DAGS:
Key Encapsulation from Dyadic GS Codes

Gustavo Banegas[1], Paulo S. L. M. Barreto[2], Brice Odilon Boidje[3], Pierre-Louis Cayrel[4], Gilbert Ndollane Dione[3], Kris Gaj[7], Cheikh Thiécoumba Gueye[3], Richard Haeussler[7], Jean Belo Klamti[3], Ousmane N'diaye[3], Duc Tri Nguyen[7], Edoardo Persichetti[5], and Jefferson E. Ricardini[6]

[1]Technische Universiteit Eindhoven, The Netherlands
[2]University of Washington Tacoma, USA
[3]Université Cheikh Anta Diop, Dakar, Senegal.
[4]Laboratoire Hubert Curien, Saint-Etienne, France
[5]Florida Atlantic University, USA
[6]Universidade de São Paulo, Brazil
[7]George Mason University, USA

The team listed above is the principal submitter; there are no auxiliary submitters.

Owner, inventors and developers of this submission are the same as the principal submitter. Relevant prior work is credited where appropriate.

Email Address (preferred): epersichetti@fau.edu

Postal Address and Telephone (if absolutely necessary):
Edoardo Persichetti, Department of Mathematical Sciences, 777 Glades Rd, Boca Raton, FL, 33431, +1 561 297 4136.

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

Contents

1 Introduction

Code-based cryptography is one of the main candidates for post-quantum cryptography standardization. The area is largely based on the Syndrome Decoding Problem [12], which shows to be strong against quantum attacks. Over the years, since McEliece's seminal work [33], many cryptosystems have been proposed, trying to balance security and efficiency. In particular dealing with inherent flaws such as the large size of the public keys. In fact, while McEliece's cryptosystem, which is based binary Goppa codes, is still unbroken, it features a key of several kilobytes, which has effectively prevented its use in many applications.

There are currently two main trends to deal with this issue, and they both involve structured matrices: the first, is based on "traditional" algebraic codes such as Goppa or Srivastava codes; the second suggests to use sparse matrices as in LDPC/MDPC codes. This work builds on the former approach, initiated in 2009 by Berger et al. [11], who proposed Quasi-Cyclic (QC) codes, and Misoczki and Barreto [34], suggesting Quasi-Dyadic (QD) codes instead (later generalized to Quasi-Monoidic (QM) codes [10]). Both proposals feature very compact public keys due to the introduction of the extra algebraic structure, but unfortunately this also leads to a vulnerability. Indeed, Faugère, Otmani, Perret and Tillich [23] devised a clever attack (known simply as FOPT) which exploits the algebraic structure to build a system of equations, which can successively be solved using Gröbner bases techniques. As a result, the QC proposal is definitely compromised, while the QD/QM approach needs to be treated with caution. In fact, for a proper choice of parameters, it is still possible to design secure schemes using for instance binary Goppa codes, or Generalized Srivastava (GS) codes as suggested by Persichetti in [38].

In this document, we present DAGS, a Key Encapsulation Mechanism (KEM) that follows the QD approach using GS codes. To the best of our knowledge, this is the first code-based KEM that uses structured algebraic codes. The KEM achieves IND-CCA security following the recent framework by Kiltz et al. [29], and features compact public keys and efficient encapsulation and decapsulation algorithms. We modulate our parameters to achieve the most efficient scheme, while at the same time avoiding the FOPT attack.

2 Notation

This section describes the notation used in this document.

a	a constant
$\boldsymbol{a}$	a vector
A	a matrix
$\mathcal{A}$	an algorithm or (hash) function
A	a set
$\mathrm{Diag}(\boldsymbol{a})$	the diagonal matrix formed by the vector $\boldsymbol{a}$
I_n	the $n \times n$ identity matrix
$\xleftarrow{\$}$	choosing a random element from a set or distribution

2.1 Formats and Conventions

DAGS operates on vectors of elements of the finite field $\mathbb{F}_q$, where q is a power of 2 as specified by the choice of parameters.

1. Finite field elements are represented as bit strings using standard log/antilog tables (see for instance [32, Ch. 4, §5]) which are stored in the memory.

2. Field operations are performed using the log/antilog tables, and implemented in an isochronous way.

3. Every vector or matrix defined over an extension field $\mathbb{F}_{q^m}$ can be *projected* onto the base field $\mathbb{F}_q$ by replacing each element with the (column) vector formed by the coefficients of its representation over $\mathbb{F}_q$.

4. We use the hash function SHA3-512 with 256 bits input for the random oracles $\mathcal{G}, \mathcal{H}$ and $\mathcal{K}$ (see Section 3.1).

3 Full Protocol Specification (2.B.1)

3.1 Design Rationale

In this section we introduce the three algorithms that form DAGS. System parameters are the code length n and dimension k, the values s and t which define a GS code, the cardinality of the base field q and the degree of the field extension m. In addition, we have $k = k' + k''$, where k' is arbitrary and is set to be "small". In practice, the value of k' depends on the base field and is such that a vector of length k' provides at least 256 bits of entropy.

The key generation process uses the following fundamental equation

$$\frac{1}{h_{i\oplus j}} = \frac{1}{h_i} + \frac{1}{h_j} + \frac{1}{h_0}. \tag{1}$$

to build the vector $\boldsymbol{h} = (h_0, \ldots, h_{n-1})$ of elements of $\mathbb{F}_{q^m}$, which is known as *signature* of a dyadic matrix. This is then used to form a Cauchy matrix, i.e. a matrix $C(\boldsymbol{u}, \boldsymbol{v})$ with components $C_{ij} = \frac{1}{u_i - v_j}$. The matrix is then successively powered (element by element) forming several blocks which are superimposed and then multiplied by a random diagonal matrix. Finally, the resulting matrix is projected onto the base field and row-reduced to systematic form. The overall process is described below.

