A Democratic Distributed Post-Quantum Certificateless Encryption Scheme[⋆]

Thomas Prévost[1] [0009−0000−2224−8574], Bruno Martin[1] [0000−0002−0048−5197], and Olivier Alibart[2] [0000−0003−4404−4067]

[1] Université Côte d'Azur, CNRS, I3S, France
{thomas.prevost,bruno.martin}@univ-cotedazur.fr
[2] Université Côte d'Azur, CNRS, InPhyNi, France
olivier.alibart@univ-cotedazur.fr

Abstract. We propose a post-quantum certificateless encryption scheme based on a web of trust instead of a centralized Key Generation Center. Our scheme allows nodes to communicate securely. It is the nodes already present in the network that vote on the acceptance of new nodes, and agree on the shared key. The threshold required for the acceptance of a new node is configurable. Our protocol thus allows to completely operate without the Key Generation Center (or Key Distribution Center).
Our scheme is based on Quasi-Cyclic Moderate Density Parity Check Code McEliece, which is resistant to quantum computer attacks. The voting system uses Shamir secret sharing, coupled with the Kabatianskii-Krouk-Smeets signature scheme, both are also resistant to quantum computer attacks.
We provide a security analysis of our protocol, as well as a formal verification and a proof of concept code.

Keywords: Certificateless encryption · QC-MDPC McEliece · Post-quantum encryption · Web of trust · Distributed Identity Management · KKS.

1 Introduction

Certificateless encryption is designed as an extension of Identity-based encryption. In this scheme, invented by Shamir in 1984 [40], the public key is directly derived from an identifier that uniquely designates a user (IP address, email address, domain name, etc.). Thus, anyone is able to verify a public key from publicly accessible information, without resorting to a trusted authority. However, anyone is then able to generate a key pair for a given identity.

The certificateless encryption scheme [2] attempts to solve this problem by submitting the generation of a key pair by a new participant to the approval of a trusted third party, the Key Generation Center (KGC), also called the Key

⋆ This work has been supported by a government grant managed by the Agence Nationale de la Recherche under the Investissement d'avenir program, reference ANR-17-EURE-004

Distribution Center (KDC). All nodes in the network are thus able to ensure that a public key has been generated with the approval of the KGC, which then acts as a trusted intermediary. The KGC cannot, however, know the generated private key.

In the classical certificateless encryption scheme, when a new node wants to join the network, and therefore generates a key pair based on its identity, it will ask the Key Generation Center for permission. The KGC will then broadcast to all nodes already present in the network a random value, the partial private key (psk). The KGC will also transmit the psk to the new node. This new node will then generate a key pair from the partial private key and its identity. Thus, all nodes in the network will be able to ensure that the generated public key corresponds to both the identity of the new node and the partial private key generated by the KGC.

Our contribution In this paper, we propose a new certificateless encryption protocol in which the Key Generation Center is replaced by a direct agreement of the nodes already present in the network. Thus, it is no longer necessary to trust a single KGC, but a web of trust of existing nodes.

The threshold of votes required for a new node to be considered approved will depend on the trust placed in the network, i.e. the estimated number of potentially malicious nodes. In the Bitcoin network, six approvals are enough for a transaction to be considered secure [29], while Condorcet showed that two-thirds of the votes guarantee a decision that is very likely to be good [17].

Our protocol is based on the McEliece cryptosystem [33], which is currently considered secure against quantum computer attacks. The certificateless encryption used in this paper is based on [31].

This paper is organized as follows: first, in section 2, we recall how McEliece's cryptosystem works, and in section 3, the particular case of Quasi-Cyclic Low Density Parity Check codes (QC-LDPC) and Moderate Density Parity Check codes (QC-MDPC). In section 4, we explain how to develop a digital signature system from error-correcting codes, and especially how to achieve this without performing a decoding operation with the scheme proposed by Kabatianskii, Krouk and Smeets (KKS) [27]. In section 5, we show how the secret sharing scheme developed by Shamir [39] works. Next, in section 6, we detail the operation of our post-quantum distributed certificateless encryption protocol. We also provide a proof-of-concept written in Rust in section 7. Afterwards, we analyze the security properties of our protocol (including formal verification using ProVerif) in section 8, as well as its performances, both in terms of computing power required and number of messages sent (section 9). Finally, we conclude with the prospects for using and improving our protocol.

Related work [10] proposes a similar certificateless encryption protocol, in which the KGC is replaced by a vote of the nodes already present in the network. This cryptosystem is based on threshold group signature. [30] also proposes a certificateless encryption scheme without KGC, by distributing the private key of the KGC which would exist in a classic certificateless encryption in shares, between the existing nodes. [26] proposes something analogous but with approval by other nodes based on a PGP-like system. [35] proposes a more original certificateless encryption solution to avoid the problem of malicious KGC, with several KGCs having to work simultaneously to accept a new node. Another solution proposed by [19] is to record the approval of new nodes by the KGC on a blockchain, so that all nodes in the system can control this approval.

2 McEliece cryptosystem

2.1 Linear error correction code

Error correction codes were proposed by Richard Hamming in 1950 [25]. They allow messages to be transmitted over a noisy channel, correcting at most t errors per word of k symbols. Here we are only interested in binary codes, so the symbols are defined on the alphabet $\{0, 1\}$. Error-correction codes differ from checksums in that they not only detect the existence of a transmission error, but also correct it to find the original word. Error correcting codes are used in noisy communication channels, disks (which can be scratched), Random Access Memories [37] or even in quantum error correction [11].

An (n, k, t)-error-correction code is a code of length n that can correct at most t errors on words of length k. Most codes used in real-world applications are *linear*.

Let C be a binary (n, k, t)-linear code. C is a subspace of $\mathbb{F}_2^n$. Furthermore, $\forall u, v \in C, u + v \in C$ (linearity) and $dim(C) = k$.

Moreover, the subspace C is equipped with the Hamming distance h^*. Let $u, v \in C$, the Hamming distance between u and v, denoted $h^*(u, v)$, is defined as the number of bits of u different from v. For example, if $u = 0\mathbf{10}0$ and $v = 0\mathbf{01}0$, then $h^*(u, v) = h^*(v, u) = 2$, since the second and third bits of u and v differ. Obviously, $h^*(u, u) = 0$. Let us denote by $w_h(x)$ the Hamming weight of word x, which is $h^*(0, x)$. Two different words of an (n, k, t)-linear error-correction code C have a Hamming distance greater than t:

$$\forall u, v \in C, u \neq v \Longleftrightarrow h^*(u, v) > t \tag{1}$$

Let $\mathcal{M}_{\{0,1\}}^{k\times n}$ denote the set of binary matrices with k rows and n columns. A (n, k, t)-linear error correction code C is defined by its generator matrix in the canonical basis $G \in \mathcal{M}_{\{0,1\}}^{k\times n}$, such that $\forall x \in \mathbb{F}_2^k, x \cdot G \in C \subset \mathbb{F}_2^n$.

In order to detect and correct possible errors, we use a linear map $\eta : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^{n-k}$, such that $ker(\eta) = C$ (meaning $C = \eta^{-1}(0_{\mathbb{F}_2^{n-k}})$), with $0_{\mathbb{F}_2^{n-k}}$ the null binary vector of dimension $n-k$. The matrix of η in the canonical basis, denoted H, is the *parity-check matrix*:

$$\forall x \in \mathbb{F}_2^n, H^T \cdot x = 0_{\mathbb{F}_2^{n-k}} \Longleftrightarrow x \in C \tag{2}$$

With H^T being the transpose of the matrix H.

Definition 1. *For any word with error $z = x + e \in \mathbb{F}_2^n$, with $x \in C$ and e error vector, we call* syndrome *of x the element*

$$S = H^T \cdot z = H^T \cdot x + H^T \cdot e = 0_{\mathbb{F}_2^{n-k}} + H^T \cdot e = H^T \cdot e \tag{3}$$

Since all the words of C are spaced by a Hamming distance greater or equal than $2t + 1$, then we can correct at most t errors on the word x by determining the word x' having the lowest possible Hamming distance from x such that $H^T \cdot x' = 0_{\mathbb{F}_2^{n-k}}$. There are several algorithms for this, with a complexity depending on the code used [3,32]. Most of these algorithms gain in efficiency by being probabilistic, that is to say they decode a message with at most t errors with a very high probability.

