Research Report for Qredo

Code-Based Ring Signatures

Authors:

Sofoclis Zambirinis
Theodosis Mourouzis
Nikolas Markou
Alexandros Hasikos

Thursday 30th August, 2018

Contents

Abstract

In this research report we describe in detail how to implement the code-based $(\widetilde{t}, N)$ Threshold Ring Signature Scheme with a Leader-Participant Model, which was presented in Zhou et al. (2017) [40]. For this, assume that there are N *possible signers* $S_i,\ i = 1, 2, ..., N$, forming a ring R and the threshold of generating a valid signature is $\widetilde{t}$, where $\widetilde{t} < N$. The purpose of the scheme is to allow at least $\widetilde{t}$ members (*true signers*) who belong to the ring of N possible signers, to cooperate with each other in order to sign a message, without leaking any identity information of the $\widetilde{t}$ true signers.

This Threshold Ring Signature Scheme is based on binary Goppa Codes, Patterson decoding algorithm, the Extended Euclidean Algorithm and Syndrome Decoding, and is essentially comprised of the following algorithms: *Setup* (Algorithm 1), *KeyGen* (Algorithm 2), *SignParticipant* (Algorithm 3), *SignLeader* (Algorithm 4), *Verify* (Algorithm 5).

1 Introduction

1.1 Introduction to Ring Signatures

A digital signature scheme (or simply a digital signature), is a mathematical scheme for verifying the authenticity of a digital document or message. A valid digital signature provides a reason for the recipient to believe that: (i) the message received was created by a known sender (authentication), (ii) the sender cannot refuse having sent the message (non-repudiation), and (iii) the message was not modified in transit (integrity). Digital signatures have been established as a standard ingredient of most cryptographic protocol suites and have a range of applications, such as to allow a signer to sign a message anonymously in a group or 'ring' of signers.

A ring signature is a special type of digital signature, which can be performed by any member of a group of users that each have keys. A message signed with a ring signature is therefore endorsed by someone in the ring or group of people. Such ring signatures should have the property that it should be computationally infeasible to determine which one of the group members' keys was used to produce the signature.

Here we present a short introduction to ring signatures, by following the introduction of Bender et al. (2006) [3], modified accordingly. Ring signatures allow a user to sign a message, so that a set of possible signers (called a 'ring') of which the user is a member, is identified, without revealing exactly which was the member of the ring that actually generated the signature. Therefore, a ring signature makes it possible to specify a set of possible signers without revealing which member actually produced the signature. Ring signatures were first formally proposed by Rivest et al. (2001) [29]. and has been studied extensively since then (e.g. see [1], [2], [5], [6], [13], [17], [21], [25], [37], [39]). Ring signatures are related, but incomparable, to group signatures (see [9]). Specifically, group signatures have the feature that the anonymity of a signer can be revoked by a designated group manager and so the signer can be traced. Unlike group signatures, ring signatures have no setup or revocation procedures, no group managers and no coordination. Any user can choose any set of possible signers which include himself, and sign any message by making use of his secret key and the others' public keys, without getting their approval or assistance. Here, users may be unaware of each other at the time they generate their public keys, rings may be formed 'on the fly' and in an ad-hoc manner, and users are given fine-grained control over the level of anonymity that is associated with a particular signature (by selecting the appropriate ring). In particular, ring signatures can be useful when the members do not want to cooperate.

Ring signatures naturally lend themselves to various applications (for example, see [29], [25], [13], [2]). The original motivation was to allow a secret to be leaked anonymously. Here we present a simple example that was proposed by Rivest et al. (2001) [29]. Suppose that Bob, a member of the cabinet wishes to leak a certain secret fact which concerns the Prime Minister to a journalist, in such a way that Bob remains anonymous, yet the journalist is convinced that the leak was indeed from a cabinet member. If Bob sends a standard digitally signed message to the journalist, although it would convince the journalist that the message came from a cabinet member, his identity will be revealed. If Bob sends a message to the journalist through a standard anonymizer, then the journalist would have no reason to believe that the message really came from a cabinet member at all. A standard group signature scheme does not solve the problem, since it requires the prior cooperation of the other group members to set up, and leaves Bob vulnerable to later identification by the group manager, who may be controlled by the Prime Minister. The proper approach is for Bob to send the story to the journalist through an anonymizer, signed with a ring signature scheme that names each cabinet member (including himself) as a ring member. The journalist can verify the ring signature on the message and learn that it definitely came from a cabinet member. He can even post the ring signature in his web page or paper, to prove his readers that the story came from a reputable source. However, neither he nor his readers can determine the actual source of the leak, therefore the whistleblower has perfect protection, even if at a later stage the journalist is forced by a judge to reveal his source (i.e. the signed document).

1.2 Functional definition of Ring Signatures

In this subsection we present the functional definition of a ring signature scheme, according to Bender et al. (2006) [3].

We refer to an ordered list $R = (PK_1, PK_2, ..., PK_n)$ of public keys as a *ring*, and let $R[i] = PK_i$. We will freely use set notation, and say, e.g., that $PK \in R$ if there exists an index i such that $R[i] = PK$. We will always assume, without loss of generality, that the keys in a ring are ordered lexicographically.

A *Ring Signature Scheme* is a triple of PPT algorithms[1] (Gen, Sign, Vrfy) that, respectively, generate keys for a user, sign a message, and verify the signature of a message. Formally:

- $Gen(1^k)$, where k is a security parameter, outputs a public key PK and secret key SK.
- $Sign_{s,SK}(M, R)$ outputs a signature σ on the message M with respect to the ring $R = (PK_1, PK_2, ..., PK_n)$. We assume the following: (i) $(R[s], SK)$ is a

[1] PPT algorithm = A probabilistic polynomial-time algorithm

valid key-pair output by Gen; (ii) $|R| \geq 2$ (since a ring signature scheme is not intended to serve as a standard signature scheme); and (iii) each public key in the ring is distinct.

- $Vrfy_R(M, \sigma)$ verifies a purported signature σ on a message M with respect to the ring of public keys R.

We require the following completeness condition to hold: for any integer k, any $\{(PK_i, SK_i)\}_{i=1}^{n}$ output by $Gen(1^k)$, any $s \in [n]$, and any M, we have $Vrfy_R(M, Sign_{s,SK_s}(M, R)) = 1$ where $R = (PK_1, PK_2, ..., PK_n)$.

A *c-user ring signature scheme* is a variant of the above that only supports rings of fixed size c (i.e., the $Sign$ and $Vrfy$ algorithms only take as input rings R for which $|R| = c$, and correctness is only required to hold for such rings).

We may sometimes omit the input 's' to the signing algorithm (and simply write $\sigma \leftarrow Sign_{SK}(M, R)$), with the understanding that the signer can determine an index s for which SK is the secret key corresponding to public key $R[s]$. Strictly speaking, there may not be a unique such s when R contains incorrectly-generated keys; in real-world usage of a ring signature scheme, though, a signer will certainly be able to identify their public key.

A ring signature scheme is used as follows: At various times, some collection of users runs the key generation algorithm Gen to generate public and secret keys. We stress that no coordination among these users is assumed or required. When a user with secret key SK wishes to generate an anonymous signature on a message M, he chooses a ring R of public keys which includes his own, computes $\sigma \leftarrow Sign_{SK}(M, R)$ and outputs (σ, R). (In such a case, we will refer to the holder of SK as the *signer* of the message and to the holders of the other public keys in R as the *non-signers*.) Anyone can now verify that this signature was generated by *someone* holding a key in R by running $Vrfy_R(M, \sigma)$.

Although this functional definition by Bender et al. (2006) [3] of a ring signature scheme requires users to generate keys specifically for that purpose, this can be easily modified to work with any ring of users as long as they each have a public key for both encryption and signing.

In general, ring signatures must satisfy two independent notions of security: anonymity and unforgeability. Informally, the anonymity condition requires that an adversary will not be able to tell which member of a ring generated a particular signature. Unforgeability requires that an adversary should not be able to output (R, M, σ) such that $Vrfy_R(M, \sigma) = 1$ unless either (i) one of the public keys in R was chosen by the adversary, or (ii) a user whose public key is in R explicitly signed M previously (with respect to the same ring R). For formal definitions of anonymity and unforgeability, see [3].

1.3 A $(\widetilde{t}, N)$ Threshold Ring Signature Scheme

We use the definition of the threshold ring signature scheme, as presented in Zhou et al. (2017) [40] and Bresson et al.(2002) [6]. Assume that there are N signers $S_i,\ i = 1, 2, ..., N$, forming a ring R and the threshold of generating a valid signature is $\widetilde{t}$, where $\widetilde{t} < N$. For simplicity, assume that the first $\widetilde{t}$ signers $S_1, S_2, ..., S_t$ are the true signers in R. A $(\widetilde{t}, N)$ threshold ring signature scheme is comprised of four algorithms: $(Setup,\ KeyGen,\ Sign,\ Verify)$.

