologies like
anonymous credential systems ,
group signatures:

Accumulator Applications

Cryptographic accumulators are very useful in many privacy-preserving technologies like
anonymous credential systems ,
group signatures: a method for allowing a member of a group to anonymously sign a message on behalf of the group. Require a group manager who can reveal the original signer in the event of disputes.

Accumulator Applications

Cryptographic accumulators are very useful in many privacy-preserving technologies like
anonymous credential systems ,
group signatures: a method for allowing a member of a group to anonymously sign a message on behalf of the group. Require a group manager who can reveal the original signer in the event of disputes.
efficient time-stamping: (horodatage)

Accumulator Applications

Cryptographic accumulators are very useful in many privacy-preserving technologies like

anonymous credential systems ,

group signatures: a method for allowing a member of a group to anonymously sign a message on behalf of the group. Require a group manager who can reveal the original signer in the event of disputes.

efficient time-stamping: (horodatage)

sanitizable signatures:

Accumulator Applications

Cryptographic accumulators are very useful in many privacy-preserving technologies like

anonymous credential systems ,

group signatures: a method for allowing a member of a group to anonymously sign a message on behalf of the group. Require a group manager who can reveal the original signer in the event of disputes.

efficient time-stamping: (horodatage)

sanitizable signatures: allow designated parties (the sanitizers) to apply arbitrary modifications to some restricted parts of signed messages.

Accumulator Applications

Cryptographic accumulators are very useful in many privacy-preserving technologies like
anonymous credential systems ,
group signatures: a method for allowing a member of a group to anonymously sign a message on behalf of the group. Require a group manager who can reveal the original signer in the event of disputes.
efficient time-stamping: (horodatage)
sanitizable signatures: allow designated parties (the sanitizers) to apply arbitrary modifications to some restricted parts of signed messages.

Accumulator Applications

ring signatures:

Accumulator Applications

ring signatures: type of digital signature that can be performed by any member of a set of users that each have keys. It is then not possible to determine the person in the group who has created the signature. Excludes the requirement of a group manager.

Accumulator Applications

ring signatures: type of digital signature that can be performed by any member of a set of users that each have keys. It is then not possible to determine the person in the group who has created the signature. Excludes the requirement of a group manager.

redactable signatures:

Accumulator Applications

ring signatures: type of digital signature that can be performed by any member of a set of users that each have keys. It is then not possible to determine the person in the group who has created the signature. Excludes the requirement of a group manager.

redactable signatures: allows removing parts of a signed message without invalidating the signature. An example of use: when disclosing documents, governments and enterprises must remove privacy information concerning individuals.

identity-based encryption: allows any party to generate a public key from a known identity value.

Accumulator Applications

ring signatures: type of digital signature that can be performed by any member of a set of users that each have keys. It is then not possible to determine the person in the group who has created the signature. Excludes the requirement of a group manager.

redactable signatures: allows removing parts of a signed message without invalidating the signature. An example of use: when disclosing documents, governments and enterprises must remove privacy information concerning individuals.

identity-based encryption: allows any party to generate a public key from a known identity value.

Zerocoin, *etc...*

Some Related Works

In dynamic group signature, the group manager has the possibility to add or remove users. When a group signature scheme allows to add and remove users it is called a fully dynamic group signature ($FDGS$) scheme.

Some Related Works

In dynamic group signature, the group manager has the possibility to add or remove users. When a group signature scheme allows to add and remove users it is called a fully dynamic group signature ($FDGS$) scheme.

The first post-quantum $FDGS$ scheme is from lattice-based cryptography. In code-based cryptography, they are two dynamic group signature schemes that allow the group manager to only add users.

Some Related Works

The first scheme introduced by Benaloh is said to be static: it is not possible to add or remove elements from the set of accumulated values.

Some Related Works

The first scheme introduced by Benaloh is said to be static: it is not possible to add or remove elements from the set of accumulated values.

In 2002 Camenisch and Lysyanskaya proposed a first dynamic accumulator based on RSA assumptions which allow elements to be added or removed from the set of accumulated values, but they do not have an efficient non-membership proof.

Some Related Works

Later, many other constructions have been proposed But all of the mentioned work cannot allow providing non-membership proofs.

Some Related Works

Later, many other constructions have been proposed But all of the mentioned work cannot allow providing non-membership proofs.

Li *et al.* proposed in the construction of an accumulator which supports both membership and non-membership proofs.

Some Related Works

Later, many other constructions have been proposed But all of the mentioned work cannot allow providing non-membership proofs.

Li *et al.* proposed in the construction of an accumulator which supports both membership and non-membership proofs.

However, the proposed scheme requires a trusted accumulator manager and does not completely fit the accumulator definition given firstly by Benaloh *et al.*

Some Related Works

Later, many other constructions have been proposed But all of the mentioned work cannot allow providing non-membership proofs.

Li *et al.* proposed in the construction of an accumulator which supports both membership and non-membership proofs.

However, the proposed scheme requires a trusted accumulator manager and does not completely fit the accumulator definition given firstly by Benaloh *et al.*

To fix this lack, Camacho *et al.* proposed a strong universal accumulator which does not need a trusted accumulator manager.

Some Related Works

All of the schemes proposed above have their security based on the discrete logarithm or the factorization problems that are not resistant to the quantum computer.

Some Related Works

All of the schemes proposed above have their security based on the discrete logarithm or the factorization problems that are not resistant to the quantum computer.

This fact among others has enabled the emergency of post-quantum cryptography, like lattice-based cryptography, code-based cryptography, multivariate cryptography or isogeny-based cryptography.

Some Related Works

In 2015, based on lattice assumptions, a first direct construction of post-quantum cryptographic accumulator has been proposed by Jhanwar and Safavi-Naini in

Some Related Works

In 2015, based on lattice assumptions, a first direct construction of post-quantum cryptographic accumulator has been proposed by Jhanwar and Safavi-Naini in

One year later, using Merkle-Tree, Libert *et al.* presented another lattice-based cryptographic accumulator.

Some Related Works

In 2015, based on lattice assumptions, a first direct construction of post-quantum cryptographic accumulator has been proposed by Jhanwar and Safavi-Naini in

One year later, using Merkle-Tree, Libert *et al.* presented another lattice-based cryptographic accumulator.

In 2018, Ling *et al.* proposed a dynamic lattice-based accumulator scheme

Some Related Works

In 2015, based on lattice assumptions, a first direct construction of post-quantum cryptographic accumulator has been proposed by Jhanwar and Safavi-Naini in

One year later, using Merkle-Tree, Libert *et al.* presented another lattice-based cryptographic accumulator.

In 2018, Ling *et al.* proposed a dynamic lattice-based accumulator scheme

In 2019, using Merkle-Tree, Nguyen *et al.* proposed a first code-based cryptographic accumulator. Their proposed scheme uses random codes which require a big storage capacity and no implementation of their scheme is provided.

Our Proposal

In this paper, we propose a new code-based cryptographic accumulator.

Our Proposal

In this paper, we propose a new code-based cryptographic accumulator.

It proceeds by computing a Merkle tree using a collision-resistant family of hash functions based on a $[2k, k]$-code over the binary field $\mathbb{F}_2$.

Our Proposal

In this paper, we propose a new code-based cryptographic accumulator.

It proceeds by computing a Merkle tree using a collision-resistant family of hash functions based on a $[2k, k]$-code over the binary field $\mathbb{F}_2$.

Moreover, this technique uses double circulant codes that provide compact keys and is securely based on the hardness of the syndrome decoding problem.

Our Proposal

In this paper, we propose a new code-based cryptographic accumulator.

It proceeds by computing a Merkle tree using a collision-resistant family of hash functions based on a $[2k, k]$-code over the binary field $\mathbb{F}_2$.

Moreover, this technique uses double circulant codes that provide compact keys and is securely based on the hardness of the syndrome decoding problem.