2.2 McEliece public key encryption

McEliece had the idea in 1978 to use (n, k, t)-error correction codes to develop a public key cryptosystem [33]. This cryptosystem works as follows:

KeyGen

- The generator matrix G, described above, is the public key.
- The parity check matrix H is the private key

Enc To encode a message $m \in \mathbb{F}_2^k$, the participant generates a random error vector $e \in \mathbb{F}_2^n$ such that $w_h(e) = t$. The one participant then generates a ciphertext $c \in \mathbb{F}_2^n$ with the operation:

$$c = x \cdot G + e \tag{4}$$

Dec We can find the message m from the ciphertext c, by decoding the errors using the parity-check matrix H that remained secret.

Currently, McEliece's cryptosystem is considered secure against quantum computer attacks [13]. Its main problem, however, remains the gigantic size of the keys, generally several tens of KB.

3 QC-LDPC and QC-MDPC

3.1 Circulant matrix

Definition 2. *Let $M \in \mathcal{M}_{\mathbb{F}_2}^{p \times p}$ be a binary square matrix. We say that M is a* circulant *matrix if*

$$M = \begin{pmatrix} m_0 & m_1 & \dots & m_{p-1} \\ m_{p-1} & m_0 & \dots & m_{p-2} \\ \vdots & \vdots & \ddots & \vdots \\ m_1 & m_2 & \dots & m_0 \end{pmatrix} \tag{5}$$

with $m_0, m_1, \dots, m_{p-1} \in \mathbb{F}_2$.

A circulant matrix M can therefore be represented by its first row vector $m = (m_0, m_1, \ldots, m_{p-1}) \in \mathbb{F}_2^p$. m is the characteristic vector of M. This allows to significantly reduce the size of the information that needs to be sent over the network (actually a quadratic reduction).

The row weight ω of M is the Hamming weight of m: $\omega = w_h(m)$.

Let $M, N \in \mathcal{M}_{\mathbb{F}_2}^{p \times p}$ be circulant matrices:

1. $M + N$ is circulant.
2. $M \cdot N$ is circulant.
3. M^T is circulant (recall that M^T is the transpose of matrix M).

As explained by [18], the algebra of binary circular matrices can be reduced to the algebra of binary polynomials in the ring $\mathcal{R}_p = \mathbb{F}_2[X]/(X^p - 1)$. The matrix above would then be represented by the polynomial $P = m_0 + m_1 \cdot X + \cdots + m_{p-1} \cdot X^{p-1} \in \mathcal{R}_p$. Since the polynomial $X^p - 1$ is itself reducible, then the ring $\mathcal{R}_p$ does not have the characteristics of a field, in particular not all non-zero polynomials in $\mathcal{R}_p$ are necessarily invertible: not all binary circulant matrices are invertible. Addition and multiplication are commutative. Reducing the space of matrices $\mathcal{M}_{\mathbb{F}_2}^{p \times p}$ to the ring of polynomials $\mathcal{R}_p$ allows to speed up computations enormously.

We developed a software library to perform computations on these binary polynomials, available at `https://github.com/thomasarmel/binary_polynomial_mod_algebra`.

3.2 Construction of QC-LDPC and QC-MDPC codes

Quasi-Cyclic codes (QC)

Definition 3. *An error correction code C is said to be Quasi-Cyclic (QC) if its parity-check matrix H is a concatenation of matrices $H = [H_{i,j}]$, with each $H_{i,j}$ being a circulant matrix.*

In general, we construct a Quasi-Cyclic code by defining its parity-check matrix H of the form $H = [H_0| \ldots |H_l]$, with $H_0, \ldots, H_l$ circulant matrices.

The row weight ω of the parity matrix H is then defined as the sum of the row weights of the matrices $H_0, \ldots, H_l$.

Low-Density Parity Check codes (LDPC)

Definition 4. *An (n, k, t)-error correction code is* Low-Density Parity Check *(LDPC) when its parity-check matrix is* sparse*, i.e. its row weight ω is low. In other words, the matrix row weight $\omega = O(1)$ is negligible compared to n, usually $\omega < 10$.*

These codes were proposed by Gallager in [22], along with an efficient probabilistic decryption algorithm.

Moderate-Density Parity Check codes (MDPC) The McEliece encryption variant based on *Moderate-Density Parity Check* matrices (MDPC) was proposed by [34] in order to avoid known plaintext attacks as well as key recovery attacks.

This involves increasing the row weight ω of the parity-check matrix H.

Definition 5. *An (n, k, t)-error correction code is* Moderate-Density Parity Check *(MDPC) when the row weight ω of its parity-check matrix is* moderate*, i.e. $\omega = O(\sqrt{n \cdot log(n)})$.*

To ensure 128-bit security, [34] proposes the following security parameters: a code length n of 19714 bits, a dimension k of 9857 bits ($n = 2k$), a row weight ω of 142 bits, and an error count t of 134 bits.

Gallager's decoding algorithm can also work on MDPC codes.

4 Kabatianskii-Krouk-Smeets signature scheme (KKS)

4.1 The problem with classical correcting codes signature schemes

McEliece's cryptosystem, while it allows asymmetric encryption operations to be performed simply, is not particularly suited to digital signatures. Most signature algorithms based on McEliece [16] consider the message to be signed w_e as a codeword with errors: the signature operation then consists of exhibiting the decrypted word x. The verifier, which possesses the generator matrix G, is then able to verify that the signer possesses the correct parity check matrix H, if $w_e = x \cdot G + e$, with $w_h(e) \leq t$.

The problem with this method is that the Hamming distance h^* between two distinct codewords u, v can be much greater than the error correction capacity of the code:

$$\forall u, v \in C, u \neq v, h^*(u, v) \gg t \tag{6}$$

The signer is then unable to find the valid codeword $w \in C$ that is the closest to w_e, and therefore to exhibit the decoded word x. This is called the *complete decoding problem.*

We are in this case: the density of the parity matrix H being too high, the distance between two valid codewords is too large. We must then find another solution to sign the messages.

4.2 A signature scheme without decoding

Kabatianskii, Krouk and Smeets proposed in 1997 a digital signature scheme [27] based on error-correcting codes, without involving the decoding step. Subsequently, only the binary version will be considered, i.e. over $\mathbb{F}_2$.

The originally proposed version works as follows: the signer begins by choosing the security parameters n, n', r, k, t_1 and $t_2 \in \mathbb{N}$ according to the desired security level, with $t_1 \leq t_2$, $r < n$ and $n' < n$.

The signer chooses a random parity check matrix $H = [I_r|D] \in \mathcal{M}_{\mathbb{F}_2}^{r\times n}$, as well as a generator matrix $G \in \mathcal{M}_{\mathbb{F}_2}^{k\times n'}$ (G is not the generator matrix corresponding to H).

G is the generator matrix of a (n', k) linear code $\mathcal{U}$. We assume that for any non-zero word $u \in \mathcal{U}$, then with very high probability $t_1 \leq w_h(u) \leq t_2$.

The signer also chooses a random subset $J \subset \{1, \ldots, n\}$, $|J| = n'$, and computes $F = H_J \cdot G^T \in \mathcal{M}_{\mathbb{F}_2}^{r\times k}$, with H_J the matrix formed by the columns of index $j \in J$ of H.

Public key The public key is made from the pair (H, F).
Private key The private key is made from the pair (J, G).
Signature The signer generates the matrix $G^* \in \mathcal{M}_{\mathbb{F}_2}^{k\times n}$ whose columns at positions $j \in J$ are the successive columns of G, and the other columns contain 0. The signature of the message $m \in \mathbb{F}_2^k$ is $(m, m \cdot G^*)$.
Verification Let (m, z) being the received signature, the receiver verifies that $t_1 \leq w_h(z) \leq t_2$ and $F \cdot m^T = H \cdot z^T$.