3.1.1 Key Generation

1. Generate dyadic signature $\boldsymbol{h}$. To do this:
   i. Choose random non-zero distinct h_0 and h_j for $j = 2^l, l = 0, \ldots, \lfloor \log q^m \rfloor$.
   ii. Form the remaining elements using (1).
   iii. Return a selection[1] of blocks of dimension s up to length n.
2. Build the Cauchy support. To do this:
   i. Choose a random[2] offset $\omega \xleftarrow{\$} \mathbb{F}_{q^m}$.
   ii. Compute $u_i = \dfrac{1}{h_i} + \omega$ for $i = 0, \ldots, s-1$.
   iii. Compute $v_j = \dfrac{1}{h_j} + \dfrac{1}{h_0} + \omega$ for $j = 0, \ldots, n-1$.
   iv. Set $\boldsymbol{u} = (u_0, \ldots, u_{s-1})$ and $\boldsymbol{v} = (v_0, \ldots, v_{n-1})$.
3. Form Cauchy matrix $\hat{H}_1 = C(\boldsymbol{u}, \boldsymbol{v})$.
4. Build $\hat{H}_i$, $i = 2, \ldots t$, by raising each element of $\hat{H}_1$ to the power of i.
5. Superimpose blocks $\hat{H}_i$ in ascending order to form matrix $\hat{H}$.
6. Generate vector $\boldsymbol{z}$ by sampling uniformly at random elements in $\mathbb{F}_{q^m}$ with the restriction $z_{is+j} = z_{is}$ for $i = 0, \ldots, n_0 - 1$, $j = 0, \ldots, s-1$.

[1] Making sure to exclude any block containing an undefined entry.
[2] See Appendix A for restrictions about the choice of the offset.

7. Set $y_j = \dfrac{z_j}{\prod_{i=0}^{s-1}(u_i - v_j)^t}$ for $j = 0, \ldots, n-1$ and $\boldsymbol{y} = (y_0, \ldots, y_{n-1})$.

8. Form $H = \hat{H} \cdot \mathrm{Diag}(\boldsymbol{z})$.

9. Project H onto $\mathbb{F}_q$ using the co-trace function: call this H_{base}.

10. Write H_{base} in systematic form $(M \mid I_{n-k})$.

11. The public key is the generator matrix $G = (I_k \mid M^T)$.

12. The private key is the pair $(\boldsymbol{v}, \boldsymbol{y})$.

The encapsulation and decapsulation algorithms make use of two hash functions[3] $\mathcal{G} : \mathbb{F}_q^{k'} \to \mathbb{F}_q^k$ and $\mathcal{H} : \mathbb{F}_q^{k'} \to \mathbb{F}_q^{k'}$, the former with the task of generating randomness for the scheme, the latter to provide "plaintext confirmation" as in [29]. The shared symmetric key is obtained via another hash function $\mathcal{K} : \{0,1\}^* \to \{0,1\}^\ell$, where ℓ is the desired key length.

3.1.2 Encapsulation

1. Choose $\boldsymbol{m} \xleftarrow{\$} \mathbb{F}_q^{k'}$.

2. Compute $\boldsymbol{r} = \mathcal{G}(\boldsymbol{m})$ and $\boldsymbol{d} = \mathcal{H}(\boldsymbol{m})$.

3. Parse $\boldsymbol{r}$ as $(\boldsymbol{\rho} \parallel \boldsymbol{\sigma})$ then set $\boldsymbol{\mu} = (\boldsymbol{\rho} \parallel \boldsymbol{m})$.

4. Generate error vector $\boldsymbol{e}$ of length n and weight w from $\boldsymbol{\sigma}$.

5. Compute $\boldsymbol{c} = \boldsymbol{\mu} G + \boldsymbol{e}$.

6. Compute $\boldsymbol{k} = \mathcal{K}(\boldsymbol{m})$.

7. Output ciphertext $(\boldsymbol{c}, \boldsymbol{d})$; the encapsulated key is $\boldsymbol{k}$.

The decapsulation algorithm consists mainly of decoding the noisy codeword received as part of the ciphertext. This is done using the alternant decoding algorithm described in [32, Ch. 12, §9] and requires the parity-check matrix to be in alternant form.

[3] As specified in Section 2.1

3.1.3 Decapsulation

1. Get parity-check matrix H' in alternant form from private key[4].
2. Use H' to decode $\boldsymbol{c}$ and obtain codeword $\boldsymbol{\mu}'G$ and error $\boldsymbol{e}'$.
3. Output $\perp$ if decoding fails or $\mathsf{wt}(\boldsymbol{e}') \neq w$
4. Recover $\boldsymbol{\mu}'$ and parse it as $(\boldsymbol{\rho}' \parallel \boldsymbol{m}')$.
5. Compute $\boldsymbol{r}' = \mathcal{G}(\boldsymbol{m}')$ and $\boldsymbol{d}' = \mathcal{H}(\boldsymbol{m}')$.
6. Parse $\boldsymbol{r}'$ as $(\boldsymbol{\rho}'' \parallel \boldsymbol{\sigma}')$.
7. Generate error vector $\boldsymbol{e}''$ of length n and weight w from $\boldsymbol{\sigma}'$.
8. If $\boldsymbol{e}' \neq \boldsymbol{e}'' \vee \boldsymbol{\rho}' \neq \boldsymbol{\rho}'' \vee \boldsymbol{d} \neq \boldsymbol{d}'$ output $\perp$.
9. Else compute $\boldsymbol{k} = \mathcal{K}(\boldsymbol{m}')$.
10. The decapsulated key is $\boldsymbol{k}$.

