BIG QUAKE BInary Goppa QUAsi–cyclic Key Encapsulation

Magali Bardet, Elise Barelli, Olivier Blazy, Rodolfo Canto Torres, Alain Couvreur, Philippe Gaborit, Ayoub Otmani, Nicolas Sendrier, Jean-Pierre Tillich

BIG QUAKE

BInary **G**oppa **QUA**si–cyclic **K**ey **E**ncapsulation

Magali Bardet, University of Rouen, France
Élise Barelli, Inria & École Polytechnique, France
Olivier Blazy, University of Rouen, France
Rodolfo Canto–Torres, Inria, France
Alain Couvreur, Inria & École Polytechnique, France
Philippe Gaborit, University of Limoges, France
Ayoub Otmani, University of Rouen, France
Nicolas Sendrier, Inria, France
Jean-Pierre Tillich, Inria, France

Principal submitter: Alain Couvreur.

Auxiliary submitters: Listed above.

Inventors/Developers: Same as the submitters. Relevant prior work is credited where appropriate.

Implementation Owners: Submitters.

Email Address (preferred): alain.couvreur@inria.fr

Postal Address and Telephone (if absolutely necessary):
Alain Couvreur, LIX, École Polytechnique 91128 Palaiseau Cédex, +33 1 74 85 42 66 .

Signature: ×. See also printed version of "Statement by Each Submitter".

Abstract

The present proposal is a key encapsulation scheme based on a Niederreiter–like public key encryption scheme using binary quasi–cyclic Goppa codes.

Contents

1 Introduction

1.1 Motivation for this proposal

The original McEliece system [**?**] is the oldest public key cryptosystem which is still resistant to classical and quantum computers. It is based on binary Goppa codes. Up to now, all known attacks on the scheme have at least exponential complexity. A security proof is given in [**?**] which relies on two assumptions (i) the hardness of decoding a generic linear code and (ii) distinguishing a Goppa code from a random linear code. It is well known to provide extremely fast encryption and fast decryption [**?**], but has large public keys, about 200 kilobytes for 128 bits of security and slightly less than one megabyte for 256 bits of security [**?**].

The aim of this proposal is to propose a key-encapsulation scheme based on binary Goppa codes by reducing the key size by a moderate factor ℓ in the range [3..19]. This is obtained by using binary quasi–cyclic Goppa codes of order ℓ instead of plain binary Goppa codes. The rationale behind this is that for the original McEliece cryptosystem key-recovery attacks have a much higher complexity than message recovery attacks. By focusing on quasi–cyclic Goppa codes, there is a security loss with respect to key recovery attacks but this loss is affordable due to the big gap between the complexity of key recovery attacks and message recovery attacks, and because the security loss with respect to message recovery attacks is negligible.

1.2 Quasi–cyclic codes in cryptography

This is not the first time that quasi–cyclic codes have been proposed in this context. The first proposal can be traced back to [**?**] where quasi–cyclic subcodes of BCH codes are suggested. This proposal was broken in [**?**], essentially because the number of possible keys was too low.

A second proposal based on quasi–cyclic alternant codes (a family of codes containing the Goppa code family) was made in [**?**]. Because of a too large order of quasi cyclicity, all the parameters of this proposal have been broken in [**?**]. Another proposal with quasi–dyadic and quasi–p–adic Goppa codes was given in [**?**, **?**]. Some parameters have been broken in [**?**, **?**]. In both cases, the corresponding attacks are based on an algebraic modeling of the key recovery attack and using Groebner basis techniques to solve them. This can be done in this case because the quasi–cyclic/dyadic structure allows to drastically reduce the number of variables in the system when compared to the polynomial system associated to unstructured alternant or Goppa codes.

Later on [**?**] provided a further insight on these algebraic attacks by proving that in the case of quasi–cyclic alternant or Goppa codes of order ℓ it is possible to construct another alternant or Goppa code whose length is divided by ℓ without knowing the secret algebraic structure of the code. This code is called the *folded code* there. There is a strong relation between this code and the *invariant code* considered in [**?**]. This explains why in the case of key-recovery attacks attacking quasi–cyclic alternant or Goppa codes we can reduce the problem to a key recovery of a much smaller code.

This sequence of proposals and subsequent attacks lead to the following observations. For a code based scheme using quasi–cyclic algebraic codes to be secure, the following requirements are fundamental:

1. The family of codes providing the keys should be large enough;

2. The security of the key must be studied in terms of the public key and all the smaller codes deriving from the public key (invariant code, folded code, see §4 for further details).

3. The cryptosystem should be resistant to attacks on the message that is generic decoding algorithms.

1.3 Type of proposal

We propose a public key encryption scheme (PKE) which is converted into a key encapsulation mechanism (KEM) using a generic transformation due to Hohheinz, Hövelmanns and Kiltz [**?**] in order to get an INDCCA2 security. Our public key encryption scheme is a Niederreiter–like scheme. Compared to the original Niederreiter scheme, our proposal avoids the computation of a bijection between words of fixed length and constant weight words. This avoids cumbersome computations involving large integers and provides a light scheme more suitable for embedded system with restricted computing resources. The PKE is proved to be IND–CPA and the generic conversion described in [**?**] leads to an IND–CCA2 KEM.

Acknowledgements

The submitters express their gratitude to Daniel Augot and Julien Lavauzelle for their helpful comments. Submitters are are supported by French ANR grant CBCrypt and by the Commission of the European Communities through the Horizon 2020 program under project number 645622 PQCRYPTO.

2 Goppa codes, QC Goppa codes

2.1 Context

In what follows, any finite field is an extension of the binary field $\mathbb{F}_2$.That is, any field is of the form $\mathbb{F}_{2^m}$ for some positive integer m.

2.2 Vectors, matrices

Vectors and matrices are respectively denoted in bold letters and bold capital letters such as $\boldsymbol{a}$ and $\boldsymbol{A}$. We always denote the entries of a vector $\boldsymbol{u} \in \mathbb{F}_q^n$ by $u_0, \ldots, u_{n-1}$.

2.3 Polynomials

Given a finite field $\mathbb{F}_{2^m}$ for some positive m, the ring of polynomials with coefficients in $\mathbb{F}_q$ is denoted by $\mathbb{F}_q[z]$, while the subspace of $\mathbb{F}_q[z]$ of polynomials of degree strictly less than t is denoted by $\mathbb{F}_q[z]_{<t}$. For every rational fraction $P \in \mathbb{F}_q(z)$, with no poles at the elements $u_0, \ldots, u_{n-1}$, $P(\boldsymbol{u})$ stands for $(P(u_0), \ldots, P(u_{n-1}))$. In particular for a vector $\boldsymbol{y} = (y_0, \ldots, y_{n-1})$ that has only nonzero entries, the vector $\boldsymbol{y}^{-1}$ denotes $(y_0^{-1}, \ldots, y_{n-1}^{-1})$.

2.4 Generalized Reed Solomon codes and alternant codes

Definition 1 (Generalized Reed-Solomon code)**.** Let $q = 2^m$ for some positive integer m and k, n be integers such that $1 \leqslant k < n \leqslant q$. Let $\boldsymbol{x}$ and $\boldsymbol{y}$ be two n-tuples such that the entries of $\boldsymbol{x}$ are pairwise distinct elements of $\mathbb{F}_q$ and those of $\boldsymbol{y}$ are nonzero elements in $\mathbb{F}_q$. The *generalized Reed-Solomon* code (GRS in short) $\mathbf{GRS}_k(\boldsymbol{x}, \boldsymbol{y})$ of dimension k associated to $(\boldsymbol{x}, \boldsymbol{y})$ is defined as

$$\mathbf{GRS}_k(\boldsymbol{x}, \boldsymbol{y}) \stackrel{\text{def}}{=} \Big\{ \big(y_0 P(x_0), \ldots, y_{n-1} P(x_{n-1})\big) \mid P \in \mathbb{F}_q[z]_{<k} \Big\}.$$

Reed-Solomon codes correspond to the case where $\boldsymbol{y} = (1\ 1\ \cdots\ 1)$ and are denoted as $\mathbf{RS}_k(\boldsymbol{x})$. The vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ are called respectively the *support* and the *multiplier* of the code.

A GRS code of dimension k with support $\boldsymbol{x}$ and multiplier $\boldsymbol{y}$ has a generator matrix of the form:

$$\begin{pmatrix} y_0 & \cdots & y_{n-1} \\ x_0 y_0 & \cdots & x_{n-1} y_{n-1} \\ \vdots & & \vdots \\ x_0^{k-1} y_0 & \cdots & x_{n-1}^{k-1} y_{n-1} \end{pmatrix}.$$

This leads to the definition of alternant codes. See for instance [**?**, Chap. 12, §2].

Definition 2 (Binary alternant code)**.** Let $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}_q^n$ be a support and a multiplier as defined in Definition 1. Let r be a positive integer, the binary alternant code $\mathscr{A}_r(\boldsymbol{x}, \boldsymbol{y})$ is defined as

$$\mathscr{A}_r(\boldsymbol{x}, \boldsymbol{y}) \stackrel{\text{def}}{=} \mathbf{GRS}_r(\boldsymbol{x}, \boldsymbol{y})^{\perp} \cap \mathbb{F}_2^n.$$

The integer r is referred to as the *degree* of the alternant code and m as its *extension degree*.

Another definition of alternant code, which will be useful in the proposal is given below.