Furthermore, to support the feasibility of our scheme, we give an implementation of our cryptographic accumulator which is also, to the best of our knowledge, is the first direct implementation of post-quantum cryptographic accumulators.

Our Implementation

Practically, for a set of 16384 elements, to have an 80 bits security level, our scheme needs only 347 bits to stock the public key, 347 bits to stock the accumulated value, 347 bits to stock the auxiliary value and 4872 bits to stock the witness.

Our Implementation

Practically, for a set of 16384 elements, to have an 80 bits security level, our scheme needs only 347 bits to stock the public key, 347 bits to stock the accumulated value, 347 bits to stock the auxiliary value and 4872 bits to stock the witness.

This implementation takes less than one second to accumulate all the elements, to generate the witness of any element and to verify a witness.

Our Proposal Implementation

Since this implementation is the first direct one from post-quantum cryptographic accumulator, we cannot provide a comparison to other ones.

Our Proposal Implementation

Since this implementation is the first direct one from post-quantum cryptographic accumulator, we cannot provide a comparison to other ones.

Our construction allows deducing a fully dynamic code-based group signature scheme which is logarithmic complexity on the group size.

We recall from the definition and security requirements for a static cryptographic accumulator.

Static Accumulator Definition

- $\mathsf{Gen}(1^\lambda)$: this algorithm takes a security parameter λ and returns a public key $\mathsf{p_k}$.

Static Accumulator Definition

- $\mathsf{Gen}(1^\lambda)$: this algorithm takes a security parameter λ and returns a public key $\mathsf{p_k}$.
- $\mathsf{Eval}(\mathsf{p_k}, \mathcal{X})$: this algorithm takes public key $\mathsf{p_k}$ and a set $\mathcal{X}$ to be accumulated in $A_\mathcal{X}$ and returns an accumulator $A_\mathcal{X}$ together with some auxiliary information ξ.

Static Accumulator Definition

- $\mathsf{Gen}(1^\lambda)$: this algorithm takes a security parameter λ and returns a public key $\mathsf{p_k}$.
- $\mathsf{Eval}(\mathsf{p_k},\mathcal{X})$: this algorithm takes public key $\mathsf{p_k}$ and a set $\mathcal{X}$ to be accumulated in $A_\mathcal{X}$ and returns an accumulator $A_\mathcal{X}$ together with some auxiliary information ξ.
- $\mathsf{WitCreate}(\mathsf{p_k},\mathcal{X}, A_\mathcal{X}, \xi, x_i)$: this algorithm takes public key $\mathsf{p_k}$, the accumulated set $\mathcal{X}$, an accumulator $A_\mathcal{X}$, auxiliary information ξ and a value x_i. It returns error, if $x_i \notin \mathcal{X}$, and a witness W_{x_i} for x_i, otherwise.

Static Accumulator Definition

- $\mathsf{Gen}(1^\lambda)$: this algorithm takes a security parameter λ and returns a public key $\mathsf{p_k}$.
- $\mathsf{Eval}(\mathsf{p_k},\mathcal{X})$: this algorithm takes public key $\mathsf{p_k}$ and a set $\mathcal{X}$ to be accumulated in $A_\mathcal{X}$ and returns an accumulator $A_\mathcal{X}$ together with some auxiliary information ξ.
- $\mathsf{WitCreate}(\mathsf{p_k},\mathcal{X}, A_\mathcal{X}, \xi, x_i)$: this algorithm takes public key $\mathsf{p_k}$, the accumulated set $\mathcal{X}$, an accumulator $A_\mathcal{X}$, auxiliary information ξ and a value x_i. It returns error, if $x_i \notin \mathcal{X}$, and a witness W_{x_i} for x_i, otherwise.
- $\mathsf{TVerify}(\mathsf{p_k}, A_\mathcal{X}, W_{x_i}, x_i)$: this algorithm takes a public key $\mathsf{p_k}$, an accumulator $A_\mathcal{X}$, a witness W_{x_i} and a value x_i. It returns *true* if W_{x_i} is a witness for $x_i \in \mathcal{X}$ and *false* otherwise.

(Non)-Universal Cryptographic Accumulator

A universal cryptographic accumulator allows proving both membership and non-membership.

(Non)-Universal Cryptographic Accumulator

A universal cryptographic accumulator allows proving both membership and non-membership.

A non-universal cryptographic accumulator allows proving either membership or non-membership.

(Non)-Universal Cryptographic Accumulator

A universal cryptographic accumulator allows proving both membership and non-membership.

A non-universal cryptographic accumulator allows proving either membership or non-membership.

A non-universal cryptographic accumulator is secure if it is correct and collision free and satisfying furthermore the indistinguishability

Correctness

Correctness: This notion guarantees that for all honestly generated keys, all honestly computed accumulators and witnesses, the TVerify() algorithm will always return true.

Collision Freeness

Collision Freeness: Informally, this notion states that it is neither feasible to find a witness for a non-accumulated value.

Collision Freeness

Collision Freeness: Informally, this notion states that it is neither feasible to find a witness for a non-accumulated value.

Notation: $\mathcal{O}^E$ represents an oracle for the algorithm $\mathsf{Eval}()$ and an adversary $\mathcal{A}$ that have access to an oracle $\mathcal{O}$ is noted $\mathcal{A}^{\mathcal{O}}$.

Collision Freeness

Collision Freeness: Informally, this notion states that it is neither feasible to find a witness for a non-accumulated value.

Notation: $\mathcal{O}^E$ represents an oracle for the algorithm $\mathsf{Eval}()$ and an adversary $\mathcal{A}$ that have access to an oracle $\mathcal{O}$ is noted $\mathcal{A}^{\mathcal{O}}$.

An adversary is allowed to query them an arbitrary number of times. The oracle $\mathcal{O}^W$ allows the adversary to obtain membership witnesses for members.

Collision Freeness

Definition (Collision Freeness)

Let α be the output of Algorithm 1. A static and non-universal cryptographic accumulator is a collision-free, if for all probabilistic polynomial-time (PPT) algorithms of an adversary $\mathcal{A}$, the probability α of Collision Experiment (Algorithm 1) is negligible.

Algorithm 1 Collision Experiment

Require: The security parameter λ.
Ensure: The probability α of collision.

1. $\mathsf{p_k} \leftarrow \mathsf{Gen}(1^{\lambda})$, (where $\leftarrow$ is the affectation symbol.)
2. $\mathscr{O} \leftarrow \{\mathscr{O}^E, \mathscr{O}^W\}$
3. The adversary $\mathscr{A}^{\mathscr{O}}$ with the assess to $\mathsf{p_k}$ generates the tuples $(W_{x_i}, x_i, \mathscr{X})$

$$\alpha = Pr[\mathsf{TVerify}(\mathsf{p_k}, A_{\mathscr{X}}, W_{x_i}, x_i) = true\ and\ x_i \notin \mathscr{X}]$$

Indistinguishability (Indiscernabilité)

The notion of indistinguishability is that, given two different sets of values and an accumulated value, an adversary cannot decide from which accumulator comes the accumulated value. We give a formal definition of indistinguishability for static and non-universal cryptographic accumulator.

Indistinguishability

Definition (Indistinguishability)

A static and non-universal cryptographic accumulator is indistinguishable, if for all probabilistic polynomial-time (PPT) algorithms of an adversary $\mathcal{A}$, the probability β of Indistinguishability Experiment given by Algorithm 2 is close to $\frac{1}{2}$.

Algorithm 2 Indistinguishability Experiment

Require: The security parameter λ.
Ensure: The probability β of Indistinguishability Experiment.