DAGS is built upon the McEliece encryption framework, with a notable exception. In fact, we incorporate the "randomized" version of McEliece by Nojima et al. [37] into our scheme. This is extremely beneficial for two distinct aspects: first of all, it allows us to use a much shorter vector $\boldsymbol{m}$ to generate the remaining components of the scheme, greatly improving efficiency. Secondly, it allows us to get tighter security bounds. In fact, a shorter input makes all the hash functions easy to compute, and minimizes the overhead due to the IND-CCA2 security in the QROM. Note that our protocol differs slightly from the paradigm presented in [29], in the fact that we don't perform a full re-encryption in the decapsulation algorithm. Instead, we simply re-generate the randomness and compare it with the one obtained after decoding. This is possible since, unlike a generic PKE, McEliece decryption reveals the randomness used, in our case $\boldsymbol{e}$ (and $\boldsymbol{\rho}$). It is clear that if the re-generated randomness is equal to the retrieved one, the resulting encryption will also be equal. This allows us to further decrease computation time.

[4]See Algorithm 6.3.1.

4 Security (2.B.4)

In this section, we discuss some aspects of provable security, and in particular we show that DAGS satisfies the notion of IND-CCA security for KEMs. Before discussing the IND-CCA security of DAGS, we show that the underlying PKE (i.e. Randomized McEliece) satisfies the γ-spread property. This will allow us to get better security bounds in our reduction.

Definition 1 *Consider a probabilistic PKE with randomness set* R*. We say that PKE is* γ-spread *if for a given key pair* (sk, pk)*, a plaintext* m *and an element* y *in the ciphertext domain, we have*

$$\mathsf{Pr}[r \xleftarrow{\$} \mathsf{R} \mid y = \mathsf{Enc}_{pk}(m, r)] \leq 2^{-\gamma},$$

for a certain $\gamma \in \mathbb{R}$*.*

The definition above is presented as in [29], but note that in fact this corresponds to the notion of γ-*uniformity* given by Fujisaki and Okamoto in [26], except for a change of constants. In other words, a scheme is γ-spread if it is $2^{-\gamma}$-uniform.

It was proved in [19] that a simple variant of the (classic) McEliece PKE is γ-uniform for $\gamma = 2^{-k}$, where k is the code dimension as usual (more in general, $\gamma = q^{-k}$ for a cryptosystem defined over $\mathbb{F}_q$). We can extend this result to our scheme as follows.

Lemma 1 *Randomized McEliece is* γ*-uniform for* $\gamma = \dfrac{q^{-k''}}{\binom{n}{w}}$.

Proof Let $\boldsymbol{y}$ be a generic vector of $\mathbb{F}_q^n$. Then either $\boldsymbol{y}$ is a word at distance w from the code, or it isn't. If it isn't, the probability of $\boldsymbol{y}$ being a valid ciphertext is clearly exactly 0. On the other hand, suppose $\boldsymbol{y}$ is at distance w from the code; then there is only one choice of $\boldsymbol{\rho}$ and one choice of $\boldsymbol{e}$ that satisfy the equation (since w is below the GV bound), i.e. the probability of $\boldsymbol{y}$ being a valid ciphertext is exactly $1/q^{k''} \cdot 1/\binom{n}{w}$, which concludes the proof. □

We are now ready to present the security results.

Theorem 1 *Let* $\mathcal{A}$ *be an IND-CCA adversary against DAGS that makes at most* $q_{RO} = q_{\mathcal{G}} + q_{\mathcal{K}}$ *total random oracle queries*[5] *and* q_D *decryption queries. Then there exists an IND-CPA adversary* $\mathcal{B}$ *against PKE, running in approximately the same time as* $\mathcal{A}$*, such that*

$$\mathsf{Adv}_{KEM}^{IND-CCA}(\mathcal{A}) \leq q_{RO} \cdot 2^{-\gamma} + 3 \cdot \mathsf{Adv}_{PKE}^{IND-CPA}(\mathcal{B}).$$

Proof The thesis is a consequence of the results presented in Section 3.3 of [29]. In fact, our scheme follows the $\mathsf{KEM}_m^{\perp}$ framework that consists of applying two generic transformations to a public-key encryption scheme. The first step consists of transforming the IND-CPA encryption scheme into a OW-PCVA (i.e. Plaintext and Validity Checking) scheme. Then, the resulting scheme is transformed into a KEM in a "standard" way. Both proofs are obtained via a sequence of games, and the combination of them shows that breaking IND-CCA security of the KEM would lead to break the IND-CPA security of the underlying encryption scheme. Note that Randomized McEliece, instantiated with Quasi-Dyadic GS codes, presents no correctness error (the value δ in [29]), which greatly simplifies the resulting bound. □

The value $\boldsymbol{d}$ included in the KEM ciphertext does not contribute to the security result above, but it is a crucial factor to provide security in the Quantum Random Oracle Model (QROM). We present this in the next theorem.