- $Setup(\lambda)$: The algorithm takes as input a security parameter λ and outputs the system public parameter P.
- $KeyGen(P)$: The algorithm takes as input the parameter P and generates N pairs of public-private keys (pk_i, sk_i) for the signers $S_i \in R,\ i = 1, 2, ..., N$. The N public keys $pk_i,\ i = 1, 2, ..., N$, form the ring public key $PK = \{pk_1, pk_2, ..., pk_N\}$ and each private key sk_i is sent to the signer S_i via a secure channel, $i = 1, 2, ..., N$.
- $Sign(\widetilde{m}, P, PK, TSK)$: The algorithm takes as input a message $\widetilde{m}$, the parameter P, the ring public key PK, and a private key set $TSK = \{sk_1, sk_2, ..., sk_{\widetilde{t}}\}$ of $\widetilde{t}$ signers, and outputs a threshold ring signature σ on $\widetilde{m}$.
- $Verify(\widetilde{m}, PK, \sigma)$ The algorithm takes as input the message $\widetilde{m}$, the ring public key PK, and the threshold ring signature σ, and outputs 1 if $(\widetilde{m}, \sigma)$ is a valid message-signature pair; otherwise, it outputs 0.

Note that in section 3, where the implementation of this scheme is described in detail, we split the third step *Sign* into two parts: *SignParticipant* and *SignLeader*.

2 Introduction to Linear Codes, Finite Fields, Goppa Codes

2.1 Linear Codes

In coding theory, a linear code is an error-correcting code for which any linear combination of codewords is also a codeword (Ryan and Lin (2009) [30]). In general, linear codes allow for more efficient encoding and decoding algorithms than other codes.

Linear codes are used in forward error correction and can be applied when transmitting bits on a communications channel so that, if errors occur in the communication, some errors can be corrected or detected by the recipient of a message block. The codewords in a linear block code are blocks of symbols that are encoded using more symbols than the original value to be sent [22]. A linear code of length n transmits blocks containing n symbols. For example, the [7,4,3] Hamming code is a linear binary code which represents 4-bit messages using 7-bit codewords. Two distinct codewords differ in at least three bits. As a consequence, up to two errors per codeword can be detected while a single error can be corrected [11]. This code contains 2^4=16 codewords.

More formally, a linear code of length n and rank k is a linear subspace C of the vector space $\mathbb{F}_q^n$ with dimension k, where $\mathbb{F}_q$ is the finite field with q elements. Such a code is called a q-ary code. In the special case when $q = 2$, the code is described as a binary code. The vectors in C are called codewords. The size n of a code is the number of codewords and equals q^k (i.e. $n = q^k$).

The weight of a codeword is the number of its elements that are nonzero and the distance between two codewords is the Hamming distance between them, that is, the number of elements in which they differ. The distance d of a linear code is the minimum weight of its nonzero codewords, or equivalently, the minimum distance between any two distinct codewords. A linear code of length n, dimension k, and distance d is called an $[n, k, d]$ code. We assume that $\mathbb{F}_q^n$ is given the standard basis, since each coordinate represents a bit which is transmitted across a 'noisy channel' with some small probability of transmission error. In general, an $[n, k, d]$ code has a $\lfloor \frac{d-1}{2} \rfloor$ error-correcting cabability.

As a linear subspace of $\mathbb{F}_q^n$, the entire code C (which may be very large) may be represented as the span of a minimal set of codewords (known as a basis in linear algebra). These basis codewords are often collated in the rows of a matrix G known as a generating matrix for the code C. When G has the block matrix form $G = (I_k|A)$ where I_k denotes the $k \times k$ identity matrix and A is some $(n - k) \times k$ matrix, then we say G is in standard form.

A matrix H representing a linear function $\phi : \mathbb{F}_q^n \to \mathbb{F}_q^{n-k}$ whose kernel is C is called a parity-check matrix of C; i.e. $C = Ker(\phi) = \{c \in \mathbb{F}_q^n | \phi(c) = 0\}$. Equivalently, H is a matrix whose null space is C. If C is a code with a generating matrix G in standard form, $G = (I_k|A)$, then $H = (-A^t|I_{n-k})$ is a parity-check matrix for C. The code generated by H is called the dual code of C.

In the next two subsections we present a short and very simplified introduction to finite fields, Goppa codes, representation of field elements, error correction and decoding, that was provided by Safieddine and Desmarais (2014) [31] (edited accordingly).

2.2 Finite Fields

A finite field (also known as Galois field) is a field that contains a finite number of elements. A finite field has the same properties as a normal field as well as some additional properties which are outlined below. Specifically, for all a, b, $c \in GF(n)$ (which is a finite field with n elements) we have:

1. Any addition or multiplication of two elements within the field will result in another element within the field, i.e. $a + b \in GF(n)$ and $a \cdot b \in GF(n)$.
2. Elements are associative, i.e. $a+(b+c) = (a+b)+c$ and $a\cdot(b\cdot c) = (a\cdot b)\cdot c$
3. Elements are commutative, i.e. $a + b = b + a$ and $a \cdot b = b \cdot a$
4. There exists an additive and multiplicative identity element, i.e. $a + 0 = a$ and $a \cdot 1 = a$
5. Each element has an additive inverse, i.e. $\forall a \in GF(n)\ \exists\ a' \in GF(n) : a + a' = 0$ (we write $a' = -a$)
6. Each element has a multiplicative inverse, i.e. $\forall a \in GF(n)\ \exists\ \widetilde{a} \in GF(n) : a \cdot \widetilde{a} = 1$ (we write $\widetilde{a} = a^{-1} = 1/a$).
7. Multiplication is distributive over addition, i.e. $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$

For Goppa codes, calculations done are in the field $GF(2^{\mathrm{m}})$ with an irreducible polynomial T.

2.3 Goppa Codes

An $[n, k, d]$ Goppa Code in the field $GF(2^{\mathrm{m}})$ with a t-error-correction capability, is a type of error correcting $[n, k, d]$ linear code $\mathscr{C}$ over the binary field $\mathbb{F}_2$, with $n = 2^m$, $k \geq n - mt$ and $d \geq 2t + 1$ (if the goppa polynomial $g(X)$ is irreducible and separable), with associated:

- Support set $\mathscr{L} = \{a_1, a_2, ..., a_n\} \subseteq GF(2^{\mathrm{m}})$
- Irreducible goppa polynomial g (*degree* t) s.t. $g(X) \neq 0,\ \forall X \in \mathscr{L}$
- Parity-Check Matrix $\mathscr{H}$ of dimensions $(m \cdot t) \times n$, with elements being 0's and 1's. The construction of the Parity-Check matrix of a Goppa Code is described in detail in Algorithm 6.
- Generator Matrix $\mathscr{G}$ - the rows of a generator matrix of a linear code are a basis of the code

We summarize the notation associated with $[n, k, d]$ Goppa codes over the binary field $\mathbb{F}_2$:
• $d =$ the code minimum distance, defined as $d = \min\{weight(c_1 - c_2) | c_1 \neq c_2 \in \mathscr{C}\}$
• $n =$ The code length
• $k =$ The code dimension

- t = The guaranteed error-correction capability
- q = The size of the field used (part of the CM parameters). We use $q = 2$.
- $m = log_2 n$ (or equivalently $n = 2^m$)
- g = A polynomial in $F_q[x]$
- a_i = An element of the finite field F_q
- $\Gamma = (g, a_1, a_2, ..., a_n)$ denotes the Goppa code.

2.3.1 Representation of Field Elements

Field elements in GF(2^{m}) can be represented in several ways including:

- Polynomial: $x^3 + x + 1 = 1x^3 + 0x^2 + 1x + 1$
- Binary: $(1011)_2$
- Integer: $(11)_{10}$

Polynomials in $GF(2^{\mathrm{m}})[X]/g(X)$ can be represented by:

- Polynomial: $f_{t-1}X^{t-1} + \cdots + f_1X + f_0$
- Array: $[f_0, f_1, \ldots, f_{t-1}]$

2.3.2 Error Correction

A sample Goppa code with $m = 3$, $n = 2^m$ (hence $n = 8$) and $t = 2$, has the following codewords (words where the syndrome is 0):

- 00000000
- 01011011
- 10110101
- 11101110

If 2 errors are added to the second codeword (positions 2 and 7):

$$01000010 \oplus 01011011 \rightarrow 00011001$$

Adding the error back we get (the original codeword):

$$00011001 \oplus 01000010 \rightarrow 01011011$$

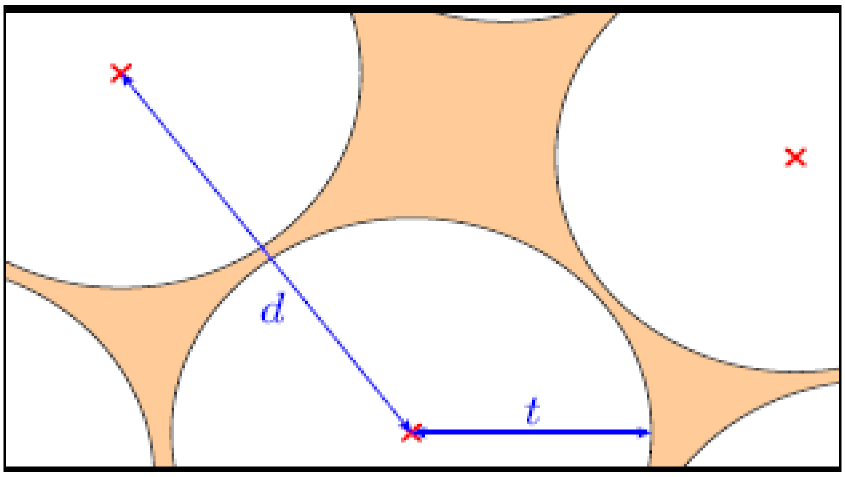
A codeword can be corrected with up to t errors.