Our protocol uses a modified version of the KKS scheme. We use the "KKS-3 #2" version proposed in [12]. The security parameters proposed in the paper are as follows: $n = 2000$, $n' = 1000$, $r = 1100$, $k = 160$. We set $t_1 = 470$ and $t_2 = 530$.

The "KKS-3 #2" construction differs from the original version in the following ways:

- The generator matrix $G \in \mathcal{M}_{\mathbb{F}_2}^{k\times n'}$ is given by $G = [I_k|B]$, $B \in \mathcal{M}_{\mathbb{F}_2}^{k\times n'-k}$ random
- The signer generates a secret non-singular matrix $A \in \mathcal{M}_{\mathbb{F}_2}^{k\times k}$, and $F = H_J \cdot (A \cdot G)^T$.

To speed up the computations, we use circulant matrices to generate the matrices $D \in \mathcal{M}_{\mathbb{F}_2}^{r\times n-r}$, $B \in \mathcal{M}_{\mathbb{F}_2}^{k\times n'-k}$, and $A \in \mathcal{M}_{\mathbb{F}_2}^{k\times k}$, as suggested in [12].

4.3 Known vulnerabilities

Limit on the number of signatures The problem with KKS signatures is that with each signature, the signer reveals a piece of the subset J, as shown in [12]. Once the attacker knows all the elements of the subset J, he can find A and G by solving the linear system $F = H_J \cdot (A \cdot G)^T$. To maintain 128-bit security, it is only possible to sign 5 times at most with "KKS-3 #2". After this time, the signer will have to regenerate the private key, as well as F. However, he can keep the same parity-check matrix H.

It is to be feared that an adversary who tries to forge a signature is equipped with a powerful enough quantum computer. He could then use Grover's algorithm [23] to try to find the subset J more quickly.

Grover's algorithm allows a quadratic reduction of the search complexity on a quantum computer. If we consider that 128 bits of security are sufficient against a classical attacker, 160 bits of security are enough against an attacker equipped with a quantum computer [20].

[12] gives us the formula to compute l_γ, the maximum number of signatures before the attacker is able to find the private key with less than 2^{80} computations. So we will recompute the formula for 160-bit security, against an adversary equipped with a quantum computer.

Let's start by computing the λ security parameter:

$$\lambda = \frac{160 - \nu \log_2(n') - \log_2(k)}{n - n'} \tag{7}$$

With $\nu = 3$ (n'^ν is the complexity of solving a system of equations with n' unknowns).

Then we define γ, the smallest positive real such that $h_2(\gamma) = \lambda$, with

$$h_2(x) = -x \log_2(x) - (1 - x) \log_2(1 - x) \tag{8}$$

Finally we compute as in [12] the value

$$l_\gamma = \left\lfloor \frac{\log_2(\frac{\gamma}{1-\gamma}) + \log_2(\frac{n}{n'} - 1)}{\log_2(1 - \frac{t_2}{n'})} \right\rfloor \tag{9}$$

We then obtain $l_\gamma = 5$, meaning the signer has to renew his key at most every 5 signatures to guarantee 160-bit security.

Attack on insecure security settings [36] also proposes an interesting attack on different proposals for KKS. The attack allows to reconstruct the public key without even knowing a single message / signature pair, when the parameters are vulnerable.

The attack can be avoided if the following two conditions are respected:

- n' must be large enough,
- $\frac{r}{n} \gg 2 \cdot \frac{k}{n'}$

5 Shamir's Secret Sharing Scheme (SSSS)

Shamir's Secret Sharing Scheme (SSSS) [39] is a cryptographic primitive allowing to share a secret S between n participants, such that at least k participants must pool their shares to reconstruct the secret S.

To this end, the dealer, i.e. the one who knows the secret S initially, generates a prime number p such that $p > S$. Then the dealer generates a random polynomial P of degree $k - 1$ as

$$P(X) = a_0 X^{k-1} + a_1 X^{k-2} + \cdots + a_{k-2} X + S \tag{10}$$

with $a_0, \ldots, a_{k-2} \in [\![0, p-1]\!]$.

Thus, $P(0) = S$.

The dealer will then distribute the secret shares to the n participants by communicating to them respectively $P(1), P(2), \ldots, P(n)$. k among n participants will then be able to reconstruct the polynomial P and so recover the secret $S = P(0)$ by Lagrangian interpolation [43], by pooling their shares.

Shamir's Secret Sharing has been proven Information Theoretic Secure [15], therefore resistant to quantum computer attacks.

6 Protocol description

Our post-quantum KGC-free certificateless encryption protocol uses QC-MDPC error-correcting codes. It is based on the certificateless encryption protocol proposed by [31]. In our protocol, we assume that each node has a *unique identifier* (id) linked to the network communication mode, therefore impossible to counterfeit and verifiable by other nodes. This identifier can be for example an IP address, an email address or a physical address. To simplify our representation, we will use in this paper incremental numbers for our unique identifiers, starting from 1 for the initial node.

6.1 Choosing the election threshold

As explained in introduction, it is necessary to define in advance the threshold T of votes that will be necessary for a new node to be accepted into the network, depending on the criticality of this network. Depending upon the context, T will denote either a proportion or a value, eg. $T = \frac{1}{3}$ or $T = \frac{1}{3} \cdot n$.

6.2 Choosing the security parameters

On our protocol based on QC-MDPC McEliece, it is necessary to define the security parameters, which will influence the length of the keys, as well as the computation time. These parameters are:

- the block length of the public key p
- the row weight determinant of the parity-check matrix ω'
- the errors count t
- the dimension of the signature generator matrix sig_k
- the length of the signature code sig_n
- the dimension of the signature parity-check matrix sig_r
- the cardinality of the signature subset n'

For the encryption part, we propose the following parameters: $p = 8009$, $\omega' = 100$ and $t = 50$. We will detail in section 8 the security analysis of our protocol based on these parameters.

Note that p must be prime in order to avoid non-prime quasi-cyclicity attacks [21].

For the signature, we keep the parameters suggested in [12]: $sig_k = 160$, $sig_n = 2000$, $sig_r = 1100$ and $sig'_n = 1000$.

6.3 Encryption primitive

Definition 6. *The encryption scheme Π, with message space $\mathbb{F}_2^p$, for security parameters $p, \omega', t \in \mathbb{N}$, node id $i \in \mathbb{N}$ and acceptance vector $s_i \in \mathbb{F}_2^p$ (defined below) consists of the algorithms (KeyGen, Enc, Dec):*

KeyGen($\boldsymbol{n}$, $\boldsymbol{\omega'}$, $\boldsymbol{t}$, $\boldsymbol{i}$, $\boldsymbol{s_i}$) Node i defines the weights $\omega_{h_1} = \left\lfloor \sqrt[3]{\frac{\omega'}{2}} \right\rfloor$, $\omega_{h_2} = \left\lfloor \sqrt[3]{\frac{\omega'}{2}} \right\rfloor$ and $\omega_{h_3} = \left\lfloor \frac{\omega'}{2} \right\rfloor$.

Node i generates the binary circulant matrix $H_{i,1} \in \mathcal{M}_{\mathbb{F}_2}^{p \times p}$ of row weight ω_{h_1} from $h(i)$, with h being a hash function, following an algorithm defined below. It then generates the random secret binary circulant matrices $H_{i,2} \in \mathcal{M}_{\mathbb{F}_2}^{p \times p}$ and $H_{i,3} \in \mathcal{M}_{\mathbb{F}_2}^{p \times p}$, of respective row weights ω_{h_2} and ω_{h_3}, where $H_{i,2}$ must be invertible.

Node i finally generates the acceptance circulant matrix S_i in $\mathcal{M}_{\mathbb{F}_2}^{p \times p}$ from the vector s_i, as defined in section 3.1.