1. $\mathsf{p_k} \leftarrow \mathsf{Gen}(1^\lambda)$
2. pick a random $b \in \{0, 1\}$
3. $(\mathscr{X}_0, \mathscr{X}_1, state) \leftarrow \mathscr{A}(\mathsf{p_k})$, the adversary $\mathscr{A}$ with the access to $\mathsf{p_k}$, generates a tuple $(\mathscr{X}_0, \mathscr{X}_1, state)$
4. $(A_{\mathscr{X}_b}, \xi) \leftarrow \mathsf{Eval}(\mathsf{p_k}, \mathscr{X}_b)$
5. $\mathscr{O} \leftarrow \{\mathscr{O}^E, \mathscr{O}^W\}$
6. The adversary $\mathscr{A}^{\mathscr{O}}$ with the access to $(\mathsf{p_k}, A_{\mathscr{X}_b}, state)$, generates a bit $g \in \{0, 1\}$

$$\beta = Pr[b = g]$$

Fully Dynamic Group Signature I

A fully dynamic group signature scheme ($FDGS$) is a tuple of polynomial-time algorithm described as follow:

1. $\mathsf{GSetup}(1^\lambda)$: this algorithm generates global public parameters of the system $\mathsf{p_p}$.

Fully Dynamic Group Signature II

2. $\langle \mathsf{GKgen}_{\mathsf{GM}}(\mathsf{p_p}), \mathsf{GKgen}_{\mathsf{TM}}(\mathsf{p_p})\rangle$: in this interactive protocol run by the group manager GM and the tracing manager, algorithm $\mathsf{GKgen}_{\mathsf{GM}}$ outputs a manager public and private keys $(\mathsf{m_{pk}}, \mathsf{m_{sk}})$. At the same time, GM initializes the group information $\mathfrak{I}$ and the registration table $\mathbf{r}$. The algorithm $\mathsf{GKgen}_{\mathsf{TM}}$ outputs the tracing public and private keys $(\mathsf{t_{pk}}, \mathsf{t_{sk}})$. At the end the group public key is $\mathsf{g_{pk}} = (\mathsf{p_p}, \mathsf{m_{pk}}, \mathsf{t_{pk}})$.

Fully Dynamic Group Signature III

3. $\mathsf{UKgen}(\mathsf{p_p})$: this algorithm generates a user public and private keys $(\mathsf{u_{pk}}, \mathsf{u_{sk}})$.
4. $\langle \mathsf{Join}(\mathfrak{I}_\tau, \mathsf{g_{pk}}, \mathsf{u_{pk}}, \mathsf{u_{sk}}); \mathsf{Issue}(\mathfrak{I}_\tau, \mathsf{m_{sk}}, \mathsf{u_{id}}) \rangle$: in this interactive protocol run by the user and GM, the algorithm Join is used to add a user as group member and to store private group signing key $\mathsf{g_{sk}}[\mathsf{u_{id}}]$. While the algorithm Issue is used to store registration information in the table $\mathbf{r}$ with index $\mathsf{u_{id}}$.

Fully Dynamic Group Signature IV

5. $\mathsf{GUpdate}(\mathsf{g_{pk}}, \mathsf{m_{sk}}, \mathfrak{I}_\tau, \mathbb{S}, \mathbf{r})$: in this algorithm run by the GM, the group information is updated while advancing the epoch. Given $\mathsf{g_{pk}}$,$\mathsf{m_{sk}}$, $\mathfrak{I}_\tau$, registration table $\mathbf{r}$, a set $\mathbb{S}$ of active users to be removed from the group, GM computes new group information $\mathfrak{I}_{\tau_{new}}$ and may update the table $\mathbf{r}$.
6. $\mathsf{IsActive}(\mathfrak{I}_\tau, \mathbf{r}, \mathsf{u_{id}})$: this algorithm outputs 1 if the user is active at epoch τ and 0 otherwise.
7. $\mathsf{Sign}(\mathsf{g_{pk}}, \mathsf{g_{sk}}(\mathsf{u_{id}}), \mathfrak{I}_\tau, M)$: this algorithm outputs a group signature Σ on message M by user $\mathsf{u_{id}}$.

Fully Dynamic Group Signature V

8. $\mathsf{Verify}(\mathsf{g_{pk}}, \mathfrak{I}_\tau, M, \Sigma)$: this algorithm checks the validity of the signature Σ on message M at epoch τ.
9. $\mathsf{Trace}(\mathsf{g_{pk}}, \mathsf{t_{sk}}, \mathfrak{I}_\tau, \mathbf{r}, M, \Sigma)$: this is an algorithm run by TM. Given the inputs, TM returns an identity $\mathsf{u_{id}}$ of a group member who signed the message and a proof indicating this tracing result Π_{trace}.
10. $\mathsf{Judge}(\mathsf{g_{pk}}, \mathsf{u_{id}}, \mathfrak{I}_\tau, \Pi_{trace}, M, \Sigma)$: this algorithm checks the validity of Π_{trace} outputted by the Trace algorithm.

The Proposed Accumulator

This section is devoted to describe our code-based cryptographic accumulator.

The Proposed Accumulator

This section is devoted to describe our code-based cryptographic accumulator.

First, we recall the Double Circulant Regular Syndrome Decoding (DCRSD) problem and 2-Double Circulant Regular Null Syndrom Decoding (2-DCRNSD) problem.

The Proposed Accumulator

This section is devoted to describe our code-based cryptographic accumulator.

First, we recall the Double Circulant Regular Syndrome Decoding (DCRSD) problem and 2-Double Circulant Regular Null Syndrom Decoding (2-DCRNSD) problem.

Augot *et al.* showed that DCRSD and 2-DCRNSD problems are hard.

The Proposed Accumulator

This section is devoted to describe our code-based cryptographic accumulator.

First, we recall the Double Circulant Regular Syndrome Decoding (DCRSD) problem and 2-Double Circulant Regular Null Syndrom Decoding (2-DCRNSD) problem.

Augot *et al.* showed that DCRSD and 2-DCRNSD problems are hard.

Second, we construct a family of collision-resistant hash functions. And third, we use these hash functions to propose a code-based cryptographic accumulator.

Code-Based Difficult Problems

Problem (1. Double Circulant Regular Syndrome Decoding (DCRSD) problem)

Given an integer w, a vector $s \in \mathbb{F}_2^k$ and a double circulant matrix $H \in \mathcal{M}_{k\times n}(\mathbb{F}_2)$ split into w sub-blocs H_i of size $k \times \frac{n}{w}$, find w columns of H, one per block H_i, such that s is the sum of the w columns.

Code-Based Difficult Problems

Problem (1. Double Circulant Regular Syndrome Decoding (DCRSD) problem)

Given an integer w, a vector $s \in \mathbb{F}_2^k$ and a double circulant matrix $H \in \mathcal{M}_{k\times n}(\mathbb{F}_2)$ split into w sub-blocs H_i of size $k \times \frac{n}{w}$, find w columns of H, one per block H_i, such that s is the sum of the w columns.

Problem (2. Two-Double Circulant Regular Null Syndrom Decoding (2-DCRNSD) problem)

Given a double circulant matrix $H \in \mathcal{M}_{k\times n}(\mathbb{F}_2)$ split into w sub-blocs H_i of size $k \times \frac{n}{w}$, find $2w'$ columns (with $0 < w' \leq w$) of H, 0 or 2 per block H_i, such that the sum of the $2w'$ columns is null.