Proposition 1. *Let $\boldsymbol{x}, \boldsymbol{y}, r$ be as in Definition 2. The binary alternant code $\mathscr{A}_r(\boldsymbol{x}, \boldsymbol{y})$ is the right kernel of the matrix:*

$$\boldsymbol{H} = \begin{pmatrix} y_0 & \cdots & y_{n-1} \\ x_0 y_0 & \cdots & x_{n-1} y_{n-1} \\ \vdots & & \vdots \\ x_0^{r-1} y_0 & \cdots & x_{n-1}^{r-1} y_{n-1} \end{pmatrix}.$$

Proposition 2 ([**?**, Chap. 12, § 2])**.** *Let $\boldsymbol{x}, \boldsymbol{y}, r$ be as in Definition 1.*

1. $\dim_{\mathbb{F}_2} \mathscr{A}_r(\boldsymbol{x}, \boldsymbol{y}) \geqslant n - mr$;
2. $d_{min}(\mathscr{A}_r(\boldsymbol{x}, \boldsymbol{y})) \geqslant r + 1$;

where $d_{min}(\cdot)$ denotes the minimum distance of a code.

The key feature of an alternant code is the following fact (see [**?**, Chap. 12, § 9]):

Fact 1. There exists a polynomial time algorithm decoding all errors of Hamming weight at most $\lfloor \frac{r}{2} \rfloor$ once the vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ are known.

2.5 Binary Goppa codes

Definition 3. Let $\boldsymbol{x} \in \mathbb{F}_q^n$ be a vector with pairwise distinct entries and $\Gamma \in \mathbb{F}_q[z]$ be a polynomial such that $\Gamma(x_i) \neq 0$ for all $i \in \{0, \ldots, n-1\}$. The binary Goppa code $\mathscr{G}(\boldsymbol{x}, \Gamma)$ associated to Γ and supported by $\boldsymbol{x}$ is defined as

$$\mathscr{G}(\boldsymbol{x}, \Gamma) \stackrel{\text{def}}{=} \mathscr{A}_{\deg \Gamma}(\boldsymbol{x}, \Gamma(\boldsymbol{x})^{-1}).$$

We call Γ the *Goppa polynomial* and m the *extension degree* of the Goppa code.

The interesting point about this subfamily of alternant codes is that under some conditions, Goppa codes can correct more errors than a general alternant code.

Theorem 1 ([**?**, Theorem 4])**.** *Let $\Gamma \in \mathbb{F}_q[z]$ be a square-free polynomial. Let $\boldsymbol{x} \in \mathbb{F}_q^n$ be a vector of pairwise distinct entries, then*

$$\mathscr{G}(\boldsymbol{x}, \Gamma) = \mathscr{G}\left(\boldsymbol{x}, \Gamma^2\right).$$

From Fact 1, if viewed as $\mathscr{A}_{2\deg\Gamma}(\boldsymbol{x}, \Gamma(\boldsymbol{x})^{-2})$ the Goppa code corrects up to $r = \deg\Gamma$ errors in polynomial-time instead of only $\lfloor \frac{\deg\Gamma}{2} \rfloor$ if viewed as $\mathscr{A}_{\deg\Gamma}(\boldsymbol{x}, \Gamma^{-1}(\boldsymbol{x})))$. On the other hand, these codes have dimension $\geqslant n - mr$ instead of $\geqslant n - 2mr$.

2.6 Quasi–cyclic codes

In what follows ℓ denotes a positive integer.

2.6.1 Definitions

Definition 4. Let ℓ be a positive integer and $\sigma : \mathbb{F}_2^\ell \to \mathbb{F}_2^\ell$ be the cyclic shift map:

$$\sigma : \left\{ \begin{array}{ccc} \mathbb{F}_2^\ell & \longrightarrow & \mathbb{F}_2^\ell \\ (x_0, x_1, \ldots, x_{\ell-1}) & \longmapsto & (x_{\ell-1}, x_0, x_1, \ldots, x_{\ell-2}) \end{array} \right.$$

Now, let n be an integer divisible by ℓ, we define the ℓ–th quasi–cyclic shift σ_ℓ as the map obtained by applying σ block-wise on blocks of length ℓ:

$$\sigma_\ell : \left\{ \begin{array}{ccc} \mathbb{F}_2^n & \longrightarrow & \mathbb{F}_2^n \\ \left(\boldsymbol{x}_0 \mid \ldots \mid \boldsymbol{x}_{\frac{n}{\ell}-1}\right) & \longmapsto & \left(\sigma(\boldsymbol{x}_0) \mid \ldots \mid \sigma\left(\boldsymbol{x}_{\frac{n}{\ell}-1}\right)\right) \end{array} \right. ,$$

where $\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_{\frac{n}{\ell}-1}$ denote consecutive sub-blocks of ℓ bits.

This notion is illustrated by Figure 1.

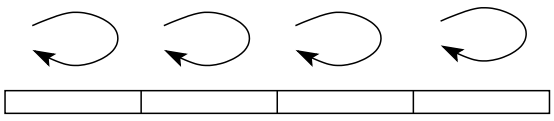

Figure 1: Illustration of the quasi–cyclic shift

Definition 5. A code $\mathscr{C} \subseteq \mathbb{F}_2^n$ is said to be *ℓ–quasi–cyclic* (ℓ–QC) if the code is stable by the quasi–cyclic shift map σ_ℓ. ℓ is also called the *order* of quasi–cyclicity of the code.

Example 1. The following matrix is a generator matrix for a 3–quasi–cyclic code.

$$\begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}.$$

2.6.2 Polynomial representation

Given a positive integer n such that ℓ divides n, then any vector of $\mathbb{F}_2^n$ can be divided into $\frac{n}{\ell}$ blocks of length ℓ. To $(m_0, \ldots, m_{\ell-1})$, one associates naturally the polynomial $m(z) = m_0 + m_1 z + \ldots + m_{\ell-1} z^{\ell-1} \in \mathbb{F}_2[z]/(z^\ell - 1)$. If we let

$$\mathcal{R} \stackrel{\text{def}}{=} \mathbb{F}_2[z]/(z^\ell - 1),$$

then, we get a canonical isomorphism

$$\mathbb{F}_2^n \stackrel{\sim}{\longrightarrow} \mathcal{R}^{\frac{n}{\ell}}$$

and, under this isomorphism, the quasi–cyclic shift corresponds to the scalar multiplication by z. Hence quasi–cyclic codes can be regarded as $\mathcal{R}$–sub–modules of $\mathcal{R}^{\frac{n}{\ell}}$.

Example 2. The code of Example 1 corresponds to the submodule of $\mathcal{R}^{\frac{n}{\ell}}$ spanned by

$$(1 \mid 1 + z).$$

2.6.3 Operations on quasi–cyclic codes

For the analysis of some attacks and hence for the security analysis of our codes, we need to introduce the notions of *invariant* code and *folded* code. These notions will be frequently used in Sections 3 and 4.

Notation 1. We denote by $\mathbf{Punct}_\ell$ the function

$$\mathbf{Punct}_\ell : \begin{cases} \mathbb{F}_2^n & \longrightarrow & \mathbb{F}_2^{\frac{n}{\ell}} \\ (x_0, \ldots, x_{n-1}) & \longmapsto & (x_0, x_\ell, x_{2\ell}, \ldots, x_{n-\ell}) \end{cases}.$$

That is, the map that keeps only the first entry of each block.

Definition 6 (Folding map)**.** Let n be a positive integer such that ℓ divides n. The *folding map* on $\mathbb{F}_2^n$ is the map obtained by summing up the components of each block:

$$\varphi_\ell : \begin{cases} \mathbb{F}_2^n & \longrightarrow & \mathbb{F}_2^{\frac{n}{\ell}} \\ (x_0, \ldots, x_{n-1}) & \longmapsto & \left(\sum_{i=0}^{\ell} x_i, \sum_{i=\ell}^{2\ell-1} x_i, \ldots, \sum_{i=n-\ell}^{n-1} x_i \right) \end{cases}.$$

Equivalently, $\varphi_\ell \stackrel{\text{def}}{=} \mathbf{Punct}_\ell \circ (\mathrm{Id} + \sigma_\ell + \cdots + {\sigma_\ell}^{\ell-1})$.

Definition 7 (Folded code)**.** Given an ℓ–quasi–cyclic code $\mathscr{C} \subseteq \mathbb{F}_2^n$, the *folded code* $\varphi_\ell(\mathscr{C}) \subseteq \mathbb{F}_2^{\frac{n}{\ell}}$ is the image of $\mathscr{C}$ by the folding map.

Definition 8 (Invariant code)**.** Given an ℓ–quasi–cyclic code $\mathscr{C} \subseteq \mathbb{F}_2^n$, the invariant code $\mathscr{C}^{\sigma_\ell} \subseteq \mathbb{F}_2^{\frac{n}{\ell}}$ is the code composed of words fixed by σ_ℓ whose redundant entries have been removed, i.e.

$$\mathscr{C}^{\sigma_\ell} \stackrel{\text{def}}{=} \mathbf{Punct}_\ell\left(\{\boldsymbol{c} \in \mathscr{C} \mid \sigma_\ell(\boldsymbol{c}) = \boldsymbol{c}\}\right).$$

Remark 1. In the previous definitions, the map $\mathbf{Punct}_\ell$ is always applied to *invariant* words, i.e. words such that $\sigma_\ell(\boldsymbol{x}) = \boldsymbol{x}$. Such words are constant on each block. Therefore, the use of $\mathbf{Punct}_\ell$ is only to remove repetitions. Actually one could have replaced $\mathbf{Punct}_\ell$ by any map that keeps one and only one arbitrary entry per block.

We recall a relation between folded and invariant code.

Proposition 3 ([**?**])**.** *If ℓ is odd (which always holds in the present proposal), then*

$$\varphi_\ell(\mathscr{C}) = \mathscr{C}^{\sigma_\ell}.$$

Example 3. If we reconsider the code in Example 1, then, we are in the context of the above proposition, hence the invariant and the folding code are both equal to the code spanned by the vector $\begin{pmatrix}1 & 0\end{pmatrix}$ which corresponds to apply $\mathbf{Punct}_\ell$ on

$$\begin{pmatrix}1 & 1 & 1 & 0 & 0 & 0\end{pmatrix}.$$

2.7 Block–circulant matrices

Definition 9. Let ℓ be a positive integer. Let $\boldsymbol{M}$ be a matrix. The matrix is said to be ℓ–block–circulant if it splits into $\ell \times \ell$ circulant blocks, i.e. blocks of the form

$$\begin{pmatrix} a_0 & a_1 & \cdots & \cdots & a_{\ell-1} \\ a_{\ell-1} & a_0 & \cdots & \cdots & a_{\ell-2} \\ \vdots & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \vdots \\ a_1 & a_2 & \cdots & a_{\ell-1} & a_0 \end{pmatrix}$$

2.8 Quasi–cyclic Goppa codes

There are several manners to construct ℓ–QC Goppa codes. See for instance reference [**?**, **?**].