Node i then generates the private key $s_k = H_i$ (parity-check matrix) and the public key $p_k = G_i$ (generator) as follows:

$$H_i = [H_{i,3} | H_{i,2} \cdot H_{i,1} \cdot S_i] \in \mathcal{M}_{\mathbb{F}_2}^{p \times 2p} \tag{11}$$

$$G_i = [I_{\mathcal{M}_{\mathbb{F}_2}^{p \times p}} | (S_i^{-1} \cdot H_{i,1}^{-1} \cdot H_{i,2}^{-1} \cdot H_{i,3})^T] \in \mathcal{M}_{\mathbb{F}_2}^{p \times 2p} \tag{12}$$

with $I_{\mathcal{M}_{\mathbb{F}_2}^{p \times p}}$ the binary identity matrix of size $p \times p$

Enc($\boldsymbol{G_i}$, $\boldsymbol{m}$) Node i generates an error vector $e \in \mathbb{F}_2^p$, such that $w_h(e) = t$. Then generates the ciphertext $c \in \mathbb{F}_2^{2p}$ by

$$c = m \cdot G_i + e \tag{13}$$

Dec($\boldsymbol{H_i}$, $\boldsymbol{c}$) Node i uses a decryption algorithm θ (bitflip, backflip...) to recover the message m from the ciphertext c, using the parity-check matrix H_i.

Proposition 1. *The McEliece cryptosystem Π = (KeyGen, Enc, Dec) as defined above with the specified parameters p, ω' and t achieves post-quantum safety.*

Proposition 1 will be proven in Section 8.

6.4 Network initialization

Our protocol initializes with an initial node, let's call it *node 1* (we will see later why we cannot initialize this identifier to 0, in the case where we use an incremental counter to uniquely identify our nodes).

Definition of public parameters From the parameters p, ω' and t defined above, node 1 starts by generating a random prime number q of $4p$ bits. This is the prime number that will be used to generate the Shamir shares.

Node 1 then defines the weight of the acceptance vector s_1, ω_s, as $\omega_s = \left\lfloor \sqrt{\frac{\omega'}{2}} \right\rfloor$. It also defines the weights ω_{h_1}, ω_{h_2} and ω_{h_3} as in section 6.3.

Finally, node 1 generates the binomial modulo $b_{mod} = \binom{q}{\omega_s}$.

Node 1 publishes the generated parameters q, ω_s, ω_{h_1}, ω_{h_2}, ω_{h_3} and b_{mod}.

Keys generation Node 1 then generates a random invertible binary circulant acceptance matrix S_1 of size $p \times p$ and row weight ω_s (as a reminder, we define the row weight of a circulant matrix as the weight of any of its rows). It is interesting to note that here, node 1 generates its own acceptance matrix, since it is the only node present on the network. The acceptance matrix actually corresponds to the *partial private key* (psk) in a classic certificateless encryption protocol. Node 1 publishes the first row s_1 of the circulant matrix S_1.

Node 1 then generates the matrices $H_{1,2}$ and $H_{1,3}$ of size $p \times p$, with row weights ω_{h_2}, ω_{h_3} respectively. These two matrix $H_{1,2}$ and $H_{1,3}$ are kept secret. $H_{1,2}$ must be invertible.

To generate the matrix $H_{1,1}$ of size $p \times p$ and row weight ω_{h_1}, node 1 proceeds as follows. First, it computes the hash of its identifier $h = hash(id) = hash(1)$. In our experiment, we used the hash function SHA3_512 [6], but other secure hash functions are also suitable. From h, node 1 generates the binary vector $h_{1,1}$ of weight ω_{h_1} and length p. To do this, it keeps the positions of the first ω_{h_1} bits which were set to 1 of h, the other bits of $h_{1,1}$ are set to 0. For example, if $h = 10111101$, $p = 5$ and $\omega_{h_1} = 2$, then $h_{1,1} = \mathbf{1}0\mathbf{1}00$. From the binary vector $h_{1,1}$, node 1 generates the invertible circulant matrix $H_{1,1}$. Obviously, each node in the network will then be able to regenerate the matrix $H_{1,1}$ from the identifier of node 1.

Node 1 then generates the private key H_1 (parity-check matrix) and the public key G_1 (generator) as follows:

$$H_1 = [H_{1,3} | H_{1,2} \cdot H_{1,1} \cdot S_1] \tag{14}$$

The row weight ω of the parity-check matrix H_1 is $\omega = O(\omega')$.

$$G_1 = [I_{\mathcal{M}_{\mathbb{F}_2}^{p \times p}} | (S_1^{-1} \cdot H_{1,1}^{-1} \cdot H_{1,2}^{-1} \cdot H_{1,3})^T] \tag{15}$$

$I_{\mathcal{M}_{\mathbb{F}_2}^{p \times p}}$ is the binary identity matrix of size $p \times p$. Note that since the left part of the generator matrix is the identity matrix, then this scheme does not guarantee the indistinguishability property (IND-CPA) on the text encrypted with the generator matrix. Indeed, the first p bits of the encrypted message contain the original message, with on average $\frac{t}{2}$ errors. Since $\frac{t}{2} \ll p$, then the attacker is able to distinguish two distinct plaintexts from the ciphertexts. It will therefore be necessary to apply another hash function on the plaintext thus transmitted, for cryptographic use.

Node 1 finally publishes its public key G_1, and keeps its private key H_1 secret.

Generation of witness vectors Node 1 generates the witness circulant binary matrix of public key R_1 as follows:

$$R_1 = H_{1,2}^{-1} \cdot H_{1,3} \tag{16}$$

Node 1 publishes the first row r_1 of R_1.

Each node will then be able to verify that the public key G_1 presented by node 1 is authentic, by comparing it with $G_{1,\text{verif}}$:

$$G_{1,\text{verif}} = G_1 = [I_{\mathcal{M}_{\mathbb{F}_2}^{p \times p}} | (S_1^{-1} \cdot H_{1,1}^{-1} \cdot R_1)^T] \tag{17}$$

If $G_{1,\text{verif}} \neq G_1$, then the public key supposedly presented by node 1 is counterfeit.

Signing keys generation Node 1 generates a pair of signing keys KKS, as shown in section 4. We then have $pk_{1,\text{sign}} = (H_{1,\text{sign}}, F_{1,\text{sign}})$ and $sk_{1,\text{sign}} = (J_{1,\text{sign}}, G_{1,\text{sign}}, A_{1,\text{sign}})$.

$F_{1,\text{sign}}$, $G_{1,\text{sign}}$ and $A_{1,\text{sign}}$ are randomly generated circulant matrices, $J_{1,\text{sign}}$ is a random subset of $\{1, \ldots, sign_n\}$. Recall that $H_{1,\text{sign}} = [I_{sign_r} | D_1] \in \mathcal{M}_{\mathbb{F}_2}^{sign_r \times sign_n}$, with $D_1 \in \mathcal{M}_{\mathbb{F}_2}^{sign_r \times sign_n - sign_r}$ circulant. In order to reduce the amount of messages sent, the matrix D_1 is generated from the first $sign_r$ rows and $sign_n - sign_r$ columns of $R1$. $G^*_{1,\text{sign}} \in \mathcal{M}_{\mathbb{F}_2}^{sign_k \times sign_n}$ is generated from the columns of $G_{1,\text{sign}}$, as explained in 4.

Node 1 then publishes the matrix $F_{1,\text{sign}}$, as everyone can reconstruct $H_{1,\text{sign}}$ from R_1.

Group key generation Node 1 finally generates a random group key k. This group key will not be of use to it for the moment, since node 1 is alone in the network.

6.5 Accepting a new node

In a classical certificateless encryption protocol, the acceptance of a new node is decided unilaterally by the Key Generation Center (KGC). Here we describe an alternative based on a distributed approach.

Suppose that a new node, with the identifier x, wishes to join the network, already made up of n nodes. Let us assume that the acceptance threshold of a new node is T votes among the n existing nodes.

New node initialization Node x will start by generating the two secret circulant matrices $H_{x,2}, H_{x,3} \in \mathcal{M}_{\mathbb{F}_2}^{p \times p}$, of row weights ω_{h_2} and ω_{h_3} respectively, as well as the matrix $H_{x,1} \in \mathcal{M}_{\mathbb{F}_2}^{p \times p}$ from its unique hash $h = hash(id) = hash(x)$, as described above. $H_{x,1}$ and $H_{x,2}$ are invertible.