Code-Based Difficult Problems

Problem (3. Decisional Double Circulant Regular Syndrome Decoding (DDCRSD) problem)

Given a pair $(A, v) \in \mathcal{M}_{k\times n}(\mathbb{F}_2) \times \mathbb{F}_2^k$, *distinguish whether* (A, v) *is a uniformly random pair over* $\mathcal{M}_{k\times n}(\mathbb{F}_2) \times \mathbb{F}_2^k$ *or it is obtained by randomly choosing* $A \in \mathcal{M}_{k\times n}(\mathbb{F}_2)$ *and outputting* (A, v), *such that after splitting* A *into* w *sub-blocs* A_i *of size* $k \times \frac{n}{w}$ *and choosing* w *columns of* A, *one per block* A_i, v *is the sum of the* w *columns.*

Code-Based Difficult Problems

Problem (3. Decisional Double Circulant Regular Syndrome Decoding (DDCRSD) problem)

Given a pair $(A, v) \in \mathcal{M}_{k\times n}(\mathbb{F}_2) \times \mathbb{F}_2^k$*, distinguish whether* (A, v) *is a uniformly random pair over* $\mathcal{M}_{k\times n}(\mathbb{F}_2) \times \mathbb{F}_2^k$ *or it is obtained by randomly choosing* $A \in \mathcal{M}_{k\times n}(\mathbb{F}_2)$ *and outputting* (A, v)*, such that after splitting* A *into* w *sub-blocs* A_i *of size* $k \times \frac{n}{w}$ *and choosing* w *columns of* A*, one per block* A_i*,* v *is the sum of the* w *columns.*

Theorem

The DDCRSD problem (Problem 3) is as hard as the DCRSD problem (Problem 1).

A Family of Code-Based Collision-Resistant Hash Functions

Algorithm 3 Function RE()

Require: $u \in \mathbb{F}_2^n$.
Ensure: $v \in \mathbb{F}_2^n$.
Let $v = 0^n \in \mathbb{F}_2^n$.
Split v into w sub-blocs v_i of size $\frac{n}{w}$
Split u in $\frac{n}{w\log_2(\frac{n}{w})}$ blocs of size $w\log_2(\frac{n}{w})$
for all bloc s of u **do**
Split s in w parts $s_1, \ldots, s_w$ of $\log_2(\frac{n}{w})$ bits
Convert each bit string s_i to its corresponding integer s'_i between 1 and $\frac{n}{w}$
Set to 1 the bit at position s'_i in each v_i.
end for
return v

A Family of Code-Based Collision-Resistant Hash Functions

Algorithm 6 Hash function h_H

Require: $(x, y) \in \mathbb{F}_2^k \times \mathbb{F}_2^k$, $H \in \mathscr{M}_{k \times n}(\mathbb{F}_2)$

Ensure: h the hash of $x \parallel y$. Where $\parallel$ denote the conca

$u \leftarrow x \parallel y$

$h \leftarrow H \cdot \mathsf{RE}(u)$, RE is as in Algorithm 3.

return h

A Family of Code-Based Collision-Resistant Hash Functions

Theorem

The family of hash functions defined by Algorithm 6 is one way and collision resistant problem as the 2-DCRNSD (Problem 2) is hard.

The Proposed Code-based Accumulator

Let ℓ be an integer ≥ 0. Our code-based accumulator uses a Merkle tree with $N = 2^{\ell}$ leaves. This accumulator scheme is based on the family of code-based collision-resistant hash functions presented above.

The Proposed Code-based Accumulator

Let ℓ be an integer ≥ 0. Our code-based accumulator uses a Merkle tree with $N = 2^{\ell}$ leaves. This accumulator scheme is based on the family of code-based collision-resistant hash functions presented above.

Definition

A binary tree is a tree data structure in which each node has at most two children, which are referred to as the left child and the right child.

The Proposed Code-based Accumulator

Let ℓ be an integer ≥ 0. Our code-based accumulator uses a Merkle tree with $N = 2^\ell$ leaves. This accumulator scheme is based on the family of code-based collision-resistant hash functions presented above.

Definition

A binary tree is a tree data structure in which each node has at most two children, which are referred to as the left child and the right child.

The length of the longest path from the root to a leaf is said height, where the length between two leafs is the number of node between these two leafs. Also, we say that a binary tree of height ℓ is complete if it has 2^ℓ leaves and $2^\ell - 1$ interior nodes.

Merkle tree

Let h be a one-way hash function and Φ a function which maps the set of nodes of arbitrary length to the set of k-length strings: $n \mapsto \Phi(n) \in \{0,1\}^k$. A Merkle tree is a complete binary tree equipped with two functions h and Φ. For two children nodes n_{left} and n_{right}, of any interior node n_{parent}, the function Φ satisfies:

$$\Phi(n_{parent}) = \Phi(n_{left} \parallel n_{right})$$

where $\parallel$ is the concatenation symbol.

The Proposed Code-based Accumulator

our accumulator is composed by Algorithms 7 to 10 hereafter.

The Proposed Code-based Accumulator

our accumulator is composed by Algorithms 7 to 10 hereafter.

$\mathsf{Gen}(1^\lambda)$: This algorithm (Algorithm 7 below) outputs the global parameters (n, k, t) needed to generate our double circulant code C and the public parameter $\mathsf{p_k} = H \in \mathcal{M}_{k\times n}(\mathbb{F}_2)$ which is a parity check matrix of C.

The Proposed Code-based Accumulator

our accumulator is composed by Algorithms 7 to 10 hereafter.

$\mathsf{Gen}(1^\lambda)$: This algorithm (Algorithm 7 below) outputs the global parameters (n, k, t) needed to generate our double circulant code C and the public parameter $\mathsf{p_k} = H \in \mathcal{M}_{k\times n}(\mathbb{F}_2)$ which is a parity check matrix of C.

Algorithm 7 Gen() Algorithm

Require: 1^λ a security parameter
Ensure: (k, n, t), $\mathsf{p_k}$

pick $(k, n, t) \in \mathbb{N}^3$, such that there exist an (n, k, t) double circulant code and Problems 1 and 2 get a security level of λ bits where $n = 2k$.
Pick randomly $H \in \mathcal{M}_{k\times n}(\mathbb{F}_2)$, a parity check matrix of an (n, k, t) double circulant code;
$\mathsf{p_k} \leftarrow H$.

$\mathsf{Eval}(\mathsf{p_k}, \mathcal{X})$:

$\mathsf{Eval}(\mathsf{p_k}, \mathcal{X})$: Using a Merkle tree, Algorithm 8 takes as input a set of values $\mathcal{X} = \{x_0, \ldots, x_{N-1}\}$ to accumulate in a value $A_{\mathcal{X}}$ and auxiliary value ξ.

$\mathsf{Eval}(\mathsf{p_k}, \mathcal{X})$:

$\mathsf{Eval}(\mathsf{p_k}, \mathcal{X})$: Using a Merkle tree, Algorithm 8 takes as input a set of values $\mathcal{X} = \{x_0, \ldots, x_{N-1}\}$ to accumulate in a value $A_{\mathcal{X}}$ and auxiliary value ξ.
For each $j \in [0, N-1]$, let $(j_1, \ldots, j_\ell)$ be the binary representation of j on ℓ bits and write $x_j = y_{j_1,\ldots,j_\ell}$.

$\mathsf{Eval}(\mathsf{p_k}, \mathcal{X})$:

$\mathsf{Eval}(\mathsf{p_k}, \mathcal{X})$: Using a Merkle tree, Algorithm 8 takes as input a set of values $\mathcal{X} = \{x_0, \ldots, x_{N-1}\}$ to accumulate in a value $A_{\mathcal{X}}$ and auxiliary value ξ.
For each $j \in [0, N-1]$, let $(j_1, \ldots, j_\ell)$ be the binary representation of j on ℓ bits and write $x_j = y_{j_1,\ldots,j_\ell}$.
We design Algorithm 8 from the tree of depth ℓ based on N leaves $y_{0,\ldots,0}$, $\ldots$, $y_{1,\ldots,1}$.

$\mathsf{Eval}(\mathsf{p_k}, \mathcal{X})$:

Algorithm 8 Eval() Algorithm

Require: $\mathsf{p_k}$, $\mathscr{X}$
Ensure: $A_{\mathscr{X}}$, ξ
set ℓ as the number of bits of $N - 1$
randomly chosen $u \in \mathbb{F}_2^k$
set $y_{1,1,\dots,1}$ and ξ to u
for $i = \ell, \dots, 0$ **do**
$y_{j_1,\dots,j_i} = h_H(y_{j_1,\dots,j_i,0}, y_{j_1,\dots,j_i,1})$, where $(j_1, \dots, j_i) \in \{0,1\}^i$
if $i = 0$ **then**
$A_{\mathscr{X}} \leftarrow h_H(y_0, y_1)$
end if
end for

$\mathsf{WitCreate}(\mathsf{p_k}, \mathcal{X}, A_{\mathcal{X}}, \xi, x)$

$\mathsf{WitCreate}(\mathsf{p_k}, \mathcal{X}, A_{\mathcal{X}}, \xi, x)$: Algorithm 9 uses the public key $\mathsf{p_k}$ and the $\mathsf{Eval}()$ algorithm to output the witness W_x associate to x.