2.3.3 Decoding

The following image illustrates how ciphertexts can be corrected:

- The red crosses represent the codewords (i.e. the word the cipher text will be corrected to).
- The white bubbles represent all the different ciphertexts that will correct to the bubble's codeword.
- t represents the correction capacity (i.e. how many errors that can be in a given codeword) - in this diagram, t is the radius of the bubble.
- d represents the minimum distance between any two codewords.
- The colored area represents elements that cannot be corrected by a decoding algorithm.

Any ciphertext in one of the white bubbles will be corrected to the nearest codeword within t errors. In other words, any ciphertext within one of the white bubbles will be corrected to the nearest red cross (x). The red cross will be within t errors because as we can see t is also the radius of the bubble.



2.4 Syndrome Decoding

Syndrome Decoding (SD) refers to the following problem: Given an integer w, a vector $s \in F_2^r$ and a $r \times n$ random binary matrix H, find a $1 \times n$ vector e, such that $weight(e) = w$ and $He^T = s^T$.

Syndrome Decoding is NP-complete (e.g. see [4]). Note that it is customary to write $e \in \epsilon_{n,w}$ to denote that vector e has length n and weight w.

3 Algorithmic Description & Pseudocode

In this section we present in detail the algorithmic description of the code-based Threshold Ring Signature Scheme of Zhou et al.(2017)[40]. For this, assume that there are N *possible signers* S_i, $i = 1, 2, ..., N$, forming a ring R and the threshold of generating a valid signature is $\widetilde{t}$, where $\widetilde{t} < N$. For simplicity, assume that the first $\widetilde{t}$ signers $S_1, S_2, ..., S_{\widetilde{t}}$ are the *true signers* in R. The purpose of the scheme is to allow at least $\widetilde{t}$ members in the ring of all N possible signers to cooperate with each other to sign a message, without leaking any identity information of the $\widetilde{t}$ signers. The code-based $(\widetilde{t}, N)$ Threshold Ring Signature Scheme is essentially comprised of the following algorithms: *Setup* (Algorithm 1), *KeyGen* (Algorithm 2), *SignParticipant* (Algorithm 3), *SignLeader* (Algorithm 4), *Verify* (Algorithm 5).

The Threshold Ring Signature Scheme is based Goppa Codes. For this, we present Algorithm 6 (*Construction of Parity-check matrix for Goppa Code*). Algorithm 6 takes as an input a binary irreducible $[n, k, d]$, t-error correcting Goppa code $\Gamma(L, g(z))$ in the finite field $GF(2^m)$, with Goppa separable irreducible polynomial $g(z) \in F_{2^m}[z]$ of degree t (where $m \cdot t = n - k$) and code-locator set $L = \{a_1, a_2, ..., a_n\} \subseteq F_{2^m}$. The output of this algorithm is an $(m \cdot t) \times n$ parity-check matrix H_b with elements being 0's and 1's (or equivalently, depending on representation, a $t \times n$ parity-check matrix H_a with elements being polynomial functions with variable a).

We also present Algorithm 9 (*Decoding Algorithm: The Decoding process of binary Goppa Codes, using Patterson Algorithm*), which takes as an input the received codeword y of length n with at most t errors, the binary Goppa code $\Gamma(L, g(z))$ with Goppa separable irreducible polynomial $g(z) \in F_{2^m}[z]$ of degree t, the code-locator set $L = \{a_1, a_2, ..., a_n\} \subseteq F_{2^m}$, and the respective parity-check matrix H. The output is the original codeword $\widetilde{c}$, after correcting the errors. This algorithm calls the Extended Euclidean Algorithm, which is discussed below.

The Extended Euclidean Algorithm (EEA) takes as an input two polynomials $a(X)$ and $b(X)$, and gives as an output the greatest common divisor (gcd) of these two polynomials, as well as two other polynomials $u(X)$ and $v(X)$ such that $gcd[a(X), b(X)] = u(X)a(X) + v(X)b(X)$. In other words, it expresses the $gcd[a(X), b(X)]$ as a linear combination of $a(X)$ and $b(X)$, with coefficients $u(X)$ and $v(X)$. The EEA, as presented in Jochemsz (2002) [19], is described in detail in Algorithm 13.

Note that in Zhou et al. (2017) [40] the terms signer and participant are sometimes interchanged, thus creating a confusion. For this, from now on, we will mainly use the term *all possible signers* or *participants* to denote all N members of the ring $R = \{S_i | i = 1, 2, ..., N\}$, and the term *true signers* to denote the $\widetilde{t}$ members of the ring that actually sign the message; i.e. the members of the set $\{S_i | i = 1, 2, ..., \widetilde{t}\}$. The remaining $N - \widetilde{t}$ members of the set $\{S_i | i = \widetilde{t}+1, \widetilde{t}+2, ..., N\}$ will denote the *not necessarily true signers* (or *possibly not signers*). A leader is a special signer, among the $\widetilde{t}$ true signers.

Note also that we have replaced the variable t of Zhou et al. (2017) [40] by $\widetilde{t}$, and then the variable w of Zhou et al. (2017) [40] by t, in order to make the notation in both the scheme and the Goppa codes consistent. Therefore, from now on and throughout this report, we will denote by $\widetilde{t}$ the number of *true signers* in a ring of N possible signers, whereas the variable t will denote the error-correction ability of the code (and, when goppa codes are considered, t will also be the degree of the goppa polynomial g). Thus, from now on we will refer to a $(\widetilde{t}, N)$ threshold ring signature scheme. Moreover, in the algorithm, the variable w will not denote the error-correcting ability of the code (in contrast to the original algorithm in the paper [40]); since we have the variable t for this purpose.

Algorithm 1 - *Setup*(λ) **(Step 1)**

```
1: Input $\lambda$                          ▷ $\lambda$ = security parameter
2: Choose integers $n,\ k,\ d,\ t$          ▷ $n$ = length, $k$ = dimension, $d$ = minimum distance, $t$ = error-correcting ability of underpinning code.
3: Output $P = (n,\ k,\ d,\ t)$             ▷ $P$ = sysyem public parameter
```

Algorithm 2 - *KeyGen(P)* **(Step 2)**

```
                                   ▷ Step 2(i)
Input: $P = (n,\ k,\ d,\ t)$ = sysyem public parameter, $N$ = number of all possible signers.
Output: Send private key $sk_i = (Q_i, H_i, P_i, DEC_{H_i})$ to signer $S_i$ via a secure channel, for all $i = 1, 2, ..., N$.
 1: for $i = 1 : N$ do      ▷ For each signer $S_i$ in the ring $R = \{S_i | i = 1, 2, \ldots, N\}$, choose a parity-check matrix $H_i$ of a t-error correcting irreducible [n, k, d] Goppa code which has a corresponding fast decoding algorithm $DEC_{H_i}$, $i = 1, 2, \ldots, N$.
 2:     Call Algorithm 6 (Goppa Code Parity-Check Matrix construction) with the appropriate input (which is explained in Algorithm 6) and using $m \cdot t = n - k$, to define the $[n, k, d]$ binary Goppa Code, and get a Parity-Check Matrix $H_b$ of dimensions $(m \cdot t) \times n = (n - k) \times n$, whose elements are 0's and 1's. (In general, the inequality $m \cdot t \geq n - k$ is valid, since $k \geq n - m \cdot t$; however, we can choose parameters $m, t, n, k$ such that the equality $m \cdot t = n - k$ holds).
 3:     let $H_i := H_b$
 4:     Denote by $DEC_{H_i}$ the corresponding fast Decoding Algorithm for $H_i$ (Algorithm 9 - The Decoding process of binary Goppa Codes, using Patterson's Algorithm).
 5: end for
                                   ▷ Step 2(ii)
 6: for $i = 1 : N$ do      ▷ For each signer $S_i \in R$, choose a random binary $(n-k) \times (n-k)$ invertible matrix $Q_i$ and a random permutation $n \times n$ matrix $P_i$.
 7:     do
 8:     $Q_i = randi([0\ \ 1], n - k, n - k)$      ▷ (MATLAB command)
 9:     while $det(Q_i) = 0$
10:     $A = eye(n);$      ▷ (MATLAB command)
11:     $P_i = A(randperm(n), :);$      ▷ (MATLAB command)
12: end for
                                   ▷ Step 2(iii)
13: for $i = 1 : N$ do
14:     $\widetilde{H}_i = Q_i H_i P_i$      ▷ Dimensions of matrix product $Q_i H_i P_i$: $[(n - k) \times (n - k)] \cdot [(n - k) \times n] \cdot [n \times n] \Rightarrow$ dimensions of $\widetilde{H}_i$: $(n - k) \times n$
15: end for
                                   ▷ Step 2(iv)
16: $PK = [\,];$      ▷ Note: $PK$ is the name of a matrix ($PK \neq P \cdot K$ !!!)
17: for $i = 1 : N$ do
18:     $sk_i = (Q_i, H_i, P_i, DEC_{H_i})$      ▷ $sk_i$=private key, for each signer $S_i$
19:     $PK = [PK; \widetilde{H}_i];$      ▷ $PK$=ring public key; $PK = (\widetilde{H}_1, \widetilde{H}_2, ..., \widetilde{H}_N)$
20:     Send private key $sk_i = (Q_i, H_i, P_i, DEC_{H_i})$ to signer $S_i$ via a secure channel
21: end for
```

Algorithm 3 - $SignParticipant(\widetilde{m}, P, TSK)$ **(Step 3A)**

Input: message $\widetilde{m}$, system parameter $P = (n, k, d, t)$, N, $\widetilde{t}$, private key set $TSK = (sk_1, sk_2, ..., sk_{\widetilde{t}})$ of $\widetilde{t}$ true signers, where $sk_i = (Q_i, H_i, P_i, DEC_{H_i})$ for each $i = 1, 2, ..., \widetilde{t}$, and a one-way collision-resistant hash function $h : \{0,1\}^* \to \{0,1\}^{n-k}$ (for this, we can use SHA-256).