In this proposal, we will consider ℓ–QC binary Goppa codes for some prime integer ℓ constructed as follows. The exact constraints on ℓ are given in §3.1 and justified in §4.

- Let ℓ be a prime dividing $2^m - 1$. Let ζ_ℓ be a primitive ℓ–th root of unity;
- Let n, t be positive integers divisible by ℓ and set $r \stackrel{\text{def}}{=} \frac{t}{\ell}$;
- The support $\boldsymbol{x} = (x_0, \ldots, x_{n-1})$ is a vector of elements of $\mathbb{F}_{2^m}$ whose entries are pairwise distinct. It splits into n/ℓ blocks of length ℓ of the form $(x_{i\ell}, x_{i\ell+1}, \ldots, x_{(i+1)\ell-1})$ such that for any $j \in \{1, \ldots, \ell-1\}$, $x_{i\ell+j} = \zeta_\ell^j x_{i\ell}$. That is, the support is a disjoint union of orbits under the action of the cyclic group generated by ζ_ℓ. From now on, such blocks are referred to as ζ–**orbits**.
- The Goppa polynomial $\Gamma(z)$ is chosen as $\Gamma(z) = g(z^\ell)$ for some monic polynomial $g \in \mathbb{F}_{2^m}[z]$ of degree $r = t/\ell$ such that $g(z^\ell)$ is irreducible.

Proposition 4. *The Goppa code $\mathscr{G}(\boldsymbol{x}, \Gamma)$ is ℓ–QC.*

2.9 QC–Goppa codes of interest in the present proposal

The ℓ–QC Goppa codes we will consider are those which satisfy the following condition:

Condition 1. The quasi–cyclic code admits a parity–check of the form

$$\boldsymbol{H} = \left(\, I_{n-k} \mid \boldsymbol{M} \, \right)$$

such that $\boldsymbol{M}$ is ℓ–block–circulant.

For a given quasi–cyclic Goppa code it is unclear that such a property is verified. But we observed that, after a possible blockwise permutation of the support (so that the resulting code is still a quasi–cyclic Goppa code of order ℓ) this property is in general satisfied by the codes presented in §2.8. In Appendix A we give an algorithm to find a blockwise permutation providing a quasi–cyclic code satisfying Condition 1. This may happen for instance if no disjoint union of $\frac{n-k}{\ell}$ blocks is an information set. Among the 5000 tests we ran on various parameters, this situation never happened.

Remark 2. Of course, for Condition 1 to be satisfied, the dimension of the code should be a multiple of ℓ. This necessary condition is satisfied if the designed dimension of the Goppa code $n - m \deg g(z^\ell)$ equals the actual dimension (indeed, $\deg(g(z^\ell)) = \ell \cdot \deg(g)$), which almost always holds.

Definition 10. From now on, Goppa codes satisfying Condition 1 are referred to as *Systematic quasi–cyclic Goppa codes*.

The choice of systematic quasi–cyclic codes instead of general quasi–cyclic codes is twofold. First, it makes the security reduction to follow (see §2.10) less technical. Second, such matrices permit to reduce optimally the public key size. Indeed, from such a matrix, $(I_{n-k} \mid \boldsymbol{M})$, it is sufficient to publish only the first row of each circulant block in $\boldsymbol{M}$. Hence, this leads to a public key size $k \times \frac{n-k}{\ell}$. See §5 for further details.

Remark 3. Actually, in our reference implementation, we store the first *column* of each column of blocks.

2.10 Difficult problems from coding theory

Definition 11 ((Search) ℓ-Quasi–Cyclic Syndrome Decoding (ℓ–QCSD) Problem)**.** For positive integers n, t, ℓ, a random parity check matrix $\mathbf{H}$ of a systematic ℓ–quasi–cyclic code $\mathscr{C}$ of dimension k and a uniformly random vector $\mathbf{s} \in \mathbb{F}^{n-k}$, the *Search ℓ-Quasi-Cyclic Syndrome Decoding Problem* ℓ–$\mathsf{QCSD}(n, k, w)$ asks to find $\mathbf{e} = (e_0, \ldots, e_{n-1}) \in \mathbb{F}_2^n$ of Hamming weight t, and $\mathbf{s}^\top = \mathbf{H} \cdot \boldsymbol{e}^\top$.

It would be somewhat more natural to choose the parity-check matrix $\mathbf{H}$ to be made up of independent uniformly random circulant submatrices, rather than with the special form required by Condition 1. We choose this distribution so as to make the security reduction to follow less technical. It is readily seen that, for fixed ℓ, when choosing quasi-cyclic codes with this more general distribution, one obtains with non-negligible probability, a quasi-cyclic code that satisfies Condition 1. Therefore requiring quasi-cyclic codes to be systematic does not hurt the generality of the decoding problem for quasi-cyclic codes.

Assumption 2. Although there is no general complexity result for quasi-cyclic codes, decoding these codes is considered hard. There exist general attacks which use the cyclic structure of the code [**?**] but these attacks have only a very limited impact on the practical complexity of the problem. The conclusion is that in practice, the best attacks are the same as those for non-circulant codes up to a small factor.

Definition 12 (Decisional Indistinguishability of Quasi–Cyclic Goppa Codes from Public Key Sampling (DIQCG problem))**.** Given a random ℓ–quasi–cyclic random code in systematic form and an ℓ–quasi–cyclic Goppa code, distinguish the two types of code.

Assumption 3. For parameters considered for our cryptosystem this problem is considered hard, see Section 4 for further details on the best attacks in this case.

3 Presentation of the scheme

3.1 Notation

In what follows and until the end of the present document, we fix the following notation.

- m denotes a positive integer which refers to the extension degree of a field $\mathbb{F}_{2^m}$. In our reference implementation $m = 12, 14, 16$ or 18.
- ℓ denotes a prime primitive integer which divides $2^m - 1$. By *primitive* we mean that ℓ is prime and 2 generates the cyclic group $\mathbb{Z}/\ell\mathbb{Z}^\times$ of nonzero elements in $\mathbb{Z}/\ell\mathbb{Z}$. The rationale behind this requirement will be explained in § 5.1.
- ζ_ℓ denotes a primitive ℓ–th root of the unity in $\mathbb{F}_{2^m}$.
- n denotes a positive integer $n < 2^m - 1$ which refers to the length of a code. It should be an integer multiple of ℓ.
- $\boldsymbol{x} = (x_0, \ldots, x_{n-1})$ denotes the support of the Goppa code. It has length n and splits into $\frac{n}{\ell}$ blocks of length ℓ such that each block is composed of elements in geometric progression. That is to say:

  $$(x_0, x_1, \ldots, x_{\ell-1}) = (x_0, \zeta_\ell x_0, \zeta_\ell^2 x_0, \ldots, \zeta_\ell^{\ell-1} x_0),$$

  and more generally,

  $$\forall a \in \left\{0, \ldots, \frac{n}{\ell} - 1\right\}, \quad (x_{a\ell}, x_{a\ell+1}, \ldots, x_{(a+1)\ell-1}) = (x_{a\ell}, \zeta_\ell x_{a\ell}, \zeta_\ell^2 x_{a\ell}, \ldots, \zeta_\ell^{\ell-1} x_{a\ell}).$$

- $g(z) \in \mathbb{F}_{2^m}[z]$ denotes a polynomial and the Goppa polynomial of our QC–Goppa codes will be $g(z^\ell)$.
- r denotes the degree of $g(z)$ and $t = r\ell$ denotes the degree of the Goppa polynomial $g(z^\ell)$. Notice that the design minimum distance of this Goppa code is $2t + 1$, therefore, t also denotes the error correcting capacity of the code.
- σ_ℓ denotes the ℓ-th quasi–cyclic shift (see Definition 4).

3.2 Key generation

Consider an ℓ–QC Goppa code of length n and dimension k with ℓ dividing n, k and satisfying Condition 1. Let $\boldsymbol{H}$ be a systematic parity–check matrix for this code:

$$\boldsymbol{H} = (\ I_{n-k} \mid \boldsymbol{M}) \tag{1}$$

where $\boldsymbol{M}$ is an ℓ–blocks–circulant matrix.

Definition 13. Given a matrix $\boldsymbol{H}$ as in (1), we define $\psi(\boldsymbol{H})$ as the matrix obtained from $\boldsymbol{M}$ by extracting only the first row of each block. That is, $\psi(\boldsymbol{H})$ is obtained by stacking rows of $\boldsymbol{M}$ with indexes $0, \ell, 2\ell, \ldots, (n-k)-\ell$.

Note that $\boldsymbol{H}$ is entirely determined by $\psi(\boldsymbol{H})$.

- **Public key** The matrix $\psi(\boldsymbol{H})$.
- **Secret key** The support $\boldsymbol{x}$ and the Goppa polynomial $\Gamma(z) = g(z^\ell)$.

More precisely, Algorithm 1 describes the full key generation algorithm. This algorithm calls Algorithm 3, described in Appendix A. Algorithm 3, performs block–Gaussian elimination on an input matrix $\boldsymbol{H}_0$ and, if succeeds, returns a pair $(\boldsymbol{H}, \tau)$ where $\boldsymbol{H}$ denotes a systematic block–circulant matrix and τ denotes the permutation on blocks applied on $\boldsymbol{H}_0$ in order to get the systematic form $\boldsymbol{H}$.

Algorithm 1: Full key generation algorithm

Input : Positive integers ℓ, $t = \ell r$, n, m such that $\ell | n$
Output: The public and secret key

1 **while** *TRUE* **do**
2 $g(z) \leftarrow$ Random monic polynomial in $\mathbb{F}_{2^m}[z]$ of degree r such that $g(z^\ell)$ is irreducible;
3 $u_0, \ldots, u_{\frac{n}{\ell}-1} \leftarrow$ random elements of $\mathbb{F}_{2^m}$ such that for any pair $i, j \in \{0, \ldots, \frac{n}{\ell} - 1\}$ with $i \neq j$ and any $s \in \{0, \ldots, \ell - 1\}$, we have $u_i \neq \zeta_\ell^s u_j$;
4 $\boldsymbol{x} \leftarrow (u_0, \zeta_\ell u_0, \zeta_\ell^2 u_0, \ldots, \zeta_\ell^{\ell-1} u_0, u_1, \zeta_\ell u_1, \ldots, \zeta_\ell^{\ell-1} u_{\frac{n}{\ell}-1})$;
5 $\boldsymbol{H}_0 \leftarrow$ parity–check matrix of $\mathscr{G}\left(\boldsymbol{x}, g(z^\ell)\right)$;
6 **if** *Algorithm 3 returns FALSE (See Appendix A, page 25)* **then**
7 **Go to line** 2;
8 **end**
9 **else**
10 $\boldsymbol{H}, \tau \leftarrow$ output of Algorithm 3;
11 $\boldsymbol{x} \leftarrow \tau(\boldsymbol{x})$;
12 **break** ;
13 **end**
14 **end**
15 **Public key** $\leftarrow (\psi(\boldsymbol{H}), t)$ (see Definition 13);
16 **Secret key** $\leftarrow (\boldsymbol{x}, g(z^\ell))$.

3.3 Description of the public key encryption scheme

3.3.1 Context

- Bob has published his public key $(\psi(\boldsymbol{H}), t)$. His secret key is denoted as the pair $(\boldsymbol{x}, g(z^{\ell}))$.
- One uses a hash function $\mathcal{H}$. In the reference implementation, we used SHA3;

Suppose Alice wants to send an encrypted message to Bob using Bob's public key. The plaintext is denoted by $\boldsymbol{m} \in \mathbb{F}_2^s$ where s is a parameter of the scheme.

3.3.2 Encryption

(1) e is drawn at random among the set of words of weight t in $\mathbb{F}_2^n$.

(2) Alice sends $\boldsymbol{c} \leftarrow (\boldsymbol{m} \oplus \mathcal{H}(\boldsymbol{e}),\ \boldsymbol{H} \cdot \boldsymbol{e}^{\top})$ to Bob.

3.3.3 Decryption

(1) Bob received $(\boldsymbol{c}_1,\ \boldsymbol{c}_2)$.

(2) Using his secret key, Bob computes $\boldsymbol{e} \in \mathbb{F}_2^n$ as the word of weight $\leqslant t$ such that $\boldsymbol{c}_2 = \boldsymbol{H} \cdot \boldsymbol{e}^{\top}$.

(3) Bob computes $\boldsymbol{m} \leftarrow \boldsymbol{c}_1 \oplus \mathcal{H}(\boldsymbol{e})$.

3.4 Description of the KEM

3.4.1 Context

Alice and Bob want to share a common session secret key K. Moreover,

- Bob publishes his public key $(\psi(\boldsymbol{H}), t)$. His secret key is denoted as the pair $(\boldsymbol{x}, g(z^{\ell}))$.
- One uses a hash function $\mathcal{H}$. In the reference implementation, we used SHA3.
- To perform a KEM, we need to de–randomize the PKE. This requires the use of a function $\mathcal{F} : \{0,1\}^* \to \{\boldsymbol{x} \in \mathbb{F}_2^n \mid w_H(\boldsymbol{x}) = t\}$ taking an arbitrary binary string as input and returning a word of weight t. The construction of this function is detailed further in § 3.4.4.
- We also introduce a security parameter s which will be the number of bits of security. That is, $s = 128$ (resp. 192, resp. 256) for a 128 (resp. 192, resp. 256) bits security proposal, i.e. for NIST security Levels 1, resp. 3, resp. 5.

3.4.2 Key encapsulation mechanism

(1) Alice generates a random $\boldsymbol{m} \in \mathbb{F}_2^s$;

(2) $\boldsymbol{e} \leftarrow \mathcal{F}(\boldsymbol{m})$;

(3) Alice sends $\boldsymbol{c} \leftarrow (\boldsymbol{m} \oplus \mathcal{H}(\boldsymbol{e}), \boldsymbol{H} \cdot \boldsymbol{e}^{T}, \mathcal{H}(m))$ to Bob;

(4) The session key is defined as:

$$K \leftarrow \mathcal{H}(\boldsymbol{m}, \boldsymbol{c}).$$

3.4.3 Decapsulation

(1) Bob received $\boldsymbol{c} = (\boldsymbol{c}_1, \boldsymbol{c}_2, \boldsymbol{c}_3)$;

(2) Using his secret key, Bob can find $\boldsymbol{e}'$ of weight $\leqslant t$ such that $\boldsymbol{c}_2 = \boldsymbol{H} \cdot \boldsymbol{e}'^T$;

(3) Bob computes, $\boldsymbol{m}' \leftarrow \boldsymbol{c}_1 \oplus \mathcal{H}(\boldsymbol{e}')$;

(4) Bob computes $\boldsymbol{e}'' = \mathcal{F}(\boldsymbol{m}')$.

(5) **If** $\boldsymbol{e}'' \neq \boldsymbol{e}'$ or $\mathcal{H}(\boldsymbol{m}') \neq \boldsymbol{c}_3$ then **abort**.

(6) **Else**, Bob computes the session key:

$$K \leftarrow \mathcal{H}(\boldsymbol{m}', \boldsymbol{c}).$$

3.4.4 The function $\mathcal{F}$

The function $\mathcal{F}$ takes an arbitrary binary string as input and returns a word of length n and weight t. The algorithm is rather simple. Here again, the function depends on the choice of a hash function $\mathcal{H}$. In our reference implementation, we chose SHA3.

The construction of the constant weight word, is performed in constant time using an algorithm close to Knuth's algorithm which generates a uniformly random permutation of the set $\{0, \ldots, n-1\}$. Here, the randomness is replaced by calls of the hash function $\mathcal{H}$. The algorithm of evaluation of $\mathcal{F}$ is detailed in Algorithm 2.

Algorithm 2: Function $\mathcal{F}$: construction of a word of weight t

Input : A binary vector $\boldsymbol{m}$, integers n, t
Output: A word of weight t in $\mathbb{F}_2^n$

1 $\boldsymbol{u} \leftarrow (0, 1, 2, \ldots, n-2, n-1)$;
2 $\boldsymbol{b} \leftarrow \boldsymbol{m}$;
3 **for** i **from** 0 **to** $t-1$ **do**
4 $\quad j \leftarrow \mathcal{H}(\boldsymbol{b}) \mod (n-i-1)$;
5 $\quad$ **Swap** entries u_i and u_{i+j} in u;
6 $\quad \boldsymbol{b} \leftarrow \mathcal{H}(\boldsymbol{b})$;
7 **end**
8 $\boldsymbol{e} \leftarrow$ vector with 1's at positions $u_0, \ldots, u_{t-1}$ and 0's elsewhere;
9 **return** $\boldsymbol{e}$

Further details about line 4 Actually the step $j \leftarrow \mathcal{H}(\boldsymbol{b}) \mod (n-i-1)$ should be detailed. If the hash function $\mathcal{H}$ outputs 256 or 512 bit strings, converting this string to a big integer and then reducing modulo $(n-i-1)$ would be inefficient. Hence, the approach consists in

Step 1. truncating $\mathcal{H}(b)$ to a string of s bytes, where s is larger than the byte size of n. In our proposal, $n < 2^{14}$, hence taking $s = 3$ is reasonable and is the choice of our reference implementation.

Step 2. convert this s–bytes string to an integer A:

(a) If $A > 2^{8s} - (2^{8s} \mod n-i-1)$ then go to Step 1 (this should be done to assert a uniformity of the drawn integers in $\{0, \ldots, n-i-2\}$);

(b) else set $j = A \mod (n-i-1)$

Remark 4. Because of Sub-step (2a), we cannot make sure the evaluation of $\mathcal{F}$ is done in constant time, which could represent a weakness (in terms of side channel attacks). To address this issue, first notice that the probability that Sub-step 2a happens is low, and can be reduced significantly by increasing s. Second, one can get almost constant time by finishing the evaluation of $\mathcal{F}$ by performing a small number of fake evaluations of $\mathcal{H}$ to guarantee a constant number of calls of $\mathcal{H}$ with a high probability. This precaution is not implemented in our reference implementation.

3.5 Semantic security

3.5.1 IND-CPA security of the PKE

Theorem 4. *Under the* Decisional indistinguishability of QC Goppa from Public Key Sampling (DIQCG problem)*, and the ℓ–QCSD Problem, the encryption scheme presented above in indistinguishable against Chosen Plaintext Attack in the Random Oracle Model.*

Proof. We are going to proceed in a sequence of games. The simulator first starts from the real scheme. First we replace the public key matrix by a random element, and then we use the ROM to solve the ℓ–QCSD.

We denote the ciphertext of the PKE by $\boldsymbol{c} = (\boldsymbol{c}_1, \boldsymbol{c}_2)$ and recall that $\boldsymbol{c}_1 = \boldsymbol{m} \oplus \mathcal{H}(e)$ and $\boldsymbol{c}_2 = \boldsymbol{H} \cdot \boldsymbol{e}^\top$.

We start from the normal game G_0: We generate the public key $\boldsymbol{H}$ honestly, and $\boldsymbol{e}$ and $\boldsymbol{c}_1$ also.

- In game G_1, we now replace $\boldsymbol{H}$ by a random block–circulant systematic matrix, the rest is identical to the previous game. From an adversary point of view, the only difference is the distribution on $\boldsymbol{H}$, which is either generated at random, or as a quasi–cyclic Goppa parity–check matrix. This is exactly the *DIQCG* problem, hence

$$\mathsf{Adv}_{\mathcal{A}}^{G_0} \leqslant \mathsf{Adv}_{\mathcal{A}}^{G_1} + \mathsf{Adv}_{\mathcal{A}}^{\mathrm{DIQCG}}$$

- In game G_2, we now proceed as earlier except we replace $\mathcal{H}(\boldsymbol{e})$ by random. It can be shown, that by monitoring the call to the ROM, the difference between this game and the previous one can be reduced to the ℓ–QCSD problem, so that:

$$\mathsf{Adv}_{\mathcal{A}}^{G_1} \leqslant 2^{-\lambda} + 1/q_G \cdot \mathsf{Adv}_{\mathcal{A}}^{\ell-\mathrm{QCSD}},$$

  where q_G denotes the number of calls to the random oracle.

- In a final game G_3 we replace $\boldsymbol{c}_1 = \boldsymbol{m} \oplus \mathsf{Rand}$ by just $\boldsymbol{c}_1 = \mathsf{Rand}$, which leads to the conclusion.

□

- $\mathsf{Setup}(1^\lambda)$: as before, except that s will be the length of the symmetric key being exchanged, typically $s = 128, 192$, or 256.
- $\mathsf{KeyGen}(\mathsf{param})$: exactly as before.
- $\mathsf{Encapsulate}(\mathbf{pk})$: generate[a] $\mathbf{m} \xleftarrow{\$} \mathbb{F}^s$ (this will serve as a seed to derive the shared key). Derive the randomness $\theta \leftarrow \mathcal{G}(\mathbf{m})$. Generate the ciphertext $c \leftarrow (\mathbf{u}, \mathbf{v}) = \mathcal{E}.\mathsf{Encrypt}(\mathbf{pk}, \mathbf{m}, \theta)$, and derive the symmetric key $K \leftarrow \mathcal{K}(\mathbf{m}, \mathbf{c})$. Let $\mathbf{d} \leftarrow \mathcal{H}(\mathbf{m})$, and send $(\mathbf{c}, \mathbf{d})$.
- $\mathsf{Decapsulate}(\mathbf{sk}, \mathbf{c}, \mathbf{d})$: Decrypt $\mathbf{m}' \leftarrow \mathcal{E}.\mathsf{Decrypt}(\mathbf{sk}, \mathbf{c})$, compute $\theta' \leftarrow \mathcal{G}(\mathbf{m}')$, and (re-)encrypt $\mathbf{m}'$ to get $\mathbf{c}' \leftarrow \mathcal{E}.\mathsf{Encrypt}(\mathbf{pk}, \mathbf{m}', \theta')$. If $\mathbf{c} \neq \mathbf{c}'$ or $\mathbf{d} \neq \mathcal{H}(\mathbf{m}')$ then abort. Otherwise, derive the shared key $K \leftarrow \mathcal{K}(\mathbf{m}, \mathbf{c})$.

[a] Symbol "$\xleftarrow{\$}$" means "uniformly random element of".

Figure 2: Description of our proposal KEM.

3.5.2 Conversion to an IND-CCA2 KEM/DEM

Let $\mathcal{E}$ be an instance of the public key encryption scheme defined in § 3.3. Let $\mathcal{G}$, $\mathcal{H}$, and $\mathcal{K}$ be hash functions, in our implementation, we chose SHA3. The KEM–DEM version of the system cryptosystem is defined as follows:

According to [**?**], the KEM-DEM version of our PKE is IND–CCA2.

4 Known attacks and counter–measures

We split this section in two parts : key-recovery attacks and message recovery attacks.

4.1 Key recovery attacks

For classical Goppa codes $\mathscr{G}(\boldsymbol{x}, \Gamma)$, a naive brute force key recovery attack consists in enumerating all the possible irreducible polynomials of degree t. In [**?**, **?**] and then in [**?**], it has been proved that the security of quasi–cyclic Goppa codes (and more generally quasi–cyclic *alternant codes*) reduces to that of the *invariant code* (see Definition 8). Moreover, we have the following result.

Theorem 5 ([**?**, **?**]). *Let $\mathscr{G}\left(\boldsymbol{x}, g(z^\ell)\right)$ be a QC–Goppa code. Then,*

$$\mathscr{G}\left(\boldsymbol{x}, g(z^\ell)\right)^{\sigma_\ell} = \mathscr{G}\left(\mathbf{Punct}_\ell(\boldsymbol{x}^\ell), g(z)\right)$$

where

$$\boldsymbol{x}^\ell \stackrel{def}{=} (x_0^\ell, x_1^\ell, \ldots, x_{n-1}^\ell)$$

and the map $\mathbf{Punct}_\ell$ is defined in Notation 1 page 7.

This result is crucial for the security analysis. Indeed, since the public key permits to construct the Goppa code $\mathscr{G}\left(\boldsymbol{x}, g(z^\ell)\right)$, anybody can compute the invariant code, which is a Goppa code too. Moreover, as soon as the structure of the Goppa code is recovered, lifting to the quasi–cyclic Goppa code is possible. See [**?**].

4.1.1 Exhaustive search on Goppa Polynomials and supports

A brute force attack could consist in enumerating all the possible polynomials $g(z^\ell)$ of $\mathscr{G}\left(\boldsymbol{x}, g(z^\ell)\right)$. Then guess the support as a disjoint union of ζ_ℓ–orbits, i.e. a disjoint union of ordered sets of the form $(a, \zeta_\ell a, \zeta_\ell^2 a, \ldots, \zeta_\ell^{\ell-1} a)$ (see § 2.8). If we guessed the good support as a non-ordered set, it is possible to get the good permutation using Sendrier's Support Splitting Algorithm (SSA in short, [**?**]). If it fails, then try with another support defined as a union of ζ–orbits until the good ordering of the support is obtained thanks to SSA.

Actually, this brute force approach can be done on the invariant code and then a lift operation permits to recover the public code. Hence we can proceed as follows.

- Perform brute force search among monic irreducible polynomials $g(z)$ of degree r;
- Guess the support $\mathbf{Punct}_\ell(\boldsymbol{x}^\ell) = \mathbf{Punct}_\ell(x_0^\ell, x_1^\ell, \ldots, x_{n-1}^\ell)$. Note that the elements of the support set are ℓ–th power. Hence there exists only $\frac{2^m-1}{\ell}$ such powers and we need to guess a good subset of length $\frac{n}{\ell}$ among them.
- Perform SSA to check whether the support set is the good one and, if it is, get the permutation and hence the ordered support;
- Deduce from this data the actual Goppa polynomial $g(z^\ell)$ and the good support by extracting ℓ–th roots, here again, the way to find the blockwise good ordering of the elements of the support can be done using either SSA or by solving a linear system.

Thus, let us estimate the maximum number of guesses we need to perform. We need to count the number of monic polynomials $g(z) \in \mathbb{F}_{2^m}[z]$ of degree r such that $g(z^\ell)$ is irreducible.

Set

$$s_r(2^m) \stackrel{\text{def}}{=} \#\{g(z) \in \mathbb{F}_{2^m}[z] \mid \deg(g) = r \text{ and } g(z^\ell) \text{ is irreducible}\}.$$

Remind that the number $m_r(2^m)$ of possible g's, i.e. of monic irreducible polynomials of degree r in $\mathbb{F}_{2^m}[z]$ is given by the well–known formula:

$$m_r(2^m) = \frac{1}{r} \sum_{d|r} \mu(d) 2^{\frac{mr}{d}}, \tag{2}$$

where $\mu(\cdot)$ denotes the Möbius function defined as

$$\mu(r) \stackrel{\text{def}}{=} \sum_{\substack{1 \leqslant k \leqslant r \\ \gcd(k,r)}} e^{2i\pi \frac{k}{r}}. \tag{3}$$

Remark 5. Asymptotically $m_r(2^m) \sim \frac{2^{mr}}{r}$.

Lemma 6.

$$s_r(2^m) \geqslant \left(1 - \frac{1}{\ell}\right) m_r(2^m).$$

The proof of Lemma 6 is given in Appendix C, where a more precise formula is given for $s_r(2^m)$. Consequently, the number of public keys is bounded below by the quantity

$$\#\text{public keys} \geqslant \left(1 - \frac{1}{\ell}\right) m_r(2^m). \tag{4}$$

Remark 6. Actually the number of public keys is much larger since we did not consider the fact that the support of the code need not be the whole $\mathbb{F}_{2^m} \setminus \{0\}$ and hence to estimate the actual number of keys, we need to estimate the number of all the pairs $(\boldsymbol{x}, g(z^\ell))$ where $\boldsymbol{x}$ denotes the support. Then, study the action of the affine group on this set and consider a system of representatives. The point is that for a full support (i.e. the set of elements of the support is $\mathbb{F}_{2^m} \setminus \{0\}$) two distinct Goppa polynomials give two distinct codes and hence there are at least as many keys as $s_r(2^m)$.

4.1.2 Distinguisher on the invariant code

In [**?**], it is proved that high rate Goppa codes are distinguishable from random ones in polynomial time. To assert the security of the system, the public Goppa code **and** the invariant code should be indistinguishable from random ones. Hence, the parameters of the code should be chosen in order to be out of the reach of this distinguisher.

The following statement rephrases the results of [**?**] in a simpler manner.

Proposition 7. *Consider an irreducible binary Goppa code of length n, extension degree m, associated to an irreducible polynomial of degree r. Then the Goppa code is distinguishable in polynomial time from a random code of the same length and dimension as soon as*

$$n > \max_{2 \leqslant s \leqslant r} \frac{ms}{2} \left(s(m - 2e - 1) + 2^e + 2 \right),$$

where $e = \lceil \log_2 s \rceil + 1$.

4.1.3 Algebraic cryptanalysis

The point of such an attack is to recover the structure of the Goppa code. Namely, the support $\boldsymbol{x}$ and the Goppa polynomial $g(z^\ell)$.

The algebraic modeling $A_{X,Y'}$ proposed in [**?**] consists in $\frac{k}{\ell}(t-1)$ equations in $\frac{n}{\ell}-2$ variables X and $\frac{n}{\ell} - \frac{k}{\ell}$ variables Y, that are bi-homogeneous in the X's and Y's variables (a polynomial f is bi-homogeneous of bi-degree (d_1, d_2) if $f(\alpha X, \beta Y) = \alpha^{d_1}\beta^{d_2} f(X, Y)\ \forall (\alpha, \beta) \in \mathbb{F}^2$). For each $1 \leqslant u \leqslant t-1$, there are $\frac{k}{\ell}$ equations of bi-degree $(u, 1)$ in the modeling. And as the Goppa codes considered are binary Goppa codes, even more equations may be added. The system $\mathtt{McE}_{X,Y'}$ in [**?**] contains $\frac{k}{\ell}$ equations of bi-degree $(u, 1)$ for $1 \leqslant u \leqslant t$ and $\frac{k}{\ell}$ equations of bi-degree $(u, 2)$ for $1 \leqslant u \leqslant 2t-1$.

Exhaustive search on the X_i's or Y_i's From the algebraic system we can extract a bilinear system in the X_i's and the Y_j's. A possible attack is to perform an exhaustive search for one of the X_i's or Y_j's, and solve a linear system for the others. The number of unknowns is at least $\frac{n}{\ell} - \frac{k}{\ell} - 2 = \frac{mt}{\ell} - 2$ (after specialization of 1 or 2 values for X and Y) and the cost of the search is asymptotically $2^{m(\frac{mt}{\ell}-2)}$. The bit complexity is $m(\frac{mt}{\ell} - 2)$ which is large enough.

Solving by Groebner basis algorithms A good indicator for the complexity of Groebner basis algorithms is the index of regularity of the ideal (denoted by d_{reg}), since in the homogeneous case it is a bound on the degree of the polynomials in the minimal Groebner basis of the system. For zero-dimensional ideal, the Hilbert series of the ideal is a polynomial,

and the index of regularity is the degree of this polynomial plus 1. This means that during the computation of a Groebner basis, we will have to compute polynomials that may possibly have as many monomials as the number of monomials of degree d_{reg}, which is $\binom{n+d_{reg}}{d_{reg}}$ for polynomials in n variables.

It has been shown in [**?**, **?**] that, if the system of generators of the ideal form a semi–regular sequence of s polynomials of degree $d_1, \ldots, d_s$ in n variables, then the Hilbert series is given by $\left[\frac{\prod_{i=1}^{s}(1-z^{d_i})}{(1-z)^n}\right]^+$ where $\left[\sum_{i\geqslant 0} a_i z^i\right]^+$ is the series $\sum_{i\geqslant 0} a_i z^i$ truncated at the least index i such that $a_i \leqslant 0$.

We show that for the parameters we propose, if the algebraic system from [**?**] where semi–regular (we know it is not), the index of regularity would be large, and that even if the index of regularity of the algebraic system would be small, the size of the polynomials in this degree is beyond the security level.

4.1.4 Algebraic attacks on the invariant code

The cost of such attacks is the most difficult to estimate for many reasons:

- The choice of the algebraic modeling, i.e. the polynomial system we have to solve by Groebner bases methods is not unique. We will suggest here some modeling which have been proposed in the literature but cannot assert that they are the only possible model lings;
- The choice of the monomial ordering has no influence on the theoretical complexity in the worst case, but may have a significant influence on practical complexities.
- Theoretical results on the complexity of Groebner bases suppose the polynomial system to be semi–regular which is not true for the algebraic systems to follow.
- Hence, we provide an analysis of the possible work factor but this approach requires a more thorough study.

4.2 Message recovery attacks

4.2.1 Generic decoding algorithms

Resistance to ISD and their variants. See Christiane Peters' software [**?**]. We provide here an improve version of her software called **CaWoF** (for **Ca**lculate **Wo**rk **Fa**ctor) [**?**], which tests all the most efficient generic decoding algorithms. Namely:

- Prange [**?**];
- Stern [**?**];
- Dumer [**?**];
- May, Meurer, Thomae [**?**];
- Becker, Joux, May, Meurer [**?**].

4.2.2 About the influence of quasi–cyclicity

Decoding one out of many The ℓ–quasi–cyclicity of the code may be used to improve the efficiency of the decoding using Sendrier's Decoding One out Of Many (DOOM, [**?**]). Such approach permits to improve the efficiency by a factor $\sqrt{\ell}$. Since in our proposal the largest proposed ℓ is 19, the use of DOOM may in the best case provide a less than 5 bits reduction of the work factor. Note that it is possible that this gain will be undermined by the practical complexity of a DOOM implementation. In § 5.3, we choose parameters so that the work factors provided by **CaWoF** (which does not take DOOM into account) are at least 1 bit above the limit for $\ell = 3$, 2 bits above the limit for $\ell = 5$ and 13, and 3 bits above the limit for $\ell = 19$. For any of our , we looked 3 bits security for 3–QC codes, 4 for 5, 11, 13–QC codes and 5 for 17–QC codes

An attack based on folding There is also another way to use the quasi-cyclicity for performing message recovery attacks. It consists in using the folding. Let φ_ℓ be the folding operation (see Definition 6). Assume that we want to decode $\boldsymbol{y} = \boldsymbol{c} + \boldsymbol{e}$ where $\boldsymbol{c}$ belongs to the quasi-cyclic Goppa code $\mathscr{C} \stackrel{\text{def}}{=} \mathscr{G}\left(\boldsymbol{x}, g(z^\ell)\right)$ of a certain length n and $\boldsymbol{e}$ is an error of weight t. We clearly have

$$\varphi_\ell(\boldsymbol{y}) = \varphi_\ell(\boldsymbol{c}) + \varphi_\ell(\boldsymbol{e})$$

where $\varphi_\ell(\boldsymbol{c})$ belongs to $\varphi_\ell(\mathscr{C})$ which is the Goppa code $\mathscr{G}\left(\mathbf{Punct}_\ell(\boldsymbol{x}^\ell), g(z)\right)$ by Theorem 5. Each coordinate of $\varphi_\ell(\boldsymbol{e})$ is a sum of ℓ coordinates of $\boldsymbol{e}$. By the piling up lemma (see for instance [**?**])

$$\mathbb{E}\left\{|\varphi_\ell(\boldsymbol{e})|\right\} = \frac{n(1-(1-2p)^\ell)}{2\ell}, \tag{5}$$

where $p \stackrel{\text{def}}{=} \frac{t}{n}$. We have

$$\begin{aligned}\frac{n(1-(1-2p)^\ell)}{2\ell} &\approx \frac{n(2\ell p - 2\ell(\ell-1)p^2}{2\ell} \\ &\approx t - (\ell-1)\frac{t^2}{n}.\end{aligned}$$

Let

$$t' \stackrel{\text{def}}{=} \lfloor t - (\ell-1)\frac{t^2}{n} \rfloor.$$

In other words, our task is to decode $\varphi_\ell(\boldsymbol{y})$ for about t' errors in a Goppa code of length n/ℓ and degree $r = \deg g$. If t' happens to be below the Gilbert-Varshamov distance $d_{\text{GV}}\left(\frac{n}{\ell}, \frac{n}{\ell} - rm\right)$ corresponding to the length $\frac{n}{\ell}$ and dimension $\frac{n}{\ell} - rm$, then we expect that there is typically a single solution to the decoding problem and that it corresponds to the folding of $\boldsymbol{e}$. Recall that this distance is defined by:

Definition 14 (Gilbert-Varshamov distance)**.** Let $h(x) \stackrel{\text{def}}{=} -x\log_2(x) - (1-x)\log_2(1-x)$ be the binary entropy function and h^{-1} be its inverse ranging over $[0, \frac{1}{2}]$. The Gilbert Varshamov distance $d_{\text{GV}}(n,k)$ of a code of length n and dimension k is defined by

$$d_{\text{GV}}(n,k) \stackrel{\text{def}}{=} n \cdot h^{-1}\left(1 - \frac{k}{n}\right).$$

We can hope to find $\varphi_\ell(\boldsymbol{e})$ by decoding $\varphi_\ell(\mathscr{C})$ with generic decoding techniques. It turns out that we gain in the complexity of decoding when we have to decode the folded code instead of decoding the original code with generic decoding techniques. Once we have the folding $\varphi_\ell(\boldsymbol{e})$ of the error we can use this information to perform decoding of the original code $\mathscr{C}$ by puncturing all the positions in a block which corresponds to a position in the support of $\varphi_\ell(\boldsymbol{e})$. We erase at least t' errors belonging to the support of $\boldsymbol{e}$ in this way. There remains about $t - t' \approx (\ell - 1)\frac{t^2}{n}$ errors which can be recovered by generic decoding techniques. We will chose our parameters in order to avoid this case. We namely choose our parameters so that

$$d_{\mathrm{GV}}(n', k') < t'.$$

The best strategy for an attack in the latter case seems to be

1. hope that the folded error has a certain prescribed weight s;
2. compute all possible errors $\boldsymbol{e}'$ in $\mathbb{F}_2^{n'}$ of weight s that have the same syndrome as $\varphi_\ell(\boldsymbol{y})$;
3. Puncture for each such error the s blocks of $\mathscr{C}$ that belong to the support of $\boldsymbol{e}'$. Decode the punctured code for at most $t - s$ errors.

The attack is then optimized over the choices of s.

4.3 Exploiting Quantum Computations.

Recall first that the NIST proposes to evaluate the quantum security as follows:

1. A quantum computer can only perform quantum computations of limited depth. They introduce a parameter, MAXDEPTH, which can range from 2^{40} to 2^{96}. This accounts for the practical difficulty of building a full quantum computer.
2. The amount (or bits) of security is not measured in terms of absolute time but in the time required to perform a specific task.

Regarding the second point, the NIST presents 6 security categories which correspond to performing a specific task. For example Task 1, related to Category 1, consists of finding the 128 bit key of a block cipher that uses AES-128. The security is then (informally) defined as follows:

Definition 15. A cryptographic scheme is secure with respect to Category k iff any attack on the scheme requires computational resources comparable to or greater than those needed to solve Task k.

In the sequel we will estimate that our scheme reaches a certain security level according to the NIST metric and show that the attack takes more quantum resources than a quantum attack on AES. We will use for this the following proposition.

Proposition 8. *Let f be a Boolean function which is equal to 1 on a fraction α of inputs which can be implemented by a quantum circuit of depth D_f and whose gate complexity is C_f. Using Grover's algorithm for finding an input x of f for which $f(x) = 1$ can not take less quantum resources than a Grover's attack on AES-N as soon as*

$$\frac{D_f \cdot C_f}{\alpha} \geqslant 2^N D_{AES-N} \cdot C_{AES-N}$$

where D_{AES-N} and C_{AES-N} are respectively the depth and the complexity of the quantum circuit implementing AES-N.

This proposition is proved in Appendix B. The point is that (essentially) the best quantum attack on our scheme consists in using Grover's search on either the message attacks or the key recovery attacks where Grover's search can be exploited. The message attacks consist essentially in applying Grover's algorithm on the information sets computed in Prange's algorithm (this is Bernstein's algorithm [**?**]). Theoretically there is a slightly better algorithm consisting in quantizing more sophisticated ISD algorithms [**?**], however the improvement is tiny and the overhead in terms of circuit complexity make Grover's algorithm used on top of the Prange algorithm preferable in our case.

5 Parameters

In this section we propose some parameters for various security levels. We start with informal discussions which explain in which range we choose our parameters.

5.1 Choice of the quasi–cyclicity order ℓ

The quasi–cyclicity order guarantees the reduction of the key size compared to non quasi–cyclic Goppa codes. The larger the ℓ the smaller the public key.

On the other hand, too large ℓ's may lead to algebraic attacks such as [**?**, **?**, **?**]. In addition we suggested in § 3, that ℓ should be prime and *primitive*, which means that 2 generates $(\mathbb{Z}/\ell\mathbb{Z})^\times$, or equivalently that the polynomial $1 + z + \cdots + z^{\ell-1}$ is irreducible in $\mathbb{F}_2[z]$. The motivation for this property is to limit the possibilities for the attacker to construct intermediary codes which could help to build an attack. Therefore

- ℓ **should be prime** since if not, for any $e|\ell$, then the code $\mathscr{G}\left(\boldsymbol{x}, g(z^\ell)\right)$ is also e–quasi–cyclic and one can construct an intermediary invariant code $\mathscr{G}\left(\boldsymbol{x}, g(z^\ell)\right)^{\sigma_\ell{}^e}$ which is nothing but $\mathscr{G}\left(\boldsymbol{x}^e, g(z^{\frac{\ell}{e}})\right)$. Thus, it is possible to construct an intermediary code from the single knowledge of a generator matrix of the public code and this intermediary code is a smaller Goppa code with a Goppa polynomial and support strongly related to the public code.

  Of course, we cannot avoid that an attacker can compute the invariant code, but we guess that having the possibility to build intermediary Goppa codes would be a help for the attacker, hence we reject this possibility by requiring ℓ to be prime.

- ℓ **should be primitive** Indeed, in [**?**], from a public Goppa code, the authors consider the *folded* code (see Definition 7), This folding is nothing but the image of the code by the map $\mathrm{id}+\sigma_\ell+\sigma_\ell{}^2+\cdots+\sigma_\ell{}^{\ell-1}$. In the same manner, if the polynomial $1+z+\cdots+z^{\ell-1}$ is reducible over $\mathbb{F}_2$, then, for any divisor $P(z)$ of this polynomial, one can construct an intermediary quasi–cyclic subcode of the public code, which is the image of $\mathscr{G}\left(\boldsymbol{x}, g(z^\ell)\right)$ by the map $P(\sigma_\ell)$. This code is not a Goppa code in general but we guess that its structure could be helpful for an attacker. Therefore, we exclude this possibility by requiring ℓ to be primitive. Among the odd prime numbers below 20, the primitive ones are

$$\ell \in \{3, 5, 11, 13, 19\}$$

In particular we exclude 7 and 17.

Remark 7. At several places in the discussion above we suggest that having some data could "help an attacker". We emphasize that these arguments are only precautions, we actually do **not** know how to use such data for cryptanalysis. In particular, the choice of ℓ to be primitive is more a precaution than a necessary condition for the security.

5.2 Choice of the field extension m

To provide a binary Goppa code, we first need to choose a finite extension $\mathbb{F}_{2^m}$ of $\mathbb{F}_2$. Let us first discuss the choice of m.

Informal discussion on m

By *informal*, we mean that, for the moment, we do not clarify what we mean by *large* or *small*.

(i) A large m provides codes which are "far from" generalized Reed–Solomon codes. Hence, when m is large Goppa codes have less structure. Note that q–ary Goppa codes with $m = 2$ have been broken by a polynomial-time distinguishing and filtration attack in [**?**] and that rather efficient algebraic attacks for small m ($m = 2$ or 3) over non prime q–ary fields exist [**?**]. This encourages to avoid too low values of m. In addition, m should be large enough to have a large enough code length.

(ii) On the other hand m should not be too large since it has a negative influence on the rate of the code. That is to say, for a fixed error correcting capacity t an a fixed code length n, the dimension is $n - mt$, hence the rate is $1 - m\frac{t}{n}$.

(iii) Finally, to get ℓ–quasi–cyclic codes, ℓ should divide $2^m - 1$ (see § 2.8) and ℓ should not be too large to prevent algebraic attacks as [**?**, **?**, **?**]. Thus, $2^m - 1$ should have small factors.

In this proposal we suggest that a good tradeoff between (i) and (ii) would be $m \in \{12, \ldots, 18\}$ To seek for ℓ's, let us factorize the corresponding $2^m - 1$'s.

- $\mathbb{F}_{2^{12}}$: $2^{12} - 1 = 3^2 \cdot 5 \cdot 7 \cdot 13$.
- $\mathbb{F}_{2^{13}}$: $2^{13} - 1$ is prime.
- $\mathbb{F}_{2^{14}}$: $2^{14} - 1 = 3 \cdot 43 \cdot 127$.
- $\mathbb{F}_{2^{15}}$: $2^{15} - 1 = 7 \cdot 31 \cdot 151$.
- $\mathbb{F}_{2^{16}}$: $2^{16} - 1 = 3 \cdot 5 \cdot 17 \cdot 257$.
- $\mathbb{F}_{2^{17}}$: $2^{17} - 1$ is prime.
- $\mathbb{F}_{2^{18}}$: $2^{18} - 1 = 3^3 \cdot 7 \cdot 19 \cdot 73$.

This immediately excludes $m = 13$ and 17. To prevent algebraic attacks, we prefer avoiding ℓ's larger than 20 and, as explained above and since we look only for primitive ℓ's our proposal will focus on

- 3, 5 and 13–quasi–cyclic Goppa codes with $m = 12$ (only for security Level 1, i.e. AES128)
- 3–quasi–cyclic Goppa codes with $m = 14$ (for Levels 2 and 3, i.e. respectively AES192 and AES256)
- 5–quasi–cyclic Goppa codes with $m = 16$ (for Levels 2 and 3)
- 19–quasi–cyclic Goppa codes with $m = 18$. (for Levels 2 and 3)

5.3 Proposition of parameters

In the following tables we use notation

- m : extension degree of the field of definition of the support and Goppa polynomial over $\mathbb{F}_2$;
- n length of the quasi–cyclic code;
- k dimension of the quasi–cyclic code;
- ℓ denotes the order of quasi–cyclicity of the code;
- r denotes the degree of $g(z)$;
- t denotes error–correcting capacity, which is nothing but the degree of $g(z^\ell)$;
- w_{msg} work factor for message recovery errors. It is computed using **CaWoF** library;
- Keys is a lower bound for the number of possible Goppa polynomials (see (4));
- Max Dreg denotes the maximal degree of regularity that such a system could have in order that the size of the Macaulay matrix does not exceed 2^{128} bits under the assumption that Gaussian elimination's cost on $n \times n$ matrices is $\Omega(n^2)$.

5.3.1 Parameters for reaching NIST security level 1 (AES128)

m	n	k	ℓ	Size (bytes)	r	$t = r\ell$ ($\deg g(z^\ell)$)	w_{msg}	Keys	Max Dreg
12	3600	2664	3	103896	26	78	129	1027	8
12	3500	2480	5	63240	17	85	130	684	9
12	3510	2418	13	25389	7	91	132	263	11

5.3.2 Parameters for reaching NIST security level 3 (AES192)

m	n	k	ℓ	Size (bytes)	r	$t = r\ell$ ($\deg g(z^\ell)$)	w_{msg}	Keys	Max Dreg
14	6000	4236	3	311346	42	126	193	5751	11
16	7000	5080	5	243840	24	120	195	6798	12
18	7410	4674	19	84132	8	152	195	2696	16

5.3.3 Parameters for reaching NIST security level 5 (AES256)

m	n	k	ℓ	Size bytes	r	$t = r\ell$ $(\deg g(z^\ell))$	w_{msg}	Keys	Max Dreg
14	9000	7110	3	559913	45	135	257	6039	14
16	9000	6120	5	440640	36	180	260	8129	15
18	10070	6650	19	149625	10	190	263	3412	20

6 Implementation

6.1 Reference implementation

We provide a reference implementation of the public key encryption scheme converted into a key encapsulation mechanism. That is to say, our implementation performs the encapsulation and decapsulation mechanism as described in § 3.4.2 and 3.4.3.

We remind that the hash function used in the reference implementation is SHA3.

6.2 Optimized implementation

Is the same as the reference implementation.

7 Performance Analysis

The platform used in the experiments was equipped with an Intel® Xeon™ E3-1240 v5 clocked at 3.50GHz with 32 GB of RAM and 8 MB of cache. The operating system is 64 bits Linux. The program was compiled with gcc using the `-O4` optimization option.

For the performance (and for the KAT in the next section) we selected three sets of parameters corresponding respectively to the security levels 1, 3, and 5.

- `BIG_QUAKE_1`, corresponding to $(m, n, \ell, t) = (12, 3510, 13, 91)$.
- `BIG_QUAKE_3`, corresponding to $(m, n, \ell, t) = (18, 7410, 19, 152)$.
- `BIG_QUAKE_5`, corresponding to $(m, n, \ell, t) = (18, 10070, 19, 190)$.

7.1 Running time in Milliseconds

	`BIG_QUAKE_1`	`BIG_QUAKE_3`	`BIG_QUAKE_5`
Key Generation	268	2 469	4 717
Encapsulation	1.23	3.00	4.46
Decapsulation	1.41	9.11	13.7

7.2 Space Requirements in Bytes

	`BIG_QUAKE_1`	`BIG_QUAKE_3`	`BIG_QUAKE_5`
Public Key	25 482	84 132	149 800
Secret Key	14 772	30 860	41 804
Ciphertext	201	406	492

8 Known Answer Tests – KAT

The KAT file are available in the submission package for `BIG_QUAKE_1`, `BIG_QUAKE_3`, and `BIG_QUAKE_5`:

- `KAT/PQCkemKAT_BIG_QUAKE_1.req`
- `KAT/PQCkemKAT_BIG_QUAKE_1.rsp`
- `KAT/PQCkemKAT_BIG_QUAKE_3.req`
- `KAT/PQCkemKAT_BIG_QUAKE_3.rsp`
- `KAT/PQCkemKAT_BIG_QUAKE_5.req`
- `KAT/PQCkemKAT_BIG_QUAKE_5.rsp`

For each KAT we generated 10 samples.

References

Appendix

A How to get systematic blockwise circulant parity check matrix?

Note that Condition 1 page 9 is not necessarily satisfied. However, it is in general possible to deduce from a general quasi–cyclic Goppa code another QC Goppa code satisfying this condition by applying a permutation preserving the quasi–cyclicity, i.e. a block–wise permutation. Hence, we introduce a second condition

Condition 2. The quasi–cyclic code has an information set which is a disjoint union of blocks.

Clearly, a quasi–cyclic code satisfying Condition 2 can provide after a blockwise permutation a quasi–cyclic code satisfying Condition 1.

Algorithm 3, permits the computation of such a block–wise permutation if it exists. It returns FALSE, if such a permutation is not found which happens for instance if Condition 2 is not satisfied. Applying this algorithm on quasi–cyclic Goppa codes as defined in § 2.8, then after 5000 experiments on quasi–cyclic Goppa codes of various parameters, the algorithm never returned FALSE.

As an input of the algorithm, we need a parity–check matrix $\boldsymbol{H}_0$ which is blockwise circulant. More precisely, the rows of $\boldsymbol{H}_0$ are of the form $c_0, \sigma_\ell(c_0), \ldots, {\sigma_\ell}^{\ell-1}(c_0)$, $c_1, \sigma_\ell(c_1), \ldots,$ ${\sigma_\ell}^{\ell-1}(c_1)$, $\ldots, c_s, \sigma_\ell(c_s), \ldots, {\sigma_\ell}^{\ell-1}(c_s)$. The matrix need not be full rank.

Algorithm 3: Checking Condition 1

Input : A block–wise circulant parity–check matrix $\boldsymbol{H}_0 \in \mathbb{F}_2^{(n-k)\times n}$ of an ℓ–QC Goppa code of length n and dimension k with $\ell | n, k$

Output: Returns FALSE if no block permutation is found. Else, returns TRUE together with

- a blockwise permutation τ to get a systematic code;
- a systematic blockwise circulant parity–check matrix $\boldsymbol{H}$ of the permuted code.

Note. *The Matrix $\boldsymbol{H}_0$ is split into $\ell \times \ell$ square blocks. They are denoted by B_{ij} for $0 \leqslant i < \frac{n-k}{\ell}$ and $0 \leqslant j < \frac{n}{\ell}$. Similarly, the blocks of ℓ rows are denoted by L_i and the blocks of columns by C_j;*
$\tau \leftarrow$ Id (Identity permutation on $\{0, \ldots, n-1\}$);
$\boldsymbol{H} \leftarrow \boldsymbol{H}_0$;
for *i from 0 to $\frac{n-k}{\ell} - 1$* **do**
  if *There exists t, $i \leqslant t < \frac{n-k}{\ell}$ such that B_{ti} is invertible* **then**
    $B \leftarrow B_{ti}$;
    Swap block–rows L_i and L_t;
    $L_i \leftarrow B^{-1} L_i$;
    Eliminate blocks below and above the (i,i)–th one by Gaussian elimination;
  end
  else if *There exists t, $i \leqslant t < \frac{n-k}{\ell}$ and j, $i < j < n$ such that B_{tj} is invertible* **then**
    Swap columns C_i and C_j in $\boldsymbol{H}$;
    $\tau \leftarrow \tau_{ij} \circ \tau$ (τ_{ij} denotes the transposition of i, j);
    Swap rows L_i and L_t in $\boldsymbol{H}$;
    $L_i \leftarrow B^{-1} L_i$;
    Eliminate blocks below and above the (i,i)–th one by Gaussian elimination in $\boldsymbol{H}$;
  end
  else
    return FALSE;
  end
end
return TRUE, $\boldsymbol{H}$, τ;

B Proof of Proposition 8

Let us first recall the proposition we want to prove

Proposition 8. *Let f be a Boolean function which is equal to 1 on a fraction α of inputs which can be implemented by a quantum circuit of depth D_f and whose gate complexity is C_f. Using Grover's algorithm for finding an input x of f for which $f(x) = 1$ can not take less quantum resources than a Grover's attack on AES-N as soon as*

$$\frac{D_f \cdot C_f}{\alpha} \geqslant 2^N D_{AES-N} \cdot C_{AES-N}$$

where D_{AES-N} and C_{AES-N} are respectively the depth and the complexity of the quantum circuit implementing AES-N.

Proof. Following Zalka[**?**], the best way is to perform Grover's algorithm sequentially with the maximum allowed number of iterations in order not to go beyond MAXDEPTH. Grover's algorithm consists of iterations of the following procedure:

- Apply $U : |0\rangle|0\rangle \to \sum_{x\in\{0,1\}^n} \frac{1}{2^{n/2}}|x\rangle|f(x)\rangle$.
- Apply a phase flip on the second register to get $\sum_{x\in\{0,1\}^n} \frac{1}{2^{n/2}}(-1)^{f(x)}|x\rangle|f(x)\rangle$.
- Apply $U^\dagger$.

If we perform I iterations of the above for $I \leqslant \frac{1}{\sqrt{\alpha}}$ then the winning probability is upper bounded by αI^2. In our setting, we can perform $I = \frac{\mathsf{MAXDEPTH}}{D_f}$ sequentially before measuring, and each iteration costs time C_f. At each iteration, we succeed with probability αI^2 and we need to repeat this procedure $\frac{1}{\alpha I^2}$ times to get a result with constant probability. From there, we conclude that the total complexity Q is:

$$Q = \frac{1}{\alpha I^2} \cdot I \cdot C_f = \frac{D_f \cdot C_f}{\alpha \mathsf{MAXDEPTH}}. \tag{6}$$

A similar reasoning performed on using Grover's search on AES-N leads to a quantum complexity

$$Q_{AES-N} = \frac{2^N D_{AES-N} \cdot C_{AES-N}}{\mathsf{MAXDEPTH}}. \tag{7}$$

The proposition follows by comparing (6) with (7). □

C Proof of Lemma 6

Remind that $m_r(2^m)$ denotes the number of irreducible polynomials of degree r in $\mathbb{F}_{2^m}[z]$ and $s_r(2^m)$ denotes the number of irreducible polynomials of degree r such that $g(z^\ell)$ is irreducible.

Clearly for $g(z^\ell)$ to be irreducible $g(z)$ should be irreducible too. Conversely, if $g(z)$ is irreducible, and $g(z^\ell)$ reducible, then the factorization of $g(z^\ell)$ is of the form

$$g(z^\ell) = \prod_{i=0}^{\ell-1} h(\zeta^i z) \tag{8}$$

for some irreducible polynomial h. Remind that ζ denotes a primitive ℓ–th root of unity in $\mathbb{F}_{2^m}$. Indeed, the finite subgroup of order ℓ of the affine group spanned by the map $z \mapsto \zeta z$ acts on polynomials as $f(z) \mapsto f(\zeta z)$. Under this action, $g(z^\ell)$ is fixed, hence the polynomials of its irreducible decomposition form an orbit under this action. Moreover, since ℓ is prime, the orbit has size ℓ.

Thus, the polynomials $g(z)$ such that $g(z^\ell)$ is reducible has the form (8). The number of such polynomials is bounded below by $m_r(2^m)/\ell$ which leads to

$$s_r(2^m) \geqslant \left(1 - \frac{1}{\ell}\right) m_r(2^m).$$

Remark 8. Actually one could prove that $s_r(2^m)$ is defined by the following recursive formula:

$$s_r(2^m) = \begin{cases} 2^m - 1 & \text{if} \quad r = 1 \\ m_r(2^m)(1 - \frac{1}{\ell}) & \text{if} \quad \ell \nmid r \\ m_r(2^m) - \frac{1}{\ell}(m_r(2^m) - s_{r/\ell}(2^r)) & \text{else.} \end{cases}$$