WitCreate($\mathsf{p_k}, \mathcal{X}, A_{\mathcal{X}}, \xi, x$)

WitCreate($\mathsf{p_k}, \mathcal{X}, A_{\mathcal{X}}, \xi, x$): Algorithm 9 uses the public key $\mathsf{p_k}$ and the Eval() algorithm to output the witness W_x associate to x.

Algorithm 9 WitCreate() Algorithm

Require: $\mathsf{p_k}, \mathscr{X}, \xi, A_{\mathscr{X}}, x$
Ensure: the witness W_x of x
  if $x \notin \mathscr{X}$ **then**
    return error
  else
    set $y_{1,1,\ldots,1}$ to ξ
    find the relative position j of x in $\mathscr{X}$
    write j in binary form: $(j_1, \ldots, j_\ell) \in \{0,1\}^\ell$
    $W_x = ((j_1, \ldots, j_\ell), (y_{j_1,\ldots,j_{\ell-1},\bar{j}_\ell}, \ldots, y_{j_1,\bar{j}_2}, y_{\bar{j}_1}))$ where $y_{j_1,\ldots,j_{\ell-1},\bar{j}_\ell}, \ldots, y_{j_1,\bar{j}_2}, y_{\bar{j}_1}$ are computed as in Algorithm 8, and $\bar{j}_i = j_i \oplus 1$, where $\oplus$ is the bitwise operation.
  end if

$\mathsf{TVerify}(\mathsf{p_k}, A_{\mathcal{X}}, W_x, x)$

$\mathsf{TVerify}(\mathsf{p_k}, A_{\mathcal{X}}, W_x, x)$ defined in Algorithm 10 proves using a recursive function that W_x is or not a generated witness of x.

Algorithm 10 TVerify() Algorithm

Require: $\mathsf{p_k}$, $A_{\mathscr{X}}$, x, W_x
Ensure: true or false

```
transcript W_x to ((j_1, ..., j_ℓ), (w_ℓ, ..., w_1))
v_ℓ ← x
for i = ℓ − 1 to 0 do
    if j_{i+1} = 0 then
        v_i = h_H(v_{i+1}, w_{i+1})
    else
        v_i = h_H(w_{i+1}, v_{i+1})
    end if
end for
if v_0 = A_X then
    return true
else
    return false
end if
```

At the beginning, the system uses $\mathsf{Gen}()$ Algorithm 7 to output and publish the public parameters $(k, n, t) \in \mathbb{N}^3$ and the public key $H \in \mathcal{M}_{k\times n}(\mathbb{F}_2)$.

At the beginning, the system uses $\mathsf{Gen}()$ Algorithm 7 to output and publish the public parameters $(k, n, t) \in \mathbb{N}^3$ and the public key $H \in \mathcal{M}_{k\times n}(\mathbb{F}_2)$. Thereafter the system randomly generates a set of $N-1$ values $\mathcal{X} = \{x_0, \ldots, x_{N-2}\}$ where $x_i \in \mathbb{F}_2^k$.

At the beginning, the system uses $\mathsf{Gen}()$ Algorithm 7 to output and publish the public parameters $(k, n, t) \in \mathbb{N}^3$ and the public key $H \in \mathcal{M}_{k\times n}(\mathbb{F}_2)$. Thereafter the system randomly generates a set of $N-1$ values $\mathcal{X} = \{x_0, \ldots, x_{N-2}\}$ where $x_i \in \mathbb{F}_2^k$.After all the operations above, the system computes and publish the accumulated value $A_{\mathcal{X}}$ and the auxiliary value ξ by providing the public key $\mathsf{p_k} = H$ and the set $\mathcal{X}$ to $\mathsf{Eval}()$ Algorithm 8.

At the beginning, the system uses $\mathsf{Gen}()$ Algorithm 7 to output and publish the public parameters $(k, n, t) \in \mathbb{N}^3$ and the public key $H \in \mathcal{M}_{k\times n}(\mathbb{F}_2)$. Thereafter the system randomly generates a set of $N-1$ values $\mathcal{X} = \{x_0, \dots, x_{N-2}\}$ where $x_i \in \mathbb{F}_2^k$.After all the operations above, the system computes and publish the accumulated value $A_{\mathcal{X}}$ and the auxiliary value ξ by providing the public key $\mathsf{p_k} = H$ and the set $\mathcal{X}$ to $\mathsf{Eval}()$ Algorithm 8. Now everyone can compute the witness W_x of any element $x \in \mathcal{X}$ by providing the public key $\mathsf{p_k}$, the auxiliary value ξ and the element x to $\mathsf{WitCreate}()$ Algorithm 9

At the beginning, the system uses $\mathsf{Gen}()$ Algorithm 7 to output and publish the public parameters $(k, n, t) \in \mathbb{N}^3$ and the public key $H \in \mathcal{M}_{k\times n}(\mathbb{F}_2)$. Thereafter the system randomly generates a set of $N - 1$ values $\mathcal{X} = \{x_0, \ldots, x_{N-2}\}$ where $x_i \in \mathbb{F}_2^k$.After all the operations above, the system computes and publish the accumulated value $A_{\mathcal{X}}$ and the auxiliary value ξ by providing the public key $\mathsf{p_k} = H$ and the set $\mathcal{X}$ to $\mathsf{Eval}()$ Algorithm 8. Now everyone can compute the witness W_x of any element $x \in \mathcal{X}$ by providing the public key $\mathsf{p_k}$, the auxiliary value ξ and the element x to $\mathsf{WitCreate}()$ Algorithm 9

At the end, everyone can also verify the membership of x to $\mathcal{X}$ by providing the public key $\mathsf{p_k}$; the accumulated value $A_{\mathcal{X}}$, the element x and the witness W_x to $\mathsf{TVerify}()$ Algorithm 10.

Example

Let C be a $[16, 8]$-code which a parity check matrix is H is generated by $\mathsf{Gen}()$ function (Algorithm 7).

$$H = \left(\begin{array}{c|c} & 10111001 \\ & 11011100 \\ & 01101110 \\ & 00110111 \\ I_8 & 10011011 \\ & 11001101 \\ & 11100110 \\ & 01110011 \end{array} \right) \tag{1}$$

In this example we take the following set $\mathcal{X} = \{00100011, 01110010, 01101001, 10101101, 11001001, 10100110, 10110111\}$ of $2^3 - 1 = 7$ elements.

Example

The execution of $\mathsf{Eval}()$ function (Algorithm 8) outputs two values: the accumulated value $A_{\mathcal{X}} = 00010100$ and the auxiliary value $\xi = 00100110$. In Figure 1 , we present the Merkle tree associate to this example.

Example

The execution of Eval() function (Algorithm 8) outputs two values: the accumulated value $A_{\mathcal{X}} = 00010100$ and the auxiliary value $\xi = 00100110$. In Figure 1 , we present the Merkle tree associate to this example.