Acceptance signature request The joining node x will then make admission requests until it obtains T acceptances.

When a node i accepts that node x joins the network, it responds with its acceptance signature $sign_i(x) = h_{x,\text{sign}} \cdot G^*_{i,\text{sign}}$, with $h_{x,\text{sign}}$ the hash of id x of length sig_k, and $G^*_{i,\text{sign}} \in \mathcal{M}_{\mathbb{F}_2}^{sig_k \times sig_n}$ the matrix containing the successive columns of $G_{i,\text{sign}}$ in the positions defined by the secret subset J, and 0 in the other columns.

Thus, each node is able to verify the validity of $sign_i(x)$ from the witness vectors r_i and signing public key $pk_{i,\text{sign}}$ previously broadcast by node i. The nodes regenerate the circulant witness matrices R_i from r_i and the parity-check matrix $H_{i,\text{sign}}$ from R_i. Then each node is able to verify that $t_1 \leq w_h(sign_i(x)) \leq t_2$ and $F_{i,\text{sign}} \cdot h^T_{x,\text{sign}} = H_{i,\text{sign}} \cdot sign_i(x)^T$.

Acceptance vector generation Node x will now generate the acceptance vector s_x from the T signatures $sign_i(x)$. To do this, it interprets the received vector $sign_i(x)$ as a binary encoded integer.

Node x determines the polynomial Q such that for every $sign_i(x), Q(i) = sign_i(x)$, by Lagrangian interpolation. It then computes the vector $a = Q(0)$. We then understand why it is impossible to start the unique identifiers at 0, as indicated in section 6.4.

Node x computes $y = a \mod \binom{p}{w_s}$. This is the y^{th} way to choose w_s elements among p. Node x therefore initializes the binary vector s_x by choosing the y^{th} way to set w_s elements among p to 1, the others to 0.

For example, if $p = 5$ and $w_s = 3$, then for:

- $y = 0 : s_x = \mathbf{111}00$
- $y = 1 : s_x = \mathbf{11}0\mathbf{1}0$
- $y = 2 : s_x = \mathbf{11}00\mathbf{1}$
- $y = 3 : s_x = \mathbf{1}0\mathbf{11}0$
- ...

Node x then broadcasts its vector s_x as well as the T signatures $sign_i(x)$. Thus each node is able to verify that the acceptance vector s_x has been generated from T (valid) signatures. Node x then only has to generate an acceptance matrix S_x as a circulant matrix of the vector s_x.

Key generation From the vector s_x, node x generates the acceptance invertible circulant matrix S_x. From the invertible circulant matrices S_x, $H_{x,1}$, $H_{x,2}$ and $H_{x,3}$, node x then generates the generator matrix
$G_x = [I_{\mathcal{M}_{\mathbb{F}_2}^{p \times p}} | ((S_x \cdot H_{x,1} \cdot H_{x,2})^{-1} \cdot H_{x,3})^T]$ (public key) as well as the parity-check matrix $H_x = [H_{x,3} | H_{x,2} \cdot H_{x,1} \cdot S_x]$ (private key).

Node x also generates signing keys $pk_{x,\text{sign}}$, $sk_{x,\text{sign}}$, as in section 6.4.

New group key generation When a new node x joins the network, it is necessary to regenerate a group key, in order to ensure backward secrecy. Our protocol uses a *Key Derivation Function* (KDF) [1] for this purpose. For example, we propose to use PBKDF2_SHA256 [28], but any secure key derivation function would be suitable.

After joining the network of n former nodes and generating its keys, the new node x generates a random *seed*, and sends it to all nodes, encrypted with their respective public keys.

Each former node i then computes the new group key k' by:

$$k' = KDF(k, seed) \tag{18}$$

with k being the old group key.

Each old node i then sends back the new group key k' to the new node x, encrypted with its newly generated public key G_x.

The new node x should then receive the new group key k' encrypted n times. If ever some keys differ, node x chooses the majority value for k'. We could imagine that the new node x alerts that some former nodes are trying to forge the group key, but that is beyond the scope of this paper.

Similarly, when a node leaves the network, it is also necessary to generate a new group key, to ensure forward secrecy. To do this, the node with the lowest identifier generates a random *seed*, which it sends to all other nodes in the network, encrypting it with their respective keys. The new group key k' is then generated in the same way.

6.6 Signing keys renewal

As indicated in section 4.3, nodes must regularly renew their signature keys KKS. It is necessary to renew the subset $J_{i,\text{sign}}$, and the matrices $G_{i,\text{sign}}$, $A_{i,\text{sign}}$ and $F_{i,\text{sign}}$. The parity-check matrix may not necessarily have to be renewed. When all nodes are online, node i can broadcast its new public key $pk'_{i,\text{sign}}$ to all other nodes in the network, with the signature of the new key with the old key $pk_{i,\text{sign}}$.

The problem arises during an asynchronous key renewal, that is to say that node i which renews its public key cannot temporarily communicate with another node j. Node j can wait for node i to come back online to ask it again for its new public key signed with the old one. However, if node j wants to quickly verify a signature presented to it, it can also ask one of the neighbor nodes for the new key signed with the old one.

7 Proof of concept

We provide a proof of concept of our protocol in Rust, available at `https://github.com/thomasarmel/democratic_pq_cle`.Be careful though, our implementation is not constant-time, so it could be vulnerable to side-channel attacks.

8 Security analysis

8.1 Formal verification of the protocol

We wrote a formal verification of our protocol with ProVerif. The verification is available at `https://github.com/thomasarmel/democratic_pq_cle/blob/master/formal_verif/democratic_pq_cle.pv`.

ProVerif [8,9] is a protocol verification software, which takes as input an abstract description of the protocol and primitives. It then translates the protocol into logical constraints, and tries to find a counterexample to demonstrate the existence of an attack. The verification is proven "sound", which means that there can be no attack on the protocol that is not detected by ProVerif.

ProVerif works with primitives modeled as "perfect". For example, the assertions in Figure 1 represent a perfect symmetric encryption primitive: the only way to find any information about the original message is to know the key.

```
type key.
fun senc(bitstring, key): bitstring.
reduc forall m: bitstring, k: key; sdec(senc(m, k), k) = m.
```

Fig. 1. Representation of a perfect symmetric encryption primitive in ProVerif.

Then, the user describes the protocol iteratively for each node. The nodes exchange messages on channels, which the attacker can read and modify. If ProVerif finds an attack on the protocol, then it can describe precisely how the attacker intercepts and modifies the messages to achieve the attack.

ProVerif therefore did not find any possible attack on our protocol.

8.2 Arrival of a malicious node without approval

A malicious node that wants to join the network could be tempted to create fake acceptance signatures from other nodes.

However, as we saw in section 4.3, our signature scheme guarantees 160-bit security, or the equivalent of 80-bit security against an attacker with a quantum computer, when nodes renew their keys at most every 5 signatures.

Furthermore, [27] shows that it is as difficult to forge a malicious signature as it is to decode an arbitrary code, outside the conditions mentioned by [36].

8.3 Backward and forward secrecy

As explained in [31], renewing the group key each time a node enters the network ensures backward secrecy. Indeed, the new node will not be able to decrypt the communications it would have listened to before entering the network. In addition, renewing the group key when a node leaves the network ensures forward secrecy, since it will not be able to decrypt the information exchanged after its departure.

The new group key k' is generated with a key derivation function, from the old group key k and a *seed* that is known only to the members of the network. Thus, a node that has just left the network cannot know the new key k' because it does not know the new *seed*. Moreover, a node that has just joined the network will know the new key k' generated from its own *seed* but will not know the old key k, since it is supposedly very difficult to reverse a key derivation function, i.e. to find $k = KDF^{-1}(k', seed)$.

8.4 Security of our QC-MDPC parameters

Key distinguishing attack Let $\mathcal{C}$ be an (n, p, t)-QC-MDPC code, with ω row weight of the parity-check matrix. An attacker is said to succeed in the key distinguishing attack if he can exhibit a valid codeword of weight ω of $\mathcal{C}^{\perp}$, with $\mathcal{C}^{\perp}$ the dual code of $\mathcal{C}$. Succeeding in this attack is equivalent to distinguishing a public key from a random matrix.