Theorem 2 *Let $\mathcal{A}$ be a quantum IND-CCA adversary against DAGS that makes at most $q_{RO} = q_{\mathcal{G}} + q_{\mathcal{K}}$ total quantum random oracle queries*[6] *and q_D (classical) decryption queries. Then there exists a OW-CPA adversary $\mathcal{B}$ against PKE, running in approximately the same time as $\mathcal{A}$, such that*

$$\mathsf{Adv}_{KEM}^{IND-CCA}(\mathcal{A}) \leq 8q_{RO} \cdot \sqrt{q_{RO} \cdot \sqrt{\mathsf{Adv}_{PKE}^{OW-CPA}(\mathcal{B})}}.$$

Proof The thesis is a consequence of the results presented in Section 4.4 of [29]. In fact, our scheme follows the $\mathsf{QKEM}_m^{\perp}$ framework that consists of applying two generic transformations to a public-key encryption scheme. The first step transforming the IND-CPA encryption scheme into a OW-PCVA (i.e. Plaintext and Validity Checking) scheme, is the same as in the previous case. Now, the resulting scheme is transformed into a KEM with techniques suitable for the QROM. The combination of the two proofs shows that breaking IND-CCA security of the KEM would lead to break the OW-CPA security of the underlying encryption scheme. Note, therefore, that the IND-CPA security of the underlying PKE has in this case no further effect on the final result, and can be considered instead just a guarantee that the scheme is indeed OW-CPA secure. The bound obtained is a "simplified" and "concrete" version (as presented by the authors) and, in particular, it is easy to notice that it does not depend on the number of queries $q_{\mathcal{H}}$ presented to the random oracle $\mathcal{H}$. The bound is further simplified since, as above, the underlying PKE presents no correctness error. □

[5] Respectively $q_{\mathcal{G}}$ queries to the random oracle $\mathcal{G}$ and $q_{\mathcal{K}}$ queries to the random oracle $\mathcal{K}$.

[6] Same as in Theorem 1.

5 Known Attacks and Parameters (2.B.5/2.B.1)

We start by briefly presenting the hard problem on which DAGS is based, and then discuss the main attacks on the scheme and related security concerns.

5.1 Hard Problems from Coding Theory

Most of the code-based cryptographic constructions are based on the hardness of the following problem, known as the (q-ary) *Syndrome Decoding Problem (SDP).*

Problem 1 *Given an $(n-k) \times n$ full-rank matrix H over $\mathbb{F}_q$, a vector $\boldsymbol{s} \in \mathbb{F}_q^{n-k}$, and a non-negative integer w, find a vector $\boldsymbol{e} \in \mathbb{F}_q^n$ of weight w such that $H\boldsymbol{e}^T = \boldsymbol{s}$.*

The corresponding decision problem was proved to be NP-complete in 1978 [12], but only for binary codes. In 1994, A. Barg proved that this result holds for codes over all finite fields ([7], in Russian, and [8, Theorem 4.1]).

In addition, many schemes (including the original McEliece proposal) require the following computational assumption.

Assumption 1 *The public matrix output by the key generation algorithm is computationally indistinguishable from a uniformly chosen matrix of the same size.*

The assumption above is historically believed to be true, except for very particular cases. For instance, there exists a distinguisher (Faugère et al. [22]) for cryptographic protocols that make use of high-rate Goppa codes (like the CFS signature scheme [20]). Moreover, it is worth mentioning that the "classical" methods for obtaining an indistinguishable public matrix, such as the use of scrambling matrices S and P, are rather outdated and unpractical and can introduce vulnerabilities to the scheme as per the work of Strenzke et al. ([42, 43]). Thus, traditionally, the safest method (Biswas and Sendrier, [16]) to obtain the public matrix is simply to compute the systematic form of the private matrix.

5.2 Decoding Attacks

The main approach for solving SDP is the technique known as Information Set Decoding (ISD), first introduced by Prange [41]. Among several variants and generalizations, Peters showed [40] that it is possible to apply Prange's approach to

generic q-ary codes. Other approaches such as Statistical Decoding [30, 35] are usually considered less efficient. Thus, when choosing parameters, we will focus mainly on defeating attacks of the ISD family.

Hamdaoui and Sendrier in [28] provide non-asymptotic complexity estimates for ISD in the binary case. For codes over $\mathbb{F}_q$, instead, a bound is given in [36], which extends the work of Peters. For a practical evaluation of the ISD running times and corresponding security level, we used Peters's ISDFQ script[1].

Quantum Speedup. Bernstein in [13] shows that Grover's algorithm applies to ISD-like algorithms, effectively halving the asymptotic exponent in the complexity estimates. Later, it was proven in [31] that several variants of ISD have the potential to achieve a better exponent, however the improvement was disappointingly away from the factor of 2 that could be expected. For this reason, we simply treat the best quantum attack on our scheme to be "traditional" ISD (Prange) combined with Grover search.

5.3 Algebraic Attacks

While, as we discussed above, recovering a private matrix from a public one can be in general a very difficult problem, the presence of extra structure in the code properties can have a considerable effect in lowering this difficulty.

A very effective structural attack was introduced by Faugère, Otmani, Perret and Tillich in [23]. The attack (for convenience simply called FOPT) relies on the property $H \cdot G^T = 0$ to build an algebraic system, using then Gröbner bases techniques to solve it and recover the private key. Note that this property is valid for every linear code, but it is the special properties of structured alternant codes that make the system solvable, as they contribute to considerably reduce the number of variables.

The attack was originally aimed at two variants of McEliece, introduced respectively in [11] and [34]. The first variant, using quasi-cyclic codes, was easily broken in all proposed parameters. The second variant, instead, only considered quasi-dyadic Goppa codes. In this case, most of the parameters proposed have also been broken, except for the binary case (i.e. base field $\mathbb{F}_2$). This was, in truth, not connected to the base field per se, but rather depended on the fact that, with a smaller base field, the authors provided a much higher extension degree m, as they were keeping constant the value $q^m = 2^{16}$. As it turns out, the extension degree m plays a key role in evaluating the complexity of the attack.

5.3.1 Attack Complexity

Following up on their own work, the authors in [24] analyze the attack in detail with the aim of evaluating its complexity at least somewhat rigorously. At the core of the attack, there is an affine bi-linear system, which is derived from the initial system of equations by applying various algebraic relations due to the quasi-dyadic structure. This bi-linear system has $n_{X'}+n_{Y'}$ variables, where these are, respectively, the number of X and Y "free" variables (after applying the relations) of an alternant parity-check matrix H with $H_{ij} = Y_j X_j^i$. Moreover, the *degree of regularity* (i.e. the maximal degree of the polynomials appearing during the computation) is bounded above by $1 + \min\{n_{X'}, n_{Y'}\}$. It is shown that this number dominates computation time, and so it is crucial to correctly evaluate it in our case. In fact, for the original proposal based on Goppa codes [34], we have $n_{X'} = n_0 - 2 + \log_2(\ell)$, where ℓ is the dyadic order and $n_0 = n/\ell$ is the number of dyadic blocks, and $n_{Y'} = m - 1$. We report an excerpt of some numbers from the paper in Table 1 below.

Table 1: Details of FOPT applied to Quasi-Dyadic Goppa codes [24].

q	m	n	k	n_0	ℓ	$n_{X'}$	$n_{Y'}$	Time/Operations
2	16	3584	1536	56	2^6	60	15	N/A
2^2	8	3584	1536	56	2^6	60	7	1,776.3 sec / $2^{34.2}$ op
2^4	4	2048	1024	32	2^6	36	3	0.5 sec / $2^{22.1}$ op
2^8	2	1280	768	20	2^6	24	1	0.03 sec / $2^{16.7}$ op
2^8	2	640	512	10	2^6	14	1	0.03 sec / $2^{15.9}$ op
2^8	2	768	512	6	2^7	11	1	0.02 sec / $2^{15.4}$ op
2^8	2	1024	512	4	2^8	10	1	0.11 sec / $2^{19.2}$ op
2^8	2	512	256	4	2^7	9	1	0.06 sec / $2^{17.7}$ op
2^8	2	640	384	5	2^7	10	1	0.02 sec / $2^{14.5}$ op
2^8	2	768	512	6	2^7	11	1	0.01 sec / $2^{16.6}$ op
2^8	2	1280	768	5	2^8	11	1	0.05 sec / $2^{17.5}$ op
2^8	2	1536	1024	6	2^8	12	1	0.06 sec / $2^{17.8}$ op
2^4	4	4096	3584	32	2^7	37	3	7.1 sec / $2^{26.1}$ op
2^8	2	3072	2048	6	2^9	13	1	0.15 sec / $2^{19.7}$ op

It is possible to observe several facts. For instance, in every set of parameters, $n_{X'} >> n_{Y'}$ and so $n_{Y'} = m - 1$ is the most important number here. In other words, the degree of the extension field is crucial in evaluating the complexity of the attack, as we mentioned above. As a confirmation, it is easy to notice that all

parameters were broken very easily when this is extremely small (1 in most cases), while the running time scales accordingly when m grows. In fact, the attack couldn't be performed in practice on the first set of parameters (hence the N/A).

The first three groups of parameters are taken from, respectively, Table 2, Table 3 and Table 5 of the preliminary (unpublished) version of [34], while the last group consists of some *ad hoc* parameters generated by the FOPT authors. It stands out the absence of parameters from Table 4 of [34]: in fact, all of these parameters used $\mathbb{F}_2$ as based field and thus couldn't be broken (at least not without very long computations), just like for the case of the first set. As a result, an updated version of [34] was produced for publication, in which the insecure parameters are removed and only the binary sets (those of Table 4) appear.

Towards the end of [24], the authors present a bound on the theoretical complexity of computing a Gröbner base of the affine bi-linear system which is at the core of the attack. They then evaluate this bound and compare it with the number of operations required in practice (last column of Table 1). The bound is given by

$$T_{theo} \approx \left(\sum_{\substack{d_1+d_2=D \\ 1 \leq d_1,d_2 \leq D-1}} \Big(\dim R_{d_1,d_2} - [t_1^{d_1} t_2^{d_2}] HS(t_1,t_2) \Big) \dim R_{d_1,d_2} \right) \tag{2}$$

where D is the degree of regularity of the system, $\dim R_{d_1,d_2} = \binom{d_1+n_{X'}}{d_1}\binom{d_2+n_{X'}}{d_2}$ and $[t_1^{d_1} t_2^{d_2}] HS(t_1,t_2)$ indicates the coefficient of the term $[t_1^{d_1} t_2^{d_2}]$ in the Hilbert bi-series $HS(t_1,t_2)$, as defined in Appendix A of [24].

As it turns out this bound is quite loose, being sometimes above and sometimes below the experimental results, depending on which set of parameters is considered. As such, it is to be read as a grossly approximate indication of the expected complexity of a parameter set, and it only allows to have a rough idea of the security provided for each set. Nevertheless, since are able to compute the bound for all DAGS proposed parameters, we will keep this number in mind when proposing parameters (Section 5.4), to make sure our choices are at least not obviously insecure.

As a bottomline, it is clear that the complexity of the attack scales somewhat proportionally to the value $m-1$ which defines the dimension of the solution space. The FOPT authors point out that any scheme for which this dimension is less or equal to 20 should be within the scope of the attack.

Since GS codes are also alternant codes, the attack can be applied to our proposal as well. There is, however, one very important difference to keep in mind. In fact, it is shown in [38] that, thanks to the particular structure of GS codes, the dimension of the solution space is defined by $mt - 1$, rather than $m - 1$. This provides greater flexibility when designing parameters for the code, and it allows, in particular, to "rest the weight" of the attack on two shoulders rather than just one. Thus we are able to modulate the parameters and keep the extension degree m small while still achieving a large dimension for the solution space. We will discuss parameter selection in detail in Section 5.4 as already mentioned.

5.3.2 Folded Codes

Recently, an extension of the FOPT attack appeared in [25]. The authors introduce a new technique called "folding", and show that it is possible to reduce the complexity of the FOPT attack to the complexity of attacking a smaller code (the "folded" code), thanks to the strong properties of the automorphism group of the alternant codes in use. The attack turns out to be very efficient against Goppa codes, as it is possible to recover a folded code which is also a Goppa code. As a consequence, it is possible to tweak the attack to solve a different, augmented system of equations (named $\mathsf{G}_{X,Y'}$), rather than the "basic" one which is aimed at generic alternant codes (called $\mathsf{A}_{X,Y'}$). Moreover, this can be further refined in the case of *binary* Goppa codes, leading to a third system of equations referred to as $\mathsf{McE}_{X,Y'}$. In parallel, the authors present a new method called "structural elimination" that manages to eliminate a considerable number of variables, at the price of an increased degree in the equations considered. Solving the "eliminated" systems (called respectively $\mathsf{elimA}_{X',Y'}$, $\mathsf{elimG}_{X',Y'}$ and $\mathsf{elimMcE}_{X',Y'}$) often proves a more efficient choice, but the authors do occasionally use the non-eliminated systems when it is more convenient to do so.

The paper concentrates on attacking several parameters that were proposed for signature schemes and encryption schemes in various follow-ups of [11] and [34]. The latter includes, among others, some of the parameters presented in Table 1. While codes designed to work for signature schemes turn out to be very easy to attack (due to their particular nature), the situation for encryption is more complex. The authors are able to obtain a speedup in the attack times for previously investigated parameters, but some of the parameters could still not be solved in practice. We report the results below, where we indicate the type of system chosen to be solved, and we keep some of the previously-shown parameters for ease of comparison.

Table 2: Details of Folding attack applied to Quasi-Dyadic Goppa codes [25].

q	m	n	k	n_0	ℓ	System	Folding	FOPT
2^4	4	2048	1024	32	2^6	$\mathsf{elimA}_{X',Y'}$	0.01 sec	0.5 sec
2^4	4	4096	3584	32	2^7	$\mathsf{elimA}_{X',Y'}$	0.01 sec	7.1 sec
2^2	8	3584	1536	56	2^6	$\mathsf{elimA}_{X',Y'}$	0.04 sec	1,776.3 sec
2	16	4864	4352	152	2^5	$\mathsf{elimMcE}_{X',Y'}$	18 sec	N/A
2	12	3200	1664	25	2^7	$\mathsf{elimMcE}_{X',Y'}$	$\leq 2^{83.5}$ op	N/A
2	14	5376	3584	42	2^7	$\mathsf{elimMcE}_{X',Y'}$	$\leq 2^{96.1}$ op	N/A
2	15	11264	3584	22	2^9	$\mathsf{elimMcE}_{X',Y'}$	$\leq 2^{146}$ op	N/A
2	16	6912	2816	27	2^8	$\mathsf{elimMcE}_{X',Y'}$	$\leq 2^{168}$ op	N/A
2	16	8192	4096	32	2^8	$\mathsf{elimMcE}_{X',Y'}$	$\leq 2^{157}$ op	N/A

The authors don't report timings for codes that were already broken with FOPT in negligible time (like all of those where $m = 2$). Also, we've excluded from our table parameters that are not relevant to this submission, such as the quasi-monoidic codes of [10] (where q is not a power of 2).

This table confirms our intuition that high values of m result in a high number of operations, and that complexity increases somewhat proportionally to this value. Note that the last 5 sets of parameters were not broken in practice and the estimated complexity is always quite high: it is not clear what the authors mean by $\leq$, but it is reasonable to assume that the actual complexity wouldn't be dramatically smaller than the indicated value, and thus at least 2^{80} in all cases. Consequently, the claim that parameters with $m - 1 \leq 20$ are "within the scope of the attack" looks now perhaps a bit optimistic.

The fourth set of parameters seem to contradict our intuition, since it was broken in practice with relative ease even though $m = 16$. However, it is possible to see that this is a code with a ridiculously high rate (k/n is very close to 1) and in particular, the very large number of blocks n_0 clearly stands out. We remark that this set of parameters was chosen *ad hoc* by the attack authors and in practice such a poor choice of parameters would never be considered. Nevertheless, it gives us the confirmation (if needed) that high-rate codes are a bad choice not only for ISD-like attacks, but for structural attacks also.

The authors didn't present any explicit result against GS codes and, in particular, it is not known whether a folded GS code is still a GS code. Thus, the attack in this case is limited to solving the generic system $\mathsf{A}_{X,Y'}$ (or $\mathsf{elimA}_{X',Y'}$) and doesn't benefit

from the speedups which are specific to (binary) Goppa codes. For these reasons, and until an accurate complexity analysis is available, we choose to attain to the latest measurable guidelines and choose our parameters such that the dimension of the solution space for the algebraic system is strictly greater than 20. We then compute the bound given by Equation (2) and report it as an additional indication of the expected complexity of the attack. We hope that this work will encourage further study into FOPT and folding attacks in relation to GS codes.

5.3.3 Norm-Trace Codes

An attack based on *Norm-Trace Codes* has been recently introduced by Barelli and Couvreur [6]. As the name suggests, these codes are the result of the application of both the Trace and the Norm operation to a certain support vector, and they are alternant codes. In particular, they are subfield subcodes of Reed-Solomon codes. The construction of these codes is given explicitly only for the specific case $m = 2$ (as will be the case in all DAGS parameters), i.e. the support vector has components in $\mathbb{F}_{q^2}$, in which case the norm-trace code is defined as

$$\mathcal{NT}(\boldsymbol{x}) = \langle \mathbf{1}, Tr(\boldsymbol{x}), Tr(\alpha \boldsymbol{x}), N(\boldsymbol{x}) \rangle$$

where α is an element of trace 1.

The main idea of the attack is that there exists a specific norm-trace code that is the *conductor* of the secret subcode into the public code. By "conductor" the authors refer to the largest code for which the Schur product is entirely contained in the target, i.e.

$$Cond(\mathcal{D}, \mathcal{C}) = \{\boldsymbol{u} \in \mathbb{F}_q^n : \boldsymbol{u} \star \mathcal{D} \subseteq \mathcal{C}\}$$

where $\star$ denotes the Schur product (component-wise product of all codewords).

The authors present two strategies to determine the secret subcode. The first strategy is essentially an exhaustive search over all the codes of the proper co-dimension. This co-dimension is given by $2q/s$, since s is the size of the permutation group of the code, which is non-trivial in our case due to the code being quasi-dyadic. While such a brute force in principle would be too expensive, the authors present a few refinements that make it feasible, which include an observation on the code rate of the codes in use, and the use of shortened codes.

The second strategy, instead, consists of solving a bilinear system, which is obtained using the parity-check matrix of the public code and treating as unknowns the elements of a generator matrix for the secret code (as well as the support vector $\boldsymbol{x}$). This system is solved using Gröbner bases techniques, and benefits from a reduction in the number of variables similar to the one performed in FOPT, as well as the refinements mentioned above (shortened codes).

In any case, it is easy to deduce that the two parameters q and s are crucial in determining the cost of running this step of the attack, which dominates the overall cost. In fact, the authors are able to provide an accurate complexity analysis for the first strategy which confirms this intuition. The average number of iterations of the brute force search is given by q^{2c}, where c is exactly the co-dimension described above, i.e. $c = 2q/s$. In addition, it is shown that the cost of computing Schur products is $2n^3$ operations in the base field. Thus, the overall cost of the recovery step is $2n^3q^{4q/s}$ operations in $\mathbb{F}_q$. The authors then argue that wrapping up the attack has negligible cost, and that q-ary operations can be done in constant time (using tables) when q is not too big. All this leads to a complexity which is below the desired security level for all of the DAGS parameters that had been proposed at the time of submission. We report these numbers below.

Table 3: Early DAGS Parameters.

Security Level[7]	q	m	n	k	k'	s	t	w	Attack
1	2^5	2	832	416	43	2^4	13	104	2^{70}
3	2^6	2	1216	512	43	2^5	11	176	2^{80}
5	2^6	2	2112	704	43	2^6	11	352	2^{55}

As it is possible to observe, the attack complexity is especially low for the last set of parameters since the dyadic order s was chosen to be 2^6, and this is probably too much to provide security against this attack. Still, we point out that, at the time this parameters were proposed, there was no indication this was the case, since this attack is using an entirely new technique, and it is unrelated to the FOPT and folding attacks that we just described.

Unfortunately, the attack authors were not able to provide a security analysis for the second strategy (bilinear system). This is due to the fact that the attack uses Gröbner based techniques, and it is very hard to evaluate the cost in this case (similarly to what happened for FOPT). In this case then, the only evidence the

[7]Claimed.

authors provide is experimental, and based on running the attack in practice on all the parameters. The authors report running times around 15 minutes for the first set and less than a minute for the last, while they admit they weren't able to complete the execution in the middle case. This seems to match the evidence from the complexity results obtained for the first strategy, and suggests a speedup proportional to those. Further test runs are currently planned, but the fact that the attack already fails to run in practice for the middle set, gives us some confidence to believe that updated parameters will definitely make the attack infeasible.

5.4 Parameter Selection

To choose our parameters, we keep in mind all the remarks from the previous sections about decoding attacks and structural attacks. As we have just seen, we need to respect the condition $mt \geq 21$ to guarantee security against FOPT. At the same time, to prevent the BC attack q has to be chosen large enough and s can't be too big. Finally, for ISD to be computationally intensive we require a sufficiently large number w of errors to decode: this is given by $st/2$ according to the minimum distance of GS codes.

In addition, we tune our parameters to optimize performance. In this regard, the best results are obtained when the extension degree m is as small as possible. This, however, requires the base field to be large enough to accommodate sufficiently big codes (against ISD attacks), since the maximum size for the code length n is capped by $q^m - s$. Realistically, this means we want q^m to be at least 2^{12}, and the optimal choice in this sense seems to be $q = 2^8$ (see Section 6). Finally, note that s is constrained to be a power of 2, and that odd values of t seem to offer best performance.

Putting all the pieces together, we are able to present three set of parameters, in the following table. These correspond to three of the security levels indicated by NIST (first column), which are related to the hardness of performing a key search attack on three different variants of a block cipher, such as AES (with key-length respectively 128, 192 and 256). As far as quantum attacks are concerned, we claim that ISD with Grover (see above) will usually require more resources than a Grover search attack on AES for the circuit depths suggested by NIST (parameter MAXDEPTH). Thus, classical security bits are the bottleneck in our case, and as such we choose our parameters to provide 128, 192 and 256 bits of classical security for security levels 1, 3 and 5 respectively.

Table 4: Suggested DAGS Parameters.

Security Level	q	m	n	k	k'	s	t	w	$n_{Y'}$	BC
1	2^6	2	832	416	43	2^4	13	104	25	2^{126}
3	2^8	2	1216	512	32	2^5	11	176	21	2^{288}
5	2^8	2	1600	896	32	2^5	11	176	21	2^{289}

For practical reasons, during the rest of the paper we will refer to these parameters respectively as DAGS_1, DAGS_3 and DAGS_5.

For the above parameters, it is easy to observe that the value $n_{Y'}$ is always greater or equal to 21 (it is in fact 25 for DAGS_1), which keeps us clear of FOPT. With respect to the BC attack, the complexity analysis provided by the authors results in a value of $\approx 2^{126}$ for DAGS_1 and more than 2^{288} for the other two sets. Finally, the running cost of ISD (using Peters' script) is estimated at $2^{128}, 2^{192}$ and 2^{256} respectively, as desired.

6 Implementation and Performance Analysis (2.B.2)

6.1 Components

For DAGS_1, the finite field $\mathbb{F}_{2^6}$ is built using the polynomial $x^6 + x + 1$ and then extended to $\mathbb{F}_{2^{12}}$ using the quadratic irreducible polynomial $x^2 + \alpha x + \alpha$, where α is a primitive element of $\mathbb{F}_{2^6}$. In particular, using a well-known result on finite fields, we choose $\alpha = \gamma^{65}$ where γ is a primitive element of $\mathbb{F}_{2^{12}}$. This particular choice allows for more efficient arithmetic using a *conversion matrix* to switch between different field representations. Similarly, for DAGS_3 and DAGS_5, we build the base field using $x^8 + x^4 + x^3 + x^2 + 1$ and the extension field $\mathbb{F}_{2^{16}}$ is obtained via $x^2 + \beta^{50}x + \beta$, where β is a primitive element of $\mathbb{F}_{2^8}$.

The three main functions from DAGS are defined as:

Key generation: the key generation algorithm *key_gen* is composed by three main functions: *binary_quasi_dyadique_sig*, *cauchy_support* and *key_pair*. The first two first functions are in charge of generating the signature and the Cauchy matrix respectively. The *key_pair* function generates public key and private key which is stored in memory for a better performance.

Encapsulation: the encapsulation algorithm is essentially composed of the function *encapsulation* in the file *encapsulation.c*, where it computes the expansion of the message and the McEliece-like encryption. In the end, the function computes the hash function $\mathcal{K}$ to get the shared secret.

Decapsulation: the decapsulation algorithm consists mainly of the function *decapsulation* in the file *decapsulation.c*, where we essentially run the decoding algorithm plus a few comparisons. In the end, we compute the hash function $\mathcal{K}$ to get the shared secret.

6.2 Randomness Generation

The randomness used in our implementation is provided by the NIST api. It uses AES as a PRNG, where NIST chooses the seed in order to have a controlled environment for tests. We use this random generator to obtain our input message $\boldsymbol{m}$, after which we calculate $\mathcal{G}(\boldsymbol{m})$ and $\mathcal{H}(\boldsymbol{m})$, where $\mathcal{G}$ is an expansion function and $\mathcal{H}$ is a compression function. In practice, we compute both using the *KangarooTwelve* function [15] from the Keccak family. To generate a low-weight error vector, we take part of $\mathcal{G}(\boldsymbol{m})$ as a seed $\boldsymbol{\sigma}$. We use again *KangarooTwelve* to expand the seed into a string of length n, then transform the latter into a fixed-weight string using a deterministic function.

6.3 Efficient Private Key Reconstruction and Decoding

As mentioned in Section 3.1, in our scheme we use a standard alternant decoder (Step 2 of Algorithm 3.1.3), which requires the input to be a matrix in alternant form, i.e. $H'_{ij} = y_j x_j^i$ for $i = 0, \ldots, st-1$ and $j = 0, \ldots, n-1$. The first step consists of computing the syndrome of the received word, $H'\boldsymbol{c}^T$. Now, defining the whole alternant matrix H' as private key would require storing stn elements of $\mathbb{F}_{q^m}$, leading to huge key sizes. It would be possible to store as private key just the defining vectors $\boldsymbol{u}, \boldsymbol{v}$ and $\boldsymbol{z}$, and then compute the alternant form during decapsulation. Doing so would drastically reduce the private key size, but would also significantly slow down the decapsulation algorithm. Thus we implemented the following hybrid approach. We use $\boldsymbol{u}, \boldsymbol{v}$ and $\boldsymbol{z}$ to compute the vector $\boldsymbol{y}$ during key generation and store $(\boldsymbol{v}, \boldsymbol{y})$ as private key, which still results in a compact size. Then, we complete the computation of the alternant form in the decapsulation algorithm. To avoid an unnecessary overhead, we incorporate this computation together with the