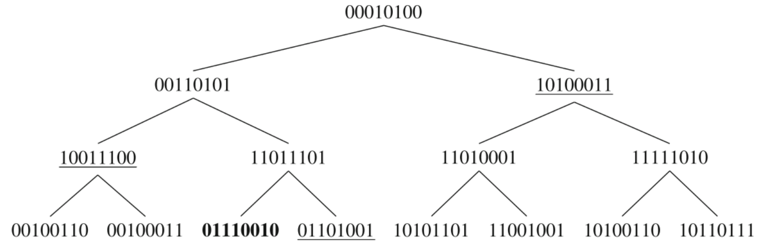

Fig. 1 Merkel Tree Associate to the set $\mathscr{X}$

Example

The root of the tree is the accumulated value $A_{\mathcal{X}} = 00010100$ and the $2^3 = 8$ leaves are the auxiliary value $\xi = 00100110$ and the elements of the set $\mathcal{X}$.

Example

The root of the tree is the accumulated value $A_{\mathcal{X}} = 00010100$ and the $2^3 = 8$ leaves are the auxiliary value $\xi = 00100110$ and the elements of the set $\mathcal{X}$. The first step to generate the witness of the element $x = \mathbf{01110010}$ which is in bold on Figure 1 is to get its relative position on the set $\mathcal{X}$ which is 2 and 010 in binary representation.

Example

The root of the tree is the accumulated value $A_{\mathcal{X}} = 00010100$ and the $2^3 = 8$ leaves are the auxiliary value $\xi = 00100110$ and the elements of the set $\mathcal{X}$. The first step to generate the witness of the element $x = \mathbf{01110010}$ which is in bold on Figure 1 is to get its relative position on the set $\mathcal{X}$ which is 2 and 010 in binary representation. The second step is to use $\mathsf{WitCreate}()$ function (Algorithm 9) to get the underlined elements on Figure 1.

Example

The root of the tree is the accumulated value $A_{\mathcal{X}} = 00010100$ and the $2^3 = 8$ leaves are the auxiliary value $\xi = 00100110$ and the elements of the set $\mathcal{X}$. The first step to generate the witness of the element $x = \mathbf{01110010}$ which is in bold on Figure 1 is to get its relative position on the set $\mathcal{X}$ which is 2 and 010 in binary representation. The second step is to use $\mathsf{WitCreate}()$ function (Algorithm 9) to get the underlined elements on Figure 1. Then the witness W_x of $x = \mathbf{01110010}$ is $W_x = \{010, 01101001, 10011100, 10100011\}$. The only way to check the membership of $x = \mathbf{01110010}$ is to use $\mathsf{TVerify}()$ function (Algorithm 10) witch should output True.

Security of the Proposed Accumulator

The security analysis of the proposed scheme consists in proving the correctness and the Collision Freeness.

Security of the Proposed Accumulator

The security analysis of the proposed scheme consists in proving the correctness and the Collision Freeness. Additionally to these properties, we also prove the Indistinguishability of our proposal.

Security of the Proposed Accumulator

The security analysis of the proposed scheme consists in proving the correctness and the Collision Freeness. Additionally to these properties, we also prove the Indistinguishability of our proposal.

Correctness: To ensure the correctness of our scheme, we show that the execution of TVerify() Algorithm 10 will always succeed if the key is honestly generated and accumulators as well as witnesses are correctly computed.

Security of the Proposed Accumulator

The security analysis of the proposed scheme consists in proving the correctness and the Collision Freeness. Additionally to these properties, we also prove the Indistinguishability of our proposal.

Correctness: To ensure the correctness of our scheme, we show that the execution of $\mathsf{TVerify}()$ Algorithm 10 will always succeed if the key is honestly generated and accumulators as well as witnesses are correctly computed.

Lemma

If all $\mathsf{Gen}()$, $\mathsf{Eval}()$ and $\mathsf{WitCreate}()$ algorithms are correctly computed, then the execution of $\mathsf{TVerify}()$ algorithm outputs true.

Security of the Proposed Accumulator

Collision Freeness: To ensure the collision freeness properties, we show that an adversary cannot find a witness for a non-accumulated value.

Lemma

If the family of hash functions defined by Algorithm 6 is resistant to collisions then our cryptographic accumulator is collision freeness

Security of the Proposed Accumulator

Theorem

The proposed cryptographic accumulator is secure based on the collision resistant property of the family of hash functions defined by Algorithm 6.

Security of the Proposed Accumulator

Indistinguishability: To ensure the Indistinguishability of the proposed cryptographic accumulator, we prove that from two sets and an accumulated value, an adversary cannot guess what set was accumulated.

Theorem

The proposed cryptographic accumulator defined by Algorithms 7 to 10 is Indistinguishable based on the hardness of the DDCRSD Problem (Problem 3).

Implementation

Implementation. To evaluate the practicability of the proposed code-based cryptographic accumulator, we have used a computer running at 2.6 GHz CPU with 8 GB of RAM.

Implementation

Implementation. To evaluate the practicability of the proposed code-based cryptographic accumulator, we have used a computer running at 2.6 GHz CPU with 8 GB of RAM. For error-correcting codes generating and all algebraic operations, we have used the FlexyProvider library implemented in Java language.

Implementation

Implementation. To evaluate the practicability of the proposed code-based cryptographic accumulator, we have used a computer running at 2.6 GHz CPU with 8 GB of RAM. For error-correcting codes generating and all algebraic operations, we have used the FlexyProvider library implemented in Java language. To get 80-bit security, we have choosed $n = 694$, $k = 347$ and $t = 76$.

Table 1 The algorithms execution time and memory storage needed (in Bit)

N	p_k	$A_{\mathcal{X}}$	ξ	W_x	Eval(ms)	WitCreate(ms)	TVerify(ms)
64	347	347	347	2088	1.705	1.981	0.190
128	347	347	347	2436	3.522	3.986	0.257
1024	347	347	347	3480	29.901	26.465	0.329
16384	347	347	347	4872	434.343	363.928	0.448

The Proposed Fully Dynamic Group Signature scheme (FDGS)

As an application, we propose a code-based fully dynamic group signature scheme using the Merkel tree cryptographic accumulator presented above.

The Updated Algorithm

By an updated algorithm, it will be possible to efficiently add and remove users from the proposed $FDGS$.

Algorithm 11 TUpdate() Algorithm

Require: A Merkel tree cryptographic accumulator $\mathscr{T}$, the binary representation $(j_1, \ldots, j_\ell)$ of an integer j and a value d'.

Ensure: The updated Merkel tree $\mathscr{T}$.

Let d_j be the value at the position j in τ and $((j_1, \ldots, j_\ell), (w_\ell, \ldots, w_1))$ be its associated witness.

Set $v_\ell = d'$ and recursively compute the path $v_\ell, v_{\ell-1}, \cdots, v_1, v_0 \in \mathbb{F}_2^k$ as in Algorithm 8.

Set $A_{\mathscr{X}} = v_0; y_{j_1} = v_1; \cdots; y_{j_1,j_2,\cdots,j_{\ell-1}} = v_{\ell-1}; y_{j_1,j_2,\cdots,j_\ell} = v_\ell = d'$.

GSetup(λ):

The setup algorithm described in algorithm **??**, takes in input the security parameter λ and outputs the public parameter $\mathsf{p_p}$.

Algorithm 12 GSetup(λ) Algorithm

Require: A security parameter λ.
Ensure: a public parameter $\mathsf{p_p}$.

- The GM chooses tow polynomially bound positive integers N and ℓ such that $N = 2^{\ell} - 1$ where N is the capacity of our group.
- Pick $(k_A, n_A, t_A) \in \mathbb{N}^3$, such that there exists an (n_A, k_A, t_A) double circulant code and Problems 1 and 2 get a security level of λ bits where $n_A = 2k_A$.
- Pick $(n_M, q_M, w_M, t_M) \in \mathbb{N}^4$, such that the Randomized QC-MDPC McEliece Encryption Scheme described in Sect. 2.3 get a security level of λ bits.
- Pick a hash function $\mathscr{H}_{FS} : \{0, 1\}^* \rightarrow \{1, 2, 3\}^{\sigma}$ to be modelled as a random oracle in the Fiat-Shamir transformations. Here, σ is the number of round needed in the Fiat-Shamir transformation to get a security level ok λ bits.
- Let COM be the string commitment scheme defined in Sect. 5.1, to be used in our zero-knowledge argument systems.

return $\mathsf{p_p} = (k_A, n_A, t_A, n_M, q_M, w_M, t_M, N, \ell, \mathscr{H}_{FS}, \mathsf{COM})$

$\langle \mathsf{GKgen}_{\mathsf{GM}}(\mathsf{p_p}), \mathsf{GKgen}_{\mathsf{TM}}(\mathsf{p_p}) \rangle$:

In this interactive protocol, described in Algorithm 13, the group manager GM and the tracing manager TM initialize their keys and the public group information.