In order to determine the complexity of this type of attack, it is necessary to find the work factor of Information Set Decoding for our parameters n, p, t, denoted $WF_{ISD}(n, p, t)$. This is the cost to find a valid codeword of weight t in our code of length n and dimension p. This cost is equivalent to decoding t errors in our code. Our calculations of $WF_{ISD}(n, p, t)$ assume a random binary code. We will see later that the complexity is lower for binary quasi-cyclic codes.

[5] gives us the complexity of the Information Set Decoding algorithm for a random binary code. To compute WF_{ISD}, it is necessary to compute the parameters $\psi, l, \varepsilon_1, \varepsilon_2 \in \mathbb{N}$ in order to minimize

$$T(\psi, l, \varepsilon_1, \varepsilon_2) \cdot \mathcal{P}(\psi, l)^{-1} = WF_{ISD} \tag{19}$$

with the following constraints:

- $l \in [\![0, \min\{n-p, n-p-t-\psi\}]\!]$
- $\psi \in [\![0, \min\{t, p+l\}]\!]$
- $\varepsilon_1 \in [\![0, \min\{p+l-\psi\}]\!]$
- $\varepsilon_2 \in [\![0, \min\{p+l-\psi_1\}]\!]$
- $0 < R_2(\psi, l, \varepsilon_1, \varepsilon_2) < R_1(\psi, l, \varepsilon_1, \varepsilon_2) < l$, R_1 and R_2 are defined below.

We have

$$\mathcal{P}(\psi, l)^{-1} = \frac{\binom{n}{t}}{\binom{p+l}{\psi} \cdot \binom{n-p-l}{t-\psi}} \tag{20}$$

number of iterations and

$$T(\psi, l, \varepsilon_1, \varepsilon_2) = \max\{T_1, T_2, T_3\} \tag{21}$$

time per iteration.

The time complexity of the three “merge-join” steps is given by

$$T_i(\psi, l, \varepsilon_1, \varepsilon_2) = \max\{C_i, S_i\} \ \forall i \in \{1,2,3\} \tag{22}$$

And the space complexity by

$$S(\psi, l, \varepsilon_1, \varepsilon_2) = \max\{S_1, S_2, S_3\} \tag{23}$$

with

$$S_i(\psi, l, \varepsilon_1, \varepsilon_2) = \frac{\binom{p+l}{\psi_i}}{2^{r_i}} \ \forall i \in \{1,2\} \tag{24}$$

and

$$S_3(\psi, l, \varepsilon_1, \varepsilon_2) = \binom{\frac{p+l}{2}}{\frac{\psi_2}{2}} \tag{25}$$

ψ_1 and ψ_2 are given by

$$\psi_1 = \left\lfloor \frac{\psi}{2} \right\rfloor + \varepsilon_1 \tag{26}$$

and

$$\psi_2 = \left\lfloor \frac{\psi_1}{2} \right\rfloor + \varepsilon_2 \tag{27}$$

And we have $r_0 = l$,

$$R_1 = \binom{\psi}{\frac{\psi}{2}} \cdot \binom{p+l-\psi}{\varepsilon_1} \tag{28}$$

$$r_1 = \lfloor \log_2(r_1) \rfloor \tag{29}$$

$$R_2 = \binom{\psi_1}{\frac{\psi_1}{2}} \cdot \binom{p+l-\psi_1}{\varepsilon_2} \tag{30}$$

$$r_2 = \lfloor \log_2(R_2) \rfloor \tag{31}$$

and $r_3 = 0$.
Finally, the expected value $\mathbb{E}(C_i)$ of the constant factor C_i is given by

$$\mathbb{E}(C_i) = S_i^2 \cdot 2^{r_i - r_{i-1}} \ \forall i \in \{1,2,3\} \tag{32}$$

Using the Mathematica notebook provided by [5], we computed the following values for $p = \lfloor 0.5 \cdot n \rfloor$:

- $l = \lfloor 1.722 \cdot 10^{-2} \cdot n \rfloor$
- $\psi = \lfloor 3.11681 \cdot 10^{-3} \cdot n \rfloor$
- $\varepsilon_1 = \lfloor 2.32741 \cdot 10^{-4} \cdot n \rfloor$

– $\varepsilon_2 = \lfloor 1.3983 \cdot 10^{-6} \cdot n \rfloor$

And so we minimized $WF_{ISD} \approx 2^{218.4}$.

For an $(n,\, p,\, t)$-QC-MDPC code, [34] gives us the complexity WF_{DIST} of the key distinguishing attack:

$$WF_{DIST} = \frac{WF_{ISD}}{n-p} \tag{33}$$

Here $n - p = p$ (recall that $n = 2p$). Finally we have $WF_{DIST} \approx 2^{205.4}$.

Key recovery attack Let $\mathcal{C}$ be an $(n,\, p,\, t)$-QC-MDPC code, with ω row weight of the parity-check matrix. An attacker is said to succeed in the key recovery attack if he can exhibit p valid codewords of weight ω of $\mathcal{C}^{\perp}$, with $\mathcal{C}^{\perp}$ the dual code of $\mathcal{C}$. Succeeding in this attack is equivalent to recompute an equivalent private key from the public key.

For an $(n,\, p,\, t)$-QC-MDPC code, [34] gives us the complexity WF_{RECO} of the key recovery attack:

$$WF_{RECO} = \frac{WF_{ISD}}{n-p} \tag{34}$$

So we have $WF_{RECO} \approx 2^{205.4}$.

Direct decoding attack Let $\mathcal{C}$ be an $(n,\, p,\, t)$-QC-MDPC code, an attacker succeeds in the decoding attack if he succeeds in decoding t errors, that is to say he finds the original message without the private key.

For an $(n,\, p,\, t)$-QC-MDPC code, [34] gives us the complexity WF_{DEC} of the decoding attack:

$$WF_{DEC} = \frac{WF_{ISD}}{\sqrt{p}} \tag{35}$$

So $WF_{DEC} \approx 2^{211.9}$.

Remarks In any case, our protocol provides more than 160 bits of security, which means that an attacker with a quantum computer would have to perform more than 2^{80} operations to succeed in one of these attacks.

In order to reproduce our results, you can recompute the values $\psi, l, \varepsilon_1, \varepsilon_2$ from the Mathematica notebook provided by [5], and you can find the details of the work factors computations at `https://github.com/thomasarmel/democratic_pq_cle/blob/master/security_assessments/workfactors.py`.

8.5 GJS attack

In 2016, Guo, Qian and Johansson proposed an attack [24] based on the observation of the node at the time of decryption ("decryption oracle"). This attack is called the *GJS attack*. Thus, it would be possible to reconstruct the private key by sending thousands of specific messages to the node, and simply observing when it succeeds or fails to decrypt them (as a reminder, decryption is a probabilistic algorithm, as indicated in section 2.1).

In order to avoid this attack, it is necessary that the Decoding Failure Rate (DFR), i.e. the rate of valid messages that are not decrypted, be negligible. As indicated by [4], the lifetime ξ of the key is determined by the DFR:

$$\xi = \mathrm{DFR}^{-1} \tag{36}$$

This means that the keys will need to be renewed every ξ messages. The attacker will therefore only see one decoding error on average per key pair.

[38] gives us the formula to compute the DFR of a key pair, given the parameters n code length, p code dimension, ω row weight of the parity-check matrix and t error count. Using the decoding algorithm specified in their paper, we have:

$$\mathrm{DFR}(t) = \sum_{S} P_S(t) \cdot \mathrm{DFR}(S, t) \tag{37}$$

With $\mathrm{DFR}(S, t)$ the probability of a decoding failure with t errors and a syndrome (defined in Definition 1) of weight S, and $P_S(t)$ the probabilistic distribution of the syndrome weight.

The computation of $\mathrm{DFR}(S, t)$, as proposed in the paper, is implemented at `https://github.com/vvasseur/qcmdpc_markov`.