Output: Each signer S_i sends the following $N - \widetilde{t} + 1$ generated vectors $(e_i, e_{i(\widetilde{t}+1)}, e_{i(\widetilde{t}+2)}, ..., e_{iN})$ to the leader S_l, for all $i \in \{1, 2, ..., l-1, l+1, l+2, ..., \widetilde{t}\}$.

```
1: Randomly select an integer $l \in \{1, 2, ..., \widetilde{t}\}$    ▷ Randomly select a leader $S_l$ from the involved signer set $\{S_i | i = 1, 2, ..., \widetilde{t}\}$.
                                                    ▷ Steps 3(i), 3(ii), 3(iii) and 3(iv)
2: input a one-way collision-resistant hash function $h : \{0,1\}^* \to \{0,1\}^{n-k}$    ▷ Call SHA(256)
3: Initialize a $\widetilde{t} \times N$ collection $E$ of binary vectors (binary strings) of length $n$    ▷ The collection $E$ will hold the $e_{ij}$'s. Essentially, we only need $E$ to hold $(\widetilde{t}-1) \times (N-\widetilde{t})$ vectors; however, the notation is easier if we define $E$ to be of size $\widetilde{t} \times N$.
4: for $i \in \{1, 2, ..., l-1, l+1, l+2, ..., \widetilde{t}\}$ do    ▷ (parallelize)
5:     repeat
6:         for $j \in \{\widetilde{t}+1, \widetilde{t}+2, ..., N\}$ do    ▷ Randomly select $N - \widetilde{t}$ vectors $e_{ij} \in F_2^n$, $j = \widetilde{t}+1, \widetilde{t}+2, ..., N$ (i.e. select $(N-\widetilde{t})$ vectors $e_{ij}$, each one of size $1 \times n$ vector, whose elements are 0 or 1 and chosen at random)
7:             choose a $1 \times n$ vector $e_{ij}$ with random binary elements
8:         end for
9:             $s_i^T := h(\widetilde{m})^T + \sum_{j=\widetilde{t}+1}^{N} \widetilde{H}_j e_{ij}^T$    ▷ $s_i^T, h(\widetilde{m})^T, \widetilde{H}_j e_{ij}^T$ : $(n-k) \times 1$ vectors
                                                    $s_i, h(\widetilde{m})$ : $1 \times (n-k)$ vectors
10:            compute $Q_i^{-1} s_i^T$    ▷ $Q_i^{-1} s_i^T$: vector of size $(n-k) \times 1$
11:        *Call Patterson's Algorithm to check if $Q_i^{-1} s_i^T$ is a decodable syndrome*
12:    until $Q_i^{-1} s_i^T$ is a decodable syndrome
13:    *Compute $DEC_{H_i}(Q_i^{-1} s_i^T)$ to obtain a vector $e_i'$ such that $H_i e_i'^T = Q_i^{-1} s_i^T$*    ▷ $\Rightarrow e_i'$: vector of size $1 \times n$
14:    Compute $e_i^T = P_i^T e_i'^T$    ▷ $\Rightarrow e_i$: vector of size $1 \times n$
15:    Signer $S_i$ sends the following $N - \widetilde{t} + 1$ generated vectors to the leader $S_l$: $(e_i, e_{i(\widetilde{t}+1)}, e_{i(\widetilde{t}+2)}, ..., e_{iN})$    ▷ Output
16: end for
```

Algorithm 4 - $SignLeader(\widetilde{m}, P,\ TSK)$ **(Step 3B)**

Input: message $\widetilde{m}$, system parameter $P = (n, k, d, t)$, number of signers N, number of participants $\widetilde{t}$, private key set $TSK = (sk_1, sk_2, ..., sk_{\widetilde{t}})$ of $\widetilde{t}$ signers, where $sk_i = (Q_i, H_i, P_i, DEC_{H_i})$ for each $i = 1, 2, ..., \widetilde{t}$, and vectors $(e_i, e_{i(\widetilde{t}+1)}, e_{i(\widetilde{t}+2)}, ..., e_{iN})$ for all $i \in \{1, 2, ..., l-1, l+1, l+2, ..., \widetilde{t}\}$.
Output: σ, the threshold ring signature on the message $\widetilde{m}$.

```
                                                        ▷ Step 3(v)
 1: for $j \in \{\widetilde{t}+1, \widetilde{t}+2, ..., N\}$ do
 2:     choose a random $e_{l,j}$ such that $weight(e_{lj} + \sum_{i=1, i\neq l}^{\widetilde{t}} e_{ij}) = t$
 3:     $e_j := \sum_{i=1}^{\widetilde{t}} e_{ij}$
 4: end for
 5: if $\widetilde{t} = odd$ then $s_l^T = h(\widetilde{m})^T + \sum_{j=\widetilde{t}+1}^{N} \widetilde{H}_j e_{lj}^T$
 6: else $s_l^T = \sum_{j=\widetilde{t}+1}^{N} \widetilde{H}_j e_{lj}^T$          ▷ i.e. if $\widetilde{t} = even$
 7: end if
 8: compute $Q_l^{-1} s_l^T$
 9: if $Q_l^{-1} s_l^T$ is a decodable syndrome then compute $DEC_{H_l}(Q_l^{-1} s_l^T)$ to obtain a vector $e'_l$ such that $H_l e_l'^T = Q_l^{-1} s_l^T$.
10: else return to Step 3(v)
11: end if
12: Compute $e_l^T = P_l^T e_l'^T$
13: Output $\sigma = (e_1, e_2, ..., e_N)$     ▷ $\sigma =$ the threshold ring signature on the message $\widetilde{m}$
```

Algorithm 5 - $Verify(\widetilde{m}, PK, \sigma)$ **(Step 4)**

Input: the ring public key $PK = (\widetilde{H}_1, \widetilde{H}_2, ..., \widetilde{H}_N)$ and a message-signature pair $(\widetilde{m}, \sigma = (e_1, e_2, ..., e_N))$.
Output: $flag_1$ ($flag_1 = 1$ if σ is verified; 0 otherwise).

```
 1: $flag_1 := 1$;
 2: for $i = 1, 2, ..., N$ do
 3:     calculate $weight(e_i)$
 4:     if $weight(e_i) \neq t$ (i.e. if $e_i \notin \epsilon_{n,t}$) then     ▷ $length(e_i) = n$ by construction
 5:         $flag_1 := 0$;
 6:         break;
 7:     end if
 8:     if $(flag_1 = 1)$ & $\left(h(\widetilde{m})^T = \sum_{i=1}^{N} \widetilde{H}_i e_i^T\right)$ then
 9:         $flag_1 := 0$;
10:     end if
11: end for
12: Output $flag_1$     ▷ $flag_1 = 1$ if verified; 0 otherwise
```

Algorithm 6 - Goppa Code Parity-Check Matrix construction - Part 1 of 3

Input: The binary Goppa code $\Gamma(L, g(z))$ in the finite field $GF(2^m)$, with goppa (irreducible) polynomial $g(z) \in F_{2^m}[z]$ of $degree[g(z)] = t$, code-locator set $L = \{a_1, a_2, ..., a_n\} \subseteq F_{2^m}$ (with $g(a_i) \neq 0,\ \forall\ a_i \in L$)

▷ $\Gamma(L, g)$ is an irreducible [n, k, d] Goppa code, with $k \geq n - mt$ and $d \geq t + 1$. Furthermore, if $g(z)$ is separable (i.e. it has no roots of multiplicity > 1), then $d \geq 2t + 1$; therefore, $\Gamma(L, g)$ is a t-error correcting code. (Note that, in general, $k \geq n - m \cdot t \ \Rightarrow\ m \cdot t \geq n - k$. However, we choose parameters m, t, n, k such that the equality holds: $m \cdot t = n - k$).

Output: - The Parity-check matrix H_a of dimensions $t \times n$, with elements being polynomial functions of a.
- The Parity-check matrix H_b, which is a block-matrix of dimensions $(m \cdot t) \times n$, with elements being 0's and 1's.