Algorithm 13 $\langle \mathsf{GKgen}_{\mathsf{GM}}(\mathsf{p_p}), \mathsf{GKgen}_{\mathsf{TM}}(\mathsf{p_p}) \rangle$ Algorithm

Require: A public parameter $\mathsf{p_p}$.

- $\mathsf{GKgen}_{\mathsf{GM}}(\mathsf{p_p})$. GM, by using k_A, n_A, t_A, run Gen algorithm, get the matrix H and set $\mathsf{m_{pk}} = H_A \in \mathbb{F}_2^{q_M \times n_M}$ and set $\mathsf{m_{sk}} = \emptyset$.
- $\mathsf{GKgen}_{\mathsf{TM}}(\mathsf{p_p})$. TM uses n_M, q_M, w_M, t_M to run the setup algorithm of the Randomized QC-MDPC McEliece Encryption Scheme described in Sect. 2.3 and get the parity-check matrix $H_M \in \mathbb{F}_2^{q_M \times n_M}$ and its corresponding generator matrix $G_M \in \mathbb{F}_2^{(n_M - q_M) \times n_M}$ in row reduced echelon form. Then, TM sets $\mathsf{t_{sk}} = H_M$ and $mdk = G_M$
- TM sends $\mathsf{t_{pk}}$ to GM who initialises as follow:
  - Table $\mathbf{r} = (\mathbf{r}[0][1], \mathbf{r}[0][2], \cdots, \mathbf{r}[N-1][1], \mathbf{r}[N-1][2])$. For each $i \in [0, N-1]$: $\mathbf{r}[i][1] = 0^{k_A}$ will be used to store the registered user public key and $\mathbf{r}[i][2] = 0$ will be used to store the epoch at which the user joins.
  - The Merkel tree $\mathscr{T}$ built on top of $\mathbf{r}[0][1], \cdots, \mathbf{r}[N-1][1]$.
  - Counter of registered users $c = 0$

  Then, GM outputs $\mathsf{g_{pk}} = (\mathsf{p_p}, \mathsf{m_{pk}}, \mathsf{t_{pk}})$ and publishes the initial information $\mathfrak{I} = \emptyset$. He keeps $\mathscr{T}$ and c as secret.

$\mathsf{UKgen}(\mathsf{p_p})$:

In this algorithm (Algorithm 14), each potential group user can generate its key pair.

Algorithm 14 $\mathsf{UKgen}(\mathsf{p_p})$ Algorithm

Require: A public parameter $\mathsf{p_p}$.
Ensure: The user key pair $(\mathsf{u_{pk}}, \mathsf{u_{sk}})$.

– The user picks a non-zero random vector $x \in \mathbb{F}_2^{n_A}$ and computes $p = H_A \cdot \mathsf{RE}(x)$.
– Set $\mathsf{u_{pk}} = p$ and $\mathsf{u_{sk}} = x$

$\mathsf{GUpdate}(\mathsf{g_{pk}}, \mathsf{m_{sk}}, \mathfrak{I}_\tau, \mathbb{S}, \mathbf{r})$:

This algorithm (Algorithm 15) is run by GM to update the group information while advancing the epoch.

Algorithm 15 $\mathsf{GUpdate}(\mathsf{g_{pk}}, \mathsf{m_{sk}}, \mathfrak{I}_\tau, \mathbb{S}, \mathbf{r})$ Algorithm

Require: the current group information $\mathfrak{I}_\tau$, the group public key $\mathsf{g_{pk}}$, the GM secret key $\mathsf{m_{sk}}$, a table $\mathbb{S}$ containing the public keys of registered users to be revoked and the table $\mathbf{r}$.

Ensure: $\mathfrak{I}_{\tau_{new}}$.

1. if $\mathbb{S} = \emptyset$ then go to Step 2.
   Otherwise, $\mathbb{S} = \{\mathbf{r}[i_1][1], \cdots, \mathbf{r}[i_r][1]\}$, for some $t \in [1, N]$ and some $i_1, \cdots, i_r \in [0, N-1]$. Then, for all $t \in [0, r]$, GM runs $\mathsf{TUpdate}(\mathsf{bin}(i_t), 0^{n_A})$ to update the tree $\mathscr{T}$.
2. For each active user $j \in [0, N-1]$, let $w_j \in \mathbb{F}_2^\ell \times \left(\mathbb{F}_2^k\right)$ be the witness for the fact that p_j is accumulated in $u_{\tau_{new}}$. Then GM publishes the group information of the new epoch as:

$$\mathfrak{I}_{\tau_{new}} = \left(u_{\tau_{new}}, \{w_j\}_j\right)$$

IsActive$(\mathfrak{I}_\tau, \mathbf{r}, \mathsf{u}_{\mathsf{id}})$:

In this algorithm (Algorithm 16), if
the user u_{id} is active at epoch τ we outputs1 and 0 otherwise.

Algorithm 16 IsActive$(\mathfrak{I}_\tau, \mathbf{r}, \mathsf{u}_{\mathsf{id}})$ Algorithm

Require: $\mathfrak{I}_\tau$, $\mathbf{r}$, u_{id}.
Ensure: A value 0 or 1.

– if $\mathbf{r}[\mathsf{u}_{\mathsf{id}})][2] \neq \tau$ or $\mathbf{r}[\mathsf{u}_{\mathsf{id}})][1] = 0_A^k$ then return 0.
– Else, return 1.

$\mathsf{Join}(\mathfrak{I}_\tau, \mathsf{g}_{\mathsf{pk}}, \mathsf{u}_{\mathsf{pk}}, \mathsf{u}_{\mathsf{sk}})$:

In this algorithm defined in Algorithm 17, the user interacts with the GM to join the group at epoch τ

Algorithm 17 $\mathsf{Join}(\mathfrak{I}_\tau, \mathsf{g}_{\mathsf{pk}}, \mathsf{u}_{\mathsf{pk}}, \mathsf{u}_{\mathsf{sk}})$ Algorithm

Require: the current group information $\mathfrak{I}_\tau$, the group public key g_{pk} and the user key pair $(\mathsf{u}_{\mathsf{pk}}, \mathsf{u}_{\mathsf{sk}}) = (p, x)$

1. GM issues a member identifier for the user as $\mathsf{u}_{\mathsf{id}} = \mathsf{bin}(c) \in \mathbb{F}_2^{\ell}$.
2. The user sets his signing key as $\mathsf{g}_{\mathsf{sk}}[c] = (\mathsf{bin}(c), p, x)$
3. GM performs the following updates:
   - Updates the tree $\mathscr{T}$ by running $\mathsf{TUpdate}(\mathsf{bin}(c), p)$.
   - Registers the user to table $\mathbf{r}$ as $\mathbf{r}[c][1] = p$ and $\mathbf{r}[c][2] = \tau$.
   - Sets $c = c + 1$.
   - Sets $\mathbb{S} = \emptyset$ and runs $\mathsf{GUpdate}$.

$\mathsf{Issue}(\mathfrak{I}_\tau, \mathsf{m}_{\mathsf{sk}}, \mathsf{u}_{\mathsf{id}})$:

In Issue algorithm defined in Algorithm 18, the GM revokes the user with identifier u_{id} to the group at epoch τ.