The distribution $P_S(t)$ is given by the binomial distribution:

$$Pr[S = l] = \binom{p}{l} \cdot (\overline{\rho})^l \cdot (1 - \overline{\rho})^{p-l} \tag{38}$$

With

$$\overline{\rho} = \sum_{l=1,\ l\ odd}^{t} \frac{\binom{\omega}{l} \cdot \binom{n-\omega}{t-l}}{\binom{n}{t}} \tag{39}$$

Finally, given our parameters, we find DFR $= 1.80 \cdot 10^{-41} \approx 2^{-135}$. This means that an attacker who wanted to recover a node's private key from the GJS attack would have to send more than 2^{135} messages to the node, and the node would attempt to decrypt each of the messages it received. The code used for these calculations can be found at `https://github.com/thomasarmel/democratic_pq_cle/blob/master/security_assessments/dfr.py`.

9 Protocol performance

In this part, we will analyze the performance of our protocol. First, we will experiment with the computing power required on each node. Then we will analyze the number of messages sent over the entire course of our protocol.

9.1 Required computing power on nodes

We tested the performance of our proof of concept from section 7, on our machine equipped with a 13th Gen Intel Core i7-13700H. For this we compiled the program `O3` optimization flag, and the specific CPU instructions (`-C target-cpu=native`). Here is the time taken by different steps of the protocol:

Matrices and keys generation	1.49 s
Encryption	99.28 ms
Decryption	312.11 ms
Node acceptance signature	901.62 µs
Node acceptance verification	924.2 µs

While the performance of our protocol is acceptable on a personal computer, it may nevertheless pose a problem on an embedded target.

9.2 Transmitted messages count

In this section, we analyze the number of messages transmitted during each of the key steps of the protocol. For this we will note n the number of nodes present in the network, and T the acceptance threshold as defined in section 6.1.

Acceptance of a new node by the former ones When a new node x wants to enter the network, it asks at least T nodes for their acceptance, which then respond with a signature. The new node x then generates its acceptance vector s_x, which it broadcasts to the entire network, with all the signatures, so that all nodes can verify the validity of the vector s_x. The number of messages sent is then $O(n)$, and total data size is $O(n \cdot T)$.

Generating the public encryption key When a new node x generates its encryption key pair, it must broadcast its witness vector r_x to the entire network. The number of messages sent is then $O(n)$.

Signing key renewal KKS signature keys can only be used a maximum of a small number of times. They must therefore be renewed regularly. To do this, node i that wants to renew its keys signs its new public signature key $pk'_{i,\text{sign}}$ with the old key, and broadcasts the new signed key to the entire network. The number of messages sent is then $O(n)$. This operation is likely to occur very often if new nodes are continuously joining the network.

Regenerating the group key on new node arrival When a new node x joins the network, it is necessary to regenerate the group key to ensure backward secrecy. To do this, the new node generates a seed, broadcasts it to the entire network (encrypted with the public keys of the network nodes), and several nodes send it the new group key, encrypted with the public key of the new node x. The number of messages sent is then $O(n)$.

Regenerating the group key when a node leaves It is also necessary to regenerate the group key when a node leaves the network. To do this, the node with the smallest id regenerates a seed that it sends to the entire network (encrypted with the nodes' public keys). It is possible that some nodes wrongly think they have the smallest id, and therefore try to generate a seed. This case must nevertheless be extremely rare, since nodes regularly broadcast messages to the entire network, so it is very unlikely that a node is unaware of the existence of another. The number of messages sent is therefore $O(n)$.

10 Further improvements

The main problem of our protocol is the large number of messages to send. In particular when renewing KKS keys which happens quite often. Nodes could then generate several signing keys in advance, to limit the number of messages to send, as suggested in [12].

It would also be possible to reduce the number of messages to send when requesting acceptances, when the acceptance threshold T is defined as a fraction of the total number of nodes (e.g. $T = \frac{1}{3}$). For this, the new node x would make the acceptance request only to a predefined set of nodes, for example the nodes with an even id. Node x is accepted only if all nodes in the subset accept its entry into the network. This way, the new node x could not "choose" the nodes that would validate its entry into the network. If we then consider that the number of malicious nodes is bounded by the threshold T as defined in section 6.1, then the maximum probability that a new node is accepted is T^m, with m the size of the chosen subset. For example, if the network contains $n = 200$ nodes, and the proportion of malicious nodes is bounded by $T = \frac{1}{3}$, then if the network asks the new node x to obtain the acceptance signatures of all nodes having an id multiple of 10, i.e. the subset of signing nodes has a size $m \approx 20$, then the maximum probability that the new node x is wrongly accepted is $T^m = (\frac{1}{3})^{20} \approx 2.87 \cdot 10^{-10}$.

It might be interesting to modify our protocol, following [14], in order to propose a constant-time implementation.

We could hierarchize the acceptance power of the different nodes. For example, a new node x could only be accepted if T nodes have authorized its entry into the network, including a super node. For this, hierarchical secret sharing would be used [7,42].

11 Discussion

Our protocol allows to manage a secure network from a web of trust. Indeed, it is no longer necessary to have any trusted authority, since the acceptance of new nodes in the network is managed by the vote of the nodes already present.

This type of protocol can only work in a small network and cannot scale since:

- the nodes must have an informed opinion on the arrival of a new peer
- the number of messages sent at each step increases with the number of nodes already present in the network

However, our protocol can be interesting in the case of small networks, where it is impossible to trust a particular node (for example if the nodes are present in different countries, in a tense geopolitical context). We believe that this protocol could be used to extend Quantum Key Distribution networks [41], which suffer from a maximum geographical distance.

12 Conclusion

In this paper, we proposed a distributed post-quantum certificateless encryption protocol, based on the McEliece cryptosystem. Unlike [31], our network does not impose the election of a trusted Key Generation Center, since it is directly the nodes of the network that vote for the integration of a new node. The encryption is based on quasi-cyclic codes with Moderate Density Parity Check Matrix (QC-MDPC). The acceptance of new nodes uses the signature scheme of Kabatianskii, Krouk and Smeets (KKS), as well as the Shamir's Secret Sharing Scheme (SSSS).

We further provide a proof of concept of our protocol, as well as its formal verification and security analysis. The computing power required for our protocol is acceptable on a personal computer or server, but may be problematic for embedded use.

The main flaw of our protocol, however, remains the large amount of messages sent, due to the lack of a centralized KGC. It would therefore be difficult to scale it to several thousand of nodes.

Data availability

All the data needed to replicate our results is freely available in open source, from the links mentioned in this paper.

Statements and Declarations

No funds, grants, or other support was received for conducting this study. The authors have no competing interests to declare that are relevant to the content of this article. The authors have no financial or proprietary interests in any material discussed in this article.