More detailed input and setup of Goppa code:

• Input an integer $m \geq 3$ ▷ Typically $m \in \{10, 11, 12\}$ (see "Bernstein 2011 - List Decoding for Binary Goppa Codes")

• Input an integer t, with $2 \leq t \leq \frac{2^m - 1}{m}$ ▷ [Goppa code will be a degree-t code, designed to correct t errors. Choose intermediate values of t (not very small or very large values); e.g. for $m = 11$, choose $t = 32$ or $t = 70$ or $t = 100$]

• Fix $n = 2^m$ (or input n) ▷ [More generally, one can input/fix an integer n with $mt + 1 \leq n \leq 2^m$. This allows a better security/efficiency tradeoff for code-based cryptography. However this makes things more complicated. From now on, assume that $n = 2^m$.]

• We choose the extension field $GF(2^m)$, which is isomorphic to $GF(2)[X]/(k(X))$, for any irreducible polynomial $k(X)$ of $degree(k(X)) = m$.

• Choose an irreducible polynomial $k(X)$ of $degree(k(X)) = m$. To find such a polynomial, fully factorise $X^{n-1} - 1 = X^{2^m - 1} - 1 \mod 2$ and take as $k(X)$ any of its irreducible factors of degree m.

• We wish to search for a primitive element of $k(X)$. Let a be a root of $k(X)$ which is a primitive element. Therefore, $order(a) = n - 1$. Since $order(element) \mid order(group) \Rightarrow order(a) \mid n - 1$. Now, let $d_1, d_2, ..., d_\lambda$ be all the positive divisors

Algorithm 7 - Goppa Code Parity-Check Matrix construction - Part 2 of 3

of $n-1$, where without loss of generality we assume that $d_1 = 1 < d_2 < d_3 < ... < d_{\lambda-1} < d_\lambda = n-1$. Check that the following inequalities are satisfied: $a^{d_1} = a \neq 1$, $a^{d_2} \neq 1$, $a^{d_3} \neq 1$,...,$a^{d_{\lambda-1}} \neq 1$. If any of the above inequalities does not hold, then return to the previous step, to choose a different irreducible polynomial $k(X)$ of $degree(k(X)) = m$.

• Therefore, $a^{n-1} = 1$ and $order(a) = n-1$.

• $GF(2^m)^* =< a >= \{1, a, a^2, a^3, ..., a^{n-2}\}$ $\quad\rightarrow n-1$ elements

• $GF(2^m) = GF(n) = \{0, 1, a, a^2, a^3, ..., a^{n-2}\}$ $\quad\rightarrow n$ elements

• We represent each element of $GF(2^m)^*$ as a power of a, and then we use $k(a) = 0$ to rewrite every element of $GF(2^m)^*$ as a sum of powers of a, up to a^{m-1}, and as binary vectors. [Example: In $GF(2^4)^*$ with $m = 4, n = 2^4, k(X) = X^4 + X^3 + 1$, then $k(a) = 0 \Rightarrow a^4 + a^3 + 1 = 0 \Rightarrow a^4 = -a^3 - 1 \Rightarrow a^4 \equiv a^3 + 1\ mod(k(X))$. Therefore, e.g. $a^5 = a^4 \cdot a = (a^3 + 1) \cdot a = a^4 + a = a^3 + 1 + a = 1 + a + a^3 = 1 + 1 \cdot a + 0 \cdot a^2 + 1 \cdot a^3 = (1, 1, 0, 1)^T$]

• Now, choose a monic, binary, separable polynomial ('generator polynomial') $g(z)$ of $degree(g(z)) = t$ with $t \leq m-1$ (separable means that the number of its distinct roots is equal to the degree of the polynomial). Although not necessary, to simplify things we can choose $g(z)$ to be irreducible (we will make this assumption, from now on). Therefore, check if $g(z)$ is irreducible; otherwise choose a different $g(z)$. Write $g(z) = g_t z^t + g_{t-1} z^{t-1} + ... + g_2 z^2 + g_1 z + g_0$, where $g_t = 1$ (i.e. g(X) is 'monic').

• Note that we should have $g(z) \neq 0 \quad \forall z \in L$ (this should always be valid, by construction). There are no standard choices for $g(z) \in F_{2^m}[z]$. It is important for g to be a randomly chosen sercet.

• We can take the Support Set $L \subseteq GF(2^m)$ to be as large as possible. Therefore, let $L = GF(2^m) = \{0, 1, a, a^2, a^3, ..., a^{n-2}\} = \{a_1, a_2, a_3, a_4, ..., a_n\}$, where the *code locators* $a_1, a_2, ..., a_n$ are defined as: $a_1 = 0$, $a_2 = 1$ and $a_i = a^{i-2}$ for $i = 3, 4, 5, ..., n$ (i.e. $a_3 = a$, $a_4 = a^2$, $a_5 = a^3$, ..., $a_n = a^{n-2}$).

• Compute $g(a_i)^{-1} = \frac{1}{g(a_i)}$ for $i = 1, 2, ..., n$

1: Define:

$$T = \begin{bmatrix} g_t & 0 & 0 & ... & 0 \\ g_{t-1} & g_t & 0 & ... & 0 \\ g_{t-2} & g_{t-1} & g_t & ... & 0 \\ ... & ... & ... & ... & ... \\ g_1 & g_2 & g_3 & ... & g_t \end{bmatrix} \tag{3.1}$$

Algorithm 8 - Goppa Code Parity-Check Matrix construction - Part 3 of 3

```
2: Define:
```

$$X = \begin{bmatrix} \frac{1}{g(a_1)} & \frac{1}{g(a_2)} & \dots & \frac{1}{g(a_n)} & \\ \frac{a_1}{g(a_1)} & \frac{a_2}{g(a_2)} & \dots & \frac{a_n}{g(a_n)} & \\ \frac{a_1^2}{g(a_1)} & \frac{a_2^2}{g(a_2)} & \dots & \frac{a_n^2}{g(a_n)} & \\ \dots & \dots & \dots & \dots & \dots \\ \frac{a_1^{t-1}}{g(a_1)} & \frac{a_2^{t-1}}{g(a_2)} & \dots & \frac{a_n^{t-1}}{g(a_n)} & \end{bmatrix} \tag{3.2}$$

```
3: Define: $H_a = T \cdot X$
4: return $H_a$                  ▷ $H_a$ is a $t \times n$ Parity-Check Matrix, with elements being polynomial functions of $a$
5: $H_b$ is created from matrix $H_a$, if we replace every element of $H_a$ (which is a polynomial in $a$) into a binary column-vector, and adjust the dimensions of the block-matrix $H_b$ accordingly.
6: return $H_b$                  ▷ $H_b$ is an $(m \cdot t) \times n$ Parity-Check block matrix, with elements being 0's and 1's
```

Note: As an alternative to steps 2 and 3, the matrix H_a can equivalently be constructed as:

$$H_a = T \cdot V \cdot D$$

where T is the $t \times t$ Toeplitz matrix that was defined in Step 1, V is a $t \times n$ Vandermonde matrix, and D is an $n \times n$ diagonal matrix, which are defined specifically as:

$$V = \begin{bmatrix} 1 & 1 & \dots & 1 & \\ a_1 & a_2 & \dots & a_n & \\ a_1^2 & a_2^2 & \dots & a_n^2 & \\ a_1^3 & a_2^3 & \dots & a_n^3 & \\ \dots & \dots & \dots & \dots & \dots \\ a_1^{t-1} & a_2^{t-1} & \dots & a_n^{t-1} & \end{bmatrix} \tag{3.3}$$

$$D = \begin{bmatrix} \frac{1}{g(a_1)} & 0 & \dots & 0 & \\ 0 & \frac{1}{g(a_2)} & \dots & 0 & \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & \frac{1}{g(a_n)} & \end{bmatrix} \tag{3.4}$$

Algorithm 9 - *Decoding Algorithm (DEC_H):* The Decoding process of binary Goppa Codes, using Patterson's Algorithm - Part 1 of 2

Input: The received vector/codeword $y = (y_1, y_2, ..., y_n)$ with at most t errors, the binary Goppa code $\Gamma(L, g)$ with goppa polynomial $g(X) \in F_{2^m}[X]$ of $degree[g(X)] = t$, code-locator set $L = \{a_1, a_2, ..., a_n\} \subseteq F_{2^m}$ (with $g(a_i) \neq 0,\ \forall\ a_i \in L$), Parity-check matrix H.

Output: The original codeword $\widetilde{c}$ (original message), after correcting the errors.

Additional Output: Syndrome S, the error positions e and the error-locator polynomial $\sigma(X) \in F_{2^m}[X]$.

Notation: Assume $y := \widetilde{c} + e$, where $y = (y_1, y_2, ..., y_n)$ is received message ('word'), $\widetilde{c} = (\widetilde{c}_1, \widetilde{c}_2, ..., \widetilde{c}_n)$ is the original message and $e = (e_1, e_2, ..., e_n)$ is the error vector.
$\triangleright$ $\Gamma := \{c = (c_1, c_2, ..., c_n) \in F_2^n \mid R_c(X) := \sum_{i=1}^{n} \frac{c_i}{X - a_i} \equiv 0 \mod g(X)\}$
$\triangleright$ Γ is a linear $[n, k, d]$-code over the field F_2.

```
1: Compute the syndrome $S := y \cdot H^T$ for the word $y$      ▷ This is a simple matrix multiplication. Essentially, this calculates the syndrome polynomial $S(X) := R_y(X) := \sum_{i=1}^{n} \frac{y_i}{X - a_i}$ for the word $y$, where $S(X) \in F_{2^m}[X]$
2: Find $T(X) \equiv S^{-1}(X) \mod g(X)$      ▷ We seek a polynomial $T(X)$ such that $S(X)T(X) \equiv 1 \mod g(X)$. For this, we call the Extended Euclidean Algorithm (EEA) (Algorithm 13) with input $S(X), g(X)$ to calculate the $gcd\big(S(X), g(X)\big)$, which should be equal to 1. The output of the EEA will be: polynomials $T(X), v(X)$ such that $T(X) \cdot S(X) + v(X) \cdot g(X) = gcd\big(S(X), g(X)\big) = 1$. Therefore, $T(X) \cdot S(X) \equiv 1 \mod g(X)$. So we have the desired polynomial $T(X)$.
3: if T(X)=X then
4:     $\sigma(X) := X$
```

Algorithm 10 - *Decoding Algorithm (DEC_H):* The Decoding process of binary Goppa Codes, using Patterson's Algorithm - Part 2 of 2

```
 5: else
 6:     Compute $\tau_0(X) = T(X) + X$
 7:     Calculate $\tau(X) =$
        $= \sqrt{\tau_0(X)} \mod g(X)$                 ▷ For this, one can use $\sqrt{\tau_0(X)} = [\tau_0(X)]^{2^{mt-1}} \mod g(X)$
 8:     Calculate polynomials $A(X)$, $b(X)$ such that $A(X) \equiv b(X)\tau(X) \mod g(X)$,
        with $deg[A(X)] \leq \lfloor \frac{t}{2} \rfloor$ and $deg[b(X)] \leq \lfloor \frac{t-1}{2} \rfloor$.
                                                       ▷ For this, we call the modified EEA (Algorithm 14) with input $\tau(X)$, $g(X)$ as the polynomials $f_0$ and $f_1$, respectively.
                                                         In general, the modified EEA (Algorithm 14) is a variation of the EEA (Algorithm 13), with a stopping criterion such that the 'while-do' loop of Algorithm 13 terminates, as soon as the degree of $u_{k+1}(z)$ drops below $\lfloor \frac{t+1}{2} \rfloor$ for the first time; then the modified EEA returns the polynomials $u_k(z)$ and $v_k(z)$, without actually returning the gcd (which is not needed).
                                                         Here, the modified EEA returns the polynomial $u_k(z)$ as $A(z)$, and the polynomial $v_k(z)$ as $b(z)$ (obviously, we can then change the variable from $z$ to $X$ and get the desired polynomials $A(X)$ and $b(X)$).
                                                         The modified EEA is faster than the EEA, which would additionally calculate the $gcd\big(\tau(X), g(X)\big)$, which should be equal to 1 and is therefore not needed.
 9:     Find $\sigma(X) = A^2(X) + b^2(X) \cdot X$
10: end if
11: Find the roots of $\sigma(X)$ by brute force search. These will give the positions of the errors. For this, evaluate $\sigma(a_i)$ for all $i = 1, 2, ..., n$ and define the set of error locations $E = \{i | \sigma(a_i) = 0\}$
                                                       ▷ Alternatively, use Horner's method: $\sigma(X) = (((\sigma_t X + \sigma_{t-1})X + ...)X + \sigma_1)X + \sigma_0$
12: Define the error vector $e = (e_1, e_2, ..., e_n)$ by $e_i = 1$ if $i \in E$ and $e_i = 0$ otherwise, for $i = 1, ..., n$.
13: return the codeword $\widetilde{c} = y - e$          ▷ Flip the bits where an error occurs
```

Algorithm 11 - Syndrome Output

Input: An $[n, k, d]$-code C, its parity-check matrix H, a vector $x \in F_2^n$.
Output: Syndrome s (vector) ▷ where $s := s_H(x) \in F_2^{n-k}$

```
1: $s = x \cdot H^T$
2: return $s$
```

Algorithm 12 - Syndrome Decoding

Input: an integer t, a vector $s \in F_2^r$, a $r \times n$ random binary matrix H
Output: an $1 \times n$ vector e (i.e. a vector e of length n), such that $weight(e) = t$ and $He^T = s^T$

```
1: $e^T = H\backslash s^T$   [or equivalently, $e^T = mldivide(H, s^T)$]   ▷ (MATLAB command)
2: return $e$   ▷ Note: MATLAB solves the system $A \cdot x = b$ (sometimes by approximation) by using the command: $x = A\backslash b$ [or equivalently, $x = mldivide(A, b)$]
```

Algorithm 13 The Extended Euclidean Algorithm (EEA)

Input: Two polynomials $f_0(z)$ and $f_1(z)$ over $GF(q)$ such that $deg[f_0(z)] \geq deg[f_1(z)]$.
Output: A polynomial $f_k(z)$, which is equal to the $gcd[f_0(z), f_1(z)]$, and two polynomials $u_k(z)$ and $v_k(z)$ such that $gcd[f_0(z), f_1(z)] = f_k(z) = u_k(z)f_0(z) + v_k(z)f_1(z)$.

```
1: Define: $u_0(z) = 1$, $u_1(z) = 0$, $v_0(z) = 0$, $v_1(z) = 1$, $k = 0$.
2: while $f_{k+1}(z) \neq 0$ do
3:     k:=k+1
4:     Divide the polynomial $f_{k-1}(z)$ by $f_k(z)$, to find $q_k(z)$ and $r_k(z)$ such that:
       $f_{k-1}(z) = q_k(z)f_k(z) + r_k(z)$ and $deg[r_k(z)] < deg[f_k(z)]$
5:     Let
       $f_{k+1}(z) = r_k(z)$,
       $u_{k+1}(z) = u_{k-1}(z) - q_k(z)u_k(z)$,
       $v_{k+1}(z) = v_{k-1}(z) - q_k(z)v_k(z)$.
6: end while
7: return polynomials $f_k(z)$, $u_k(z)$, $v_k(z)$ such that:
           $gcd[f_0(z), f_1(z)] = f_k(z) = u_k(z)f_0(z) + v_k(z)f_1(z)$.
```

Algorithm 14 The modified Extended Euclidean Algorithm (modified EEA)

The *modified EEA* is a variation of the *EEA* (Algorithm 13), with a stopping criterion such that the 'while-do' loop of Algorithm 13 terminates, as soon as the degree of $u_{k+1}(z)$ drops below $\lfloor \frac{t+1}{2} \rfloor$ for the first time; then the *modified EEA* returns the polynomials $u_k(z)$ and $v_k(z)$, without actually returning the *gcd* (which is not needed).
The *modified EEA* is faster than the *EEA*.

4 Metrics Evaluation

4.1 Security Testing

The code-based Threshold Ring Signature Scheme proposed by Zhou et al.(2017)[40] was shown to satisfy the correctness, anonymity and unforgeability properties. Therefore, the scheme is proven to be secure against the scenarios described in the paper.

4.2 Efficiency Analysis

In this subsection, we discuss the efficiency of the proposed scheme, in terms of the public key size, the signature size, and the time complexity of the signing process (as presented in [40]).

The size of the Ring Public-Key $PK = (\widetilde{H}_1, \widetilde{H}_2, ..., \widetilde{H}_N)$ is $n(n-k)N$ bits, since each $\widetilde{H}_i$ is an $(n-k) \times n$ matrix over $\mathbb{F}_2$, $i = 1, 2, ..., N$.

The size of the signature $\sigma = (e_1, e_2, ..., e_N)$ is nN bits (where $e_i = \epsilon_{n,w}$, for $i = 1, 2, ..., N$).

Regarding the Time Complexity of the Signing Process, we assume that computing a hash function is a fast operation, compared to other operations involved in the $(\widetilde{t}, N)$ threshold ring signature scheme. For each $i = 1, 2, ..., \widetilde{t}$, each signer S_i computes a vector s_i, with time complexity $O\big((N-\widetilde{t})(n-k)n\big)$. The fast decoding algorithm has time complexity $O(n^2)$, according to Engelbert et al. (2007) [14]. We should execute $\widetilde{t}!$ decoding algorithms, on average, to generate a decodable syndrome, according to Courtois et al.(2001) [10]. Therefore, the total time complexity of the signing process in the scheme is: $2\widetilde{t}!\Big(O\big((N-\widetilde{t})(n-k)n\big) + O(n^2)\Big)$.

The time complexity of the signing process in this scheme is independent of the number of signers. The factor of the complexity of the proposed method is 2, insead of $\widetilde{t}$, in comparison to the CFS scheme [10]. This is because, with the exception of the leader, the remaining $\widetilde{t}-1$ signers can undertake concurrent operations in the proposed scheme. This parallelization enables the proposed code-based threshold ring signature scheme to be efficient.

4.3 Optimal Parameters for the McEliece Cryptosystem

In this subsection we present a summary table of parameters that we can use for the Goppa Codes (which are generally used in the the McEliece cryptosystem).

Niebuhr et al.(2012) [26] solved the problem of selecting optimal parameters for the McEliece cryptosystem that provide security until a given year and give detailed recommendations. Their analysis is based on the lower bound complexity estimates by Finiasz and Sendrier (2009) [16] and the security requirements model proposed by Lenstra and Verheul (2001) [20]. In Table 1 provided below, we can see the proposed parameters for the McEliece Cryptosystem, optimized for public key size, as found by Niebuhr et al. Specifically, the table shows the following information:

- Year: the year until which data security is required. Historic data is given mainly to allow comparison with other sources.

Table 1: Proposed parameters for the McEliece Cryptosystem - optimized for public key size (from Niebuhr et al.(2012) [26])

Year	Symmetric Key Size	Lower bound for $\log_2 S(n,k,t)$	McEliece parameters (n,k,t) and	public key size (kB)	RSA Key Size and	SDL[a] Field Size	Elliptic Curve Key Size (bits) for $c = 0$,	$c = 18$	Infeasible number of MIPS-years	Corresponding number of years on a 2.4 GHz Intel Core 2 Quad Q6600
2009	77	83	(1634, 1217, 39)	62	1323	1024	145	157	$8.52 \cdot 10^{11}$	$1.94 \cdot 10^{7}$
2010	78	84	(1635, 1197, 41)	64	1369	1056	146	160	$1.45 \cdot 10^{12}$	$3.30 \cdot 10^{7}$
2011	79	85	(1652, 1203, 42)	66	1416	1088	148	163	$2.47 \cdot 10^{12}$	$5.61 \cdot 10^{7}$
2012	80	87	(1687, 1226, 43)	69	1464	1120	149	165	$4.19 \cdot 10^{12}$	$9.52 \cdot 10^{7}$
2013	80	88	(1702, 1219, 45)	72	1513	1184	151	168	$7.14 \cdot 10^{12}$	$1.62 \cdot 10^{8}$
2014	81	89	(1770, 1306, 43)	74	1562	1216	152	172	$1.21 \cdot 10^{13}$	$2.75 \cdot 10^{8}$
2015	82	90	(1823, 1368, 42)	76	1613	1248	154	173	$2.07 \cdot 10^{13}$	$4.70 \cdot 10^{8}$
2016	83	91	(1833, 1356, 44)	79	1664	1312	155	177	$3.51 \cdot 10^{13}$	$7.98 \cdot 10^{8}$
2017	83	92	(1845, 1356, 45)	81	1717	1344	157	180	$5.98 \cdot 10^{13}$	$1.36 \cdot 10^{9}$
2018	84	93	(1877, 1387, 45)	83	1771	1376	158	181	$1.02 \cdot 10^{14}$	$2.32 \cdot 10^{9}$
2019	85	95	(1951, 1481, 43)	85	1825	1440	160	185	$1.73 \cdot 10^{14}$	$3.93 \cdot 10^{9}$
2020	86	96	(1955, 1463, 45)	88	1881	1472	161	188	$2.94 \cdot 10^{14}$	$6.68 \cdot 10^{9}$
2021	86	97	(1983, 1479, 46)	91	1937	1536	163	190	$5.01 \cdot 10^{14}$	$1.14 \cdot 10^{10}$
2022	87	98	(2013, 1508, 46)	93	1995	1568	164	193	$8.52 \cdot 10^{14}$	$1.94 \cdot 10^{10}$
2023	88	99	(2018, 1491, 48)	96	2054	1632	166	197	$1.45 \cdot 10^{15}$	$3.30 \cdot 10^{10}$
2024	89	101	(2104, 1596, 46)	99	2113	1696	167	198	$2.47 \cdot 10^{15}$	$5.61 \cdot 10^{10}$
2025	89	102	(2106, 1576, 48)	102	2174	1728	169	202	$4.20 \cdot 10^{15}$	$9.55 \cdot 10^{10}$
2026	90	103	(2135, 1604, 48)	104	2236	1792	170	205	$7.14 \cdot 10^{15}$	$1.62 \cdot 10^{11}$
2027	91	104	(2157, 1614, 49)	107	2299	1856	172	207	$1.21 \cdot 10^{16}$	$2.75 \cdot 10^{11}$
2028	92	105	(2198, 1654, 49)	110	2362	1888	173	210	$2.07 \cdot 10^{16}$	$4.70 \cdot 10^{11}$
2029	93	106	(2220, 1664, 50)	113	2427	1952	175	213	$3.52 \cdot 10^{16}$	$8.00 \cdot 10^{11}$
2030	93	108	(2241, 1673, 51)	116	2493	2016	176	215	$5.98 \cdot 10^{16}$	$1.36 \cdot 10^{12}$
2032	95	110	(2344, 1784, 50)	122	2629	2144	179	222	$1.73 \cdot 10^{17}$	$3.93 \cdot 10^{12}$
2034	96	112	(2440, 1877, 50)	129	2768	2272	182	227	$5.01 \cdot 10^{17}$	$1.14 \cdot 10^{13}$
2036	98	115	(2496, 1920, 51)	135	2912	2400	185	232	$1.45 \cdot 10^{18}$	$3.30 \cdot 10^{13}$
2038	99	117	(2440, 1776, 59)	144	3061	2528	188	239	$4.20 \cdot 10^{18}$	$9.55 \cdot 10^{13}$
2040	101	119	(2521, 1854, 59)	151	3214	2656	191	244	$1.22 \cdot 10^{19}$	$2.77 \cdot 10^{14}$
2042	103	122	(2623, 1964, 58)	158	3371	2784	194	248	$3.52 \cdot 10^{19}$	$8.00 \cdot 10^{14}$
2044	104	124	(2662, 1979, 60)	165	3533	2944	197	255	$1.02 \cdot 10^{20}$	$2.32 \cdot 10^{15}$
2046	106	126	(2691, 1973, 63)	173	3700	3072	200	260	$2.95 \cdot 10^{20}$	$6.70 \cdot 10^{15}$
2048	107	129	(2798, 2088, 62)	181	3871	3232	203	265	$8.53 \cdot 10^{20}$	$1.94 \cdot 10^{16}$
2050	109	131	(2804, 2048, 66)	189	4047	3392	206	272	$2.47 \cdot 10^{21}$	$5.61 \cdot 10^{16}$

[a] Subgroup Discrete Logarithm

- Symmetric key size: the symmetric key size required to ensure data security, calculated in accordance with Lenstra and Verheul's approach [20].

- Lower bound for $log_2(S(n,k,t))$: the log_2 of the minimum number of binary operations (required to break a McEliece cryptosystem) that are infeasible in the respective year.

- RSA and EC parameters: the original parameters proposed by Lenstra and Verheul [20]. They allow for easy comparison between classical and code-based cryptosystems.

- The last two columns are a translation of the required symmetric key size into parameters relevant in practice, i.e. the number of MIPS (million instructions per second) years that render a cryptosystem infeasible to break, and the corresponding number of years on a modern Quad core CPU.

Note that this table refers to a binary $[n, k]$ Goppa code $\mathscr{C}$ defined by a Goppa polynomial of degree t, together with a fast decoding algorithm that can correct up to t errors.

5 Other Similar Schemes

As discussed before, since 2001 when Rivest et al. [29] first formally introduced the ring signature, there have been a number of ring signature schemes proposed in the literature.

Bresson et al. (2002) [6] extended the notion of ring signatures into *threshold* ring signatures, which became quite popular due to their practical applications - compared to the conventional ring signatures. Existing threshold ring signature schemes are mostly based on the number theory (e.g. see [28], [35], [38], [18]) and could be insecure in the quantum world.

On the other hand, *code-based* threshold ring signature schemes were proposed by Dallot and Vergnaud (2009) [12], Aguilar Melchor et al (2011) [24] and Cayrel et al (2012) [7]. Specifically, Dallot and Vergnaud's scheme (2009) [12] combined Bresson et al.'s construction [6] and Courtois et al.'s signature [10], which results in the signature size twice the number of system users. Aguilar Melchor et al.'s scheme (2011) [24] is a generalization of Stern's identification and signature scheme [33] and has low efficiency in the signature size. Cayrel et al. (2012) [7] proposed a threshold ring signature scheme build on the $q-$ary syndrome decoding identification scheme that was proposed by Cayrel et al. (2010) [8].

In Table 2 we present the summary results found by Cayrel et al. (2012) [7] in terms of key sizes, signature length and signing cost for the parameter set $(t, N) = (50, 100)$, comparing the code-based threshold ring signature schemes of Aguilar Melchor et al. [24], Dallot and Vergnaud [12] and Cayrel et al. [7].

Cayrel et al. (2012) [7] proposes the following parameters for the q-SD scheme, in accordance to the Information Set Decoding (ISD) algorithm: $q = 256, n = 128, n - k = 64$ (which implies that $k = 64$) and $\widetilde{t} = 49$ (error-correction ability). More details of the comparison between the three schemes, as well as the relevant parameters used, can be found in [7].

Finally, it is worth discussing in detail the complexity of an existing scheme which is similar to the proposed one. In Aguilar Melchor et al. [24], the signature length of the proposed protocol is $O(N)$; more precisely around $20ko \times N$, for $20ko$ the length of one signature by the Fiat-Shamir paradigm [15] applied to the Stern scheme (a security of 2^{-80}) is obtained by 140 repetitions of the protocol). For instance, considering a particular example with $N = 100$ and $t = 50$, they obtain a $2Mo$ signature length, which is quite large, but still tractable. Note that other number theory based protocols have shorter signature lengths (in $8ko$ or $25ko$) but are slower. They obtain a public key size in $347N$. The cost of their protocol is N times the cost of one Stern signature protocol, hence in $O(N)$ (more precisely, in $140n^2N$ operations), for any t. This improved previous fully anonymous threshold ring signature protocols that had a complexity in $O(tN)$ operations (multiplications or modular exponentiations in large integer rings, or pairings).

Table 2: Comparison of other code-based threshold ring signature schemes

Threshold Ring Singature Scheme	**Public Key Size in KBytes**	**Signature Size in KBytes**	**Signature cost in bops**
Aguilar Melchor et al. [24]	1470	2448	2^{30}
Dallot and Vergnaud [12]	10137122	7	2^{35}
Cayrel et al. [7]	400	2384	2^{26}

6 Conclusions

The code-based Threshold Ring Signature Scheme under study satisfies correctness, unforgeability and anonymity. In comparison to other postquantum digital signature schemes, this scheme has a lower signature size. Moreover, the parallelization feature of the scheme reduces significantly the time complexity of the signing process, thus making it more effective, compared to previous signature schemes.

References

[1] Abe, M., Ohkubo, M., & Suzuki, K. (2002, December). 1-out-of-n signatures from a variety of keys. In International Conference on the Theory and Application of Cryptology and Information Security (pp. 415-432). Springer, Berlin, Heidelberg.

[2] Adida, B., Hohenberger, S., & Rivest, R. L. (2005, June). Ad-hoc-group signatures from hijacked keypairs. In in DIMACS Workshop on Theft in E-Commerce.

[3] Bender, A., Katz, J., & Morselli, R. (2006, March). Ring signatures: Stronger definitions, and constructions without random oracles. In Theory of Cryptography Conference (pp. 60-79). Springer, Berlin, Heidelberg.

[4] Berlekamp, E., McEliece, R., & Van Tilborg, H. (1978). On the inherent intractability of certain coding problems (Corresp.). IEEE Transactions on Information Theory, 24(3), 384-386.

[5] Boneh, D., Gentry, C., Lynn, B., & Shacham, H. (2003, May). Aggregate and verifiably encrypted signatures from bilinear maps. In International Conference on the Theory and Applications of Cryptographic Techniques (pp. 416-432). Springer, Berlin, Heidelberg.

[6] Bresson, E., Stern, J., & Szydlo, M. (2002, August). Threshold ring signatures and applications to ad-hoc groups. In Advances in Cryptology, Lecture Notes in Comput. Sci., pp. 465-480, Springer, Berlin, Germany, 2002.

[7] Cayrel, P. L., Alaoui, S. M. E. Y., Hoffmann, G., & Veron, P. (2012, July). An improved threshold ring signature scheme based on error correcting codes. In International Workshop on the Arithmetic of Finite Fields (pp. 45-63). Springer, Berlin, Heidelberg.

[8] Cayrel, P. L., Veron, P., & El Yousfi Alaoui, S.M. (2010, August). A zero-knowledge identification scheme based on the q-ary syndrome decoding problem. In International Workshop on Selected Areas in Cryptography (pp. 171-186). Springer, Berlin, Heidelberg.

[9] Chaum, D., & Van Heyst, E. (1991, April). Group signatures. In Workshop on the Theory and Application of of Cryptographic Techniques (pp. 257-265). Springer, Berlin, Heidelberg.

[10] Courtois, N. T., Finiasz, M., & Sendrier, N. (2001, December). How to achieve a McEliece-based digital signature scheme. In International Conference on the Theory and Application of Cryptology and Information Security (pp. 157-174). Springer, Berlin, Heidelberg.

[11] Cover, T. M., & Thomas, J. A. (2012). Elements of information theory. John Wiley & Sons.

[12] Dallot, L., & Vergnaud, D. (2009, December). Provably secure code-based threshold ring signatures. In IMA International Conference on Cryptography and Coding (pp. 222-235). Springer, Berlin, Heidelberg.

[13] Dodis, Y., Kiayias, A., Nicolosi, A., & Shoup, V. (2004, May). Anonymous identification in ad hoc groups. In International Conference on the Theory and Applications of Cryptographic Techniques (pp. 609-626). Springer, Berlin, Heidelberg.

[14] Engelbert, D., Overbeck, R., & Schmidt, A. (2007). A summary of McEliece-type cryptosystems and their security. Journal of Mathematical Cryptology JMC, 1(2), 151-199.

[15] Fiat, A., & Shamir, A. (1986, August). How to prove yourself: Practical solutions to identification and signature problems. In Advances in Cryptology-CRYPTO'86 (pp. 186-194). Springer, Berlin, Heidelberg.

[16] Finiasz, M., & Sendrier, N. (2009, December). Security bounds for the design of code-based cryptosystems. In International Conference on the Theory and Application of Cryptology and Information Security (pp. 88-105). Springer, Berlin, Heidelberg.

[17] Herranz, J., & Saez, G. (2003, December). Forking lemmas for ring signature schemes. In International Conference on Cryptology in India (pp. 266-279). Springer, Berlin, Heidelberg.

[18] Hu, X., ZhiGuang, Q., & FaGen, L. (2008, May). Identity-based threshold ring signature without pairings. In Communications, Circuits and Systems, 2008. ICCCAS 2008. International Conference on (pp. 478-482). IEEE.

[19] Jochemsz, Ellen (2002). Goppa Codes & the McEliece Cryptosystem. Amsterdam: Vrije Universiteit Amsterdam, 2002. Print.

[20] Lenstra, A. K., & Verheul, E. R. (2001). Selecting cryptographic key sizes. Journal of cryptology, 14(4), 255-293.

[21] Liu, J. K., Wei, V. K., & Wong, D. S. (2004, July). Linkable spontaneous anonymous group signature for ad hoc groups. In Australasian Conference on Information Security and Privacy (pp. 325-335). Springer, Berlin, Heidelberg.

[22] MacKay, D. J., & Mac Kay, D. J. (2003). Information theory, inference and learning algorithms. Cambridge university press.

[23] McEliece, R. J. (1978). A public-key cryptosystem based on algebraic. Coding Thv, 4244, 114-116.

[24] Melchor, C. A., Cayrel, P. L., Gaborit, P., & Laguillaumie, F. (2011). A new efficient threshold ring signature scheme based on coding theory. IEEE Transactions on Information Theory, 57(7), 4833-4842.

[25] Naor, M. (2002, August). Deniable ring authentication. In Annual International Cryptology Conference (pp. 481-498). Springer, Berlin, Heidelberg.

[26] Niebuhr, R., Meziani, M., Bulygin, S., & Buchmann, J. (2012). Selecting parameters for secure McEliece-based cryptosystems. International Journal of Information Security, 11(3), 137-147.

[27] Overbeck, R., & Sendrier, N. (2006). Code-based Cryptography. Bernstein, Daniel J.(ed.) et al., Post-quantum cryptography. In First international workshop PQCrypto (pp. 23-26).

[28] Petzoldt, A., Bulygin, S., & Buchmann, J. (2013). A multivariate based threshold ring signature scheme. Applicable Algebra in Engineering, Communication and Computing, 24(3-4), 255-275.

[29] Rivest, R. L., Shamir, A., & Tauman, Y. (2001). How to leak a secret. In Advances in Cryptology-ASIACRYPT, vol. 2248 of Lecture Notes in Comput. Sci., pp. 552-565, Springer.

[30] Ryan, W., & Lin, S. (2009). Channel codes: classical and modern. Cambridge University Press.

[31] Safieddine, R. and Desmarais, A. (2014). Comparison of Different Decoding Algorithms for Binary Goppa Codes. Saint-Etienne, France. Unpublished manuscript.

[32] Soo H. Go (2012). Sparse Polynomial Interpolation and the Fast Euclidean Algorithm (Master's thesis, Simon Fraser University). Retrieved from http://www.cecm.sfu.ca/CAG/theses/soogo.pdf

[33] Stern, J. (1996). A new paradigm for public key identification. IEEE Transactions on Information Theory, 42(6), 1757-1768.

[34] Valentijn, A. (2015). Goppa Codes and Their Use in the McEliece Cryptosystems.

[35] Wang, H., & Han, S. (2010, August). A provably secure threshold ring signature scheme in certificateless cryptography. In Information Science and Management Engineering (ISME), 2010 International Conference of (Vol. 1, pp. 105-108). IEEE.

[36] Wang, K., Mu, Y., & Susilo, W. (2015). Identity-based quotable ring signature. Information Sciences, 321, 71-89.

[37] Xu, J., Zhang, Z., & Feng, D. (2004, August). A ring signature scheme using bilinear pairings. In International Workshop on Information Security Applications (pp. 160-169). Springer, Berlin, Heidelberg.

[38] Yuen, T. H., Liu, J. K., Au, M. H., Susilo, W., & Zhou, J. (2011, March). Threshold ring signature without random oracles. In Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security (pp. 261-267). ACM.

[39] Zhang, F., & Kim, K. (2002, December). ID-based blind signature and ring signature from pairings. In International Conference on the Theory and Application of Cryptology and Information Security (pp. 533-547). Springer, Berlin, Heidelberg.

[40] Zhou, G., Zeng, P., Yuan, X., Chen, S., & Choo, K.K.R. (2017). An Efficient Code-Based Threshold Ring Signature Scheme with a Leader-Participant Model. Security and Communication Networks, 2017.