Algorithm 18 $\mathsf{Issue}(\mathfrak{I}_\tau, \mathsf{m}_{\mathsf{sk}}, \mathsf{u}_{\mathsf{id}})$ Algorithm

Require: The current group information $\mathfrak{I}_\tau$, the group manager secret key m_{sk} and the user identifier u_{id}.

1. GM sets $\mathbb{S} = \{\mathbf{r}[\mathsf{u}_{\mathsf{id}}][2]\}$.
2. GM runs GUpdate.

$\mathsf{Sign}(\mathsf{g}_{\mathsf{pk}}, \mathsf{g}_{\mathsf{sk}}(j), \mathfrak{I}_\tau, M)$:

Let a user with the tuple $\mathsf{g}_{\mathsf{sk}}(j) = (\mathsf{bin}(j), p, x)$. To sign a message M using the group information at epoch τ, the user downloads u_τ and the witness $(\mathsf{bin}(j), (w_\ell, \cdots, w_1))$ from $\mathfrak{I}_\tau$ and proceeds as in Algorithm 19

Algorithm 19 $\mathsf{Sign}(\mathsf{g}_{\mathsf{pk}}, gsk(j), \mathfrak{I}_\tau, M)$ Algorithm

Require: $\mathsf{g}_{\mathsf{pk}}, gsk(j), \mathfrak{I}_\tau, M$.

1. Run the function $\mathscr{E}(G_M, \mathsf{bin}(j))$ defined in Sect. 2.3 to encrypt the vector $\mathsf{bin}(j)$ and get the cipher text $s \in \mathbb{F}_2^{n_M}$.
2. Use the protocol defined in Sect. 5.1 to generate a Non-Interactive Zero-Knowledge Argument of Knowledge (NIZKAoK) Π_{gs} to demonstrate the possession of a valid tuple

$$\zeta = (x, p, \mathsf{bin}(j), w_\ell, \cdots, w_1) \tag{12}$$

such that $\mathsf{TVerify}(H_A, u_\tau, p, (\mathsf{bin}(j), (w_\ell, \cdots, w_1))) = 1$, $H_A \cdot x = p$ and $p \neq 0^{k_A}$. Using the Fiat-Shamir heuristic, the protocol is repeated σ times to achieve a negligible soundness error and get the triple: $\Pi_{gs} = (\{\mathsf{C}\}_{i=1}^{\sigma}, \mathsf{e}, \{\mathsf{R}\}_{i=1}^{\sigma})$, where $\mathsf{e} = \mathscr{H}_{FS}(M, \{\mathsf{C}\}_{i=1}^{\sigma}, u_\tau, s, H_A, G_M)$.
3. Output the group signature

$$\Sigma = (\Pi_{gs}, s). \tag{13}$$

Verify$(\mathsf{g}_{\mathsf{pk}}, \mathfrak{I}_\tau, M, \Sigma)$:

This algorithm (Algorithm 20) uses the global public key g_{pk} and the group information at epoch τ to verify the signature Σ of the message M.

Algorithm 20 Verify$(\mathsf{g}_{\mathsf{pk}}, \mathfrak{I}_\tau, M, \Sigma)$ Algorithm

Require: $\mathsf{g}_{\mathsf{pk}}, \mathfrak{I}_\tau, M, \Sigma$.

1. Get $u_\tau \in \mathbb{F}_2^{k_A}$ from $\mathfrak{I}_\tau$.
2. Parse Σ as $\Sigma = (\{\mathsf{C}\}_{i=1}^{\sigma}, (\mathsf{e}_1, \cdots, \mathsf{e}_\sigma), \{\mathsf{R}\}_{i=1}^{\sigma}, s)$.
3. If $(\mathsf{e}_1, \cdots, \mathsf{e}_k) \neq \mathscr{H}_{FS}(M, \{\mathsf{C}\}_{i=1}^{\sigma}, u_\tau, s, H_A, G_M)$, then return 0.
4. For each $i = 1$ to σ, check the validity of R_i with respect to C_i and e_i by running the verification phase of the Stern-like protocol (Algorithm 4). If one of the conditions is not satisfied, then return 0.
5. Return 1.

$\mathsf{Trace}(\mathsf{g_{pk}}, \mathsf{t_{sk}}, \mathfrak{I}_\tau, \mathbf{r}, M, \Sigma)$:

In this algorithm (Algorithm 21), TM uses his secret key $\mathsf{t_{sk}} = H_M$ to reveal the $\mathsf{u_{id}}$ of the signer.

Algorithm 21 $\mathsf{Trace}(\mathsf{g_{pk}}, \mathsf{t_{sk}}, \mathfrak{I}_\tau, \mathbf{r}, M, \Sigma)$ Algorithm

Require: $\mathsf{g_{pk}}, \mathsf{t_{sk}}, \mathfrak{I}_\tau, \mathbf{r}, M, \Sigma$.

1. TM parses Σ as in (13) and runs the function $\mathscr{D}(H_M, s)$ defined in Sect. 2.3 to get $b' \in \mathbb{F}_2^\ell$.
2. If $\mathfrak{I}_\tau$, does not include a witness containing b', then return *null*.
3. Let $j' \in [0, N-1]$ be the integer having binary representation b'. If the record $\mathbf{r}[j'][1]$ is 0^{k_A}, then return *null*.
4. TM uses the Stern-like protocol defined in Sect. 4 to generate a Non-Interactive Zero-Knowledge Argument of Knowledge (NIZKAoK) Π_{trace} to demonstrate the knowledge of $H_M \in \mathbb{F}_2^{q_M \times n_M}$ and $y \in \mathbb{F}_2^{k_M}$, such that $y = H_M \cdot (b' \parallel 0^{n_M - \ell})$. Using the Fiat-Shamir heuristic, the protocol is repeated many σ times to achieve a negligible soundness error and get the triple: $\Pi_{trace} = (\{\mathsf{C}\}_{i=1}^{\sigma}, \mathsf{e}, \{\mathsf{R}\}_{i=1}^{\sigma})$, where $\mathsf{e} = \mathscr{H}_{FS}(M, \{\mathsf{C}\}_{i=1}^{\sigma}, \mathfrak{I}_\tau, \mathsf{g_{pk}}, \Sigma, b')$.
5. Set $\mathsf{u_{id}} = b'$
6. Output $(\mathsf{u_{id}}, \Pi_{trace})$

$\mathsf{Judge}(\mathsf{g_{pk}}, \mathsf{u_{id}}, \mathfrak{I}_\tau, \Pi_{trace}, M, \Sigma)$:

In this algorithm (Algorithm 22), we verify the argument Π_{trace}.

Algorithm 22 $\mathsf{Judge}(\mathsf{g_{pk}}, \mathsf{u_{id}}, \mathfrak{I}_\tau, \mathit{\Pi}_{trace}, M, \mathit{\Sigma})$ Algorithm

Require: $\mathsf{g_{pk}}, \mathsf{u_{id}}, \mathfrak{I}_\tau, \mathit{\Pi}_{trace}, M, \mathit{\Sigma}$.

1. Get $u_\tau \in \mathbb{F}_2^{k_A}$ from $\mathfrak{I}_\tau$.
2. Parse $\mathit{\Pi}_{trace}$, as $\mathit{\Pi}_{trace} = (\{\mathsf{C}\}_{i=1}^{\sigma}, \mathsf{e}, \{\mathsf{R}\}_{i=1}^{\sigma})$.
3. If $(\mathsf{e}_1, \cdots, \mathsf{e}_\sigma) \neq \mathscr{H}_{FS}(M, \{\mathsf{C}\}_{i=1}^{\sigma}, \mathfrak{I}_\tau, \mathsf{g_{pk}}, \mathit{\Sigma}, b')$, then return 0.
4. For each $i = 1$ to σ, check the validity of R_i with respect to C_i and e_i by running the verification phase of the Stern-like protocol (Algorithm 4). If one of the conditions is not satisfied, then return 0.
5. Return 1.