References

1. Adams, C., Kramer, G., Mister, S., Zuccherato, R.: On the security of key derivation functions. In: International Conference on Information Security. Springer (2004). https://doi.org/10.1007/978-3-540-30144-8_12
2. Al-Riyami, S.S., Paterson, K.G.: Certificateless public key cryptography. In: International conference on the theory and application of cryptology and information security. Springer (2003). https://doi.org/10.1007/978-3-540-40061-5_29
3. Aragon, N., Gaborit, P., Hauteville, A., Ruatta, O., Zémor, G.: Low rank parity check codes: New decoding algorithms and applications to cryptography. IEEE Transactions on Information Theory (2019). https://doi.org/10.1109/TIT.2019.2933535
4. Baldi, M., Barenghi, A., Chiaraluce, F., Pelosi, G., Santini, P.: Ledapkc: Low-density parity-check code-based public-key cryptosystem. Specification revision **1** (2017)
5. Becker, A., Joux, A., May, A., Meurer, A.: Decoding random binary linear codes in 2 n/20: How 1+1=0 improves information set decoding. In: Advances in Cryptology–EUROCRYPT 2012: 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings 31. Springer (2012). https://doi.org/10.1007/978-3-642-29011-4_31
6. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Keccak sponge function family main document. Submission to NIST (Round 2) (2009)
7. Birkhoff, G.D.: General mean value and remainder theorems with applications to mechanical differentiation and quadrature. Transactions of the American Mathematical Society (1906). https://doi.org/10.2307/1986339
8. Blanchet, B.: Automatic verification of security protocols in the symbolic model: The verifier proverif. In: Int. School on Foundations of Security Analysis and Design, pp. 54–87. Springer (2012). https://doi.org/10.1007/978-3-319-10082-1_3
9. Blanchet, B., Smyth, B., Cheval, V., Sylvestre, M.: ProVerif 2.00: automatic cryptographic protocol verifier, user manual and tutorial. Version from pp. 05–16 (2018)
10. Burra, M.S., Maity, S.: A distributed and decentralized certificateless framework for reliable shared data auditing for FOG-CPS networks. IEEE Access (2023). https://doi.org/10.1109/ACCESS.2023.3271605
11. Calderbank, A.R., Shor, P.W.: Good quantum error-correcting codes exist. Physical Review A (1996). https://doi.org/10.1103/PhysRevA.54.1098
12. Cayrel, P.L., Otmani, A., Vergnaud, D.: On Kabatianskii-Krouk-Smeets signatures. In: Arithmetic of Finite Fields: First International Workshop, WAIFI 2007, Madrid, Spain, June 21-22, 2007. Proceedings 1. Springer (2007). https://doi.org/10.1007/978-3-540-73074-3_18
13. Chaw, L.F.: Analysis for McEliece and Niederreiter Encryptions: An Alternative to Public Key Encryption. Ph.D. thesis, UTAR (2018)
14. Chou, T.: QcBits: constant-time small-key code-based cryptography. In: International Conference on Cryptographic Hardware and Embedded Systems. Springer (2016). https://doi.org/10.1007/978-3-662-53140-2_14
15. Corniaux, C.L., Ghodosi, H.: An entropy-based demonstration of the security of Shamir's secret sharing scheme. In: 2014 Int. Conf. on Information Science, Electronics and Electrical Engineering. IEEE (2014). https://doi.org/10.1109/InfoSEEE.2014.6948065

16. Courtois, N.T., Finiasz, M., Sendrier, N.: How to achieve a McEliece-based digital signature scheme. In: Advances in Cryptology—ASIACRYPT 2001: 7th International Conference on the Theory and Application of Cryptology and Information Security Gold Coast, Australia, December 9–13, 2001 Proceedings 7. Springer (2001). https://doi.org/10.1007/3-540-45682-1_10
17. De Condorcet, N., et al.: Essai sur l'application de l'analyse à la probabilité des décisions rendues à la pluralité des voix. Imprimerie royale (1785)
18. Fabšič, T., Grošek, O., Nemoga, K., Zajac, P.: On generating invertible circulant binary matrices with a prescribed number of ones. Cryptography and Communications (2018). https://doi.org/10.1007/s12095-017-0239-4
19. Feng, X., Wang, L., Bai, X., Yang, P.: Distributed identity management mechanism based on improved block-chain certificateless encryption algorithm. Physical Communication (2024). https://doi.org/10.1016/j.phycom.2024.102341
20. Fluhrer, S.: Reassessing Grover's algorithm. Cryptology ePrint Archive (2017)
21. Fouque, P.A., Leurent, G.: Cryptanalysis of a hash function based on quasi-cyclic codes. In: Cryptographers' Track at the RSA Conference. Springer (2008). https://doi.org/10.1007/978-3-540-79263-5_2
22. Gallager, R.: Low-density parity-check codes. IRE Transactions on information theory (1962). https://doi.org/10.1109/TIT.1962.1057683
23. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing (1996). https://doi.org/10.1145/237814.237866
24. Guo, Q., Johansson, T., Stankovski, P.: A key recovery attack on MDPC with CCA security using decoding errors. In: Advances in Cryptology–ASIACRYPT 2016: 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I 22. Springer (2016). https://doi.org/10.1007/978-3-662-53887-6_29
25. Hamming, R.W.: Error detecting and error correcting codes. The Bell system technical journal (1950). https://doi.org/10.1002/j.1538-7305.1950.tb00463.x
26. Hamouid, K., Adi, K.: Efficient certificateless web-of-trust model for public-key authentication in MANET. Computer Communications (2015). https://doi.org/10.1016/j.comcom.2015.02.009
27. Kabatianskii, G., Krouk, E., Smeets, B.: A digital signature scheme based on random error-correcting codes. In: Crytography and Coding: 6th IMA International Conference Cirencester, UK, December 17–19, 1997 Proceedings 6. Springer (1997). https://doi.org/10.1007/BFb0024461
28. Kaliski, B.: RFC2898: PKCS# 5: Password-based cryptography specification version 2.0 (2000). https://doi.org/10.17487/RFC2898
29. Kaushal, P.K., Bagga, A., Sobti, R.: Evolution of bitcoin and security risk in bitcoin wallets. In: 2017 International Conference on Computer, Communications and Electronics (Comptelix). IEEE (2017). https://doi.org/10.1109/COMPTELIX.2017.8003959
30. Li, F., Shirase, M., Takagi, T.: Key management using certificateless public key cryptography in ad hoc networks. In: Network and Parallel Computing: IFIP International Conference, NPC 2008, Shanghai, China, October 18-20, 2008. Proceedings. Springer (2008). https://doi.org/10.1007/978-3-540-88140-7_11
31. Liu, J., Tong, X., Wang, Z., Zhang, M., Ma, J.: A centralized key management scheme based on McEliece PKC for space network. IEEE Access (2020). https://doi.org/10.1109/ACCESS.2020.2976753

32. Mansour, M.M.: A turbo-decoding message-passing algorithm for sparse parity-check matrix codes. IEEE Transactions on Signal Processing (2006). https://doi.org/10.1109/TSP.2006.880240
33. McEliece, R.J.: A public-key cryptosystem based on algebraic coding theory (1978)
34. Misoczki, R., Tillich, J.P., Sendrier, N., Barreto, P.S.: MDPC-McEliece: New McEliece variants from moderate density parity-check codes. In: 2013 IEEE international symposium on information theory. IEEE (2013)
35. Nait-Hamoud, O., Kenaza, T., Challal, Y.: Certificateless public key systems aggregation: An enabling technique for 5G multi-domain security management and delegation. Computer Networks (2021). https://doi.org/10.1016/j.comnet.2021.108443
36. Otmani, A., Tillich, J.P.: An efficient attack on all concrete KKS proposals. In: Post-Quantum Cryptography: 4th International Workshop, PQCrypto 2011, Taipei, Taiwan, November 29–December 2, 2011. Proceedings 4. Springer (2011). https://doi.org/10.1007/978-3-642-25405-5_7
37. Schroeder, B., Pinheiro, E., Weber, W.D.: DRAM errors in the wild: a large-scale field study. ACM SIGMETRICS Performance Evaluation Review (2009). https://doi.org/10.1145/2492101.1555372
38. Sendrier, N., Vasseur, V.: On the decoding failure rate of QC-MDPC bit-flipping decoders. In: Post-Quantum Cryptography: 10th International Conference, PQCrypto 2019, Chongqing, China, May 8–10, 2019 Revised Selected Papers 10. Springer (2019). https://doi.org/10.1007/978-3-030-25510-7_22
39. Shamir, A.: How to share a secret. Communications of the ACM (1979). https://doi.org/10.1145/359168.359176
40. Shamir, A.: Identity-based cryptosystems and signature schemes. In: Advances in Cryptology: Proceedings of CRYPTO 84. Springer (1985). https://doi.org/10.1007/3-540-39568-7_5
41. Singh, H., Gupta, D.L., Singh, A.K.: Quantum key distribution protocols: a review. Journal of Computer Engineering (2014)
42. Tassa, T.: Hierarchical threshold secret sharing. In: Theory of Cryptography Conference. Springer (2004). https://doi.org/10.1007/978-3-540-24638-1_26
43. Waring, E.: Vii. problems concerning interpolations. Philosophical transactions of the royal society of London (1779). https://doi.org/10.1098/rstl.1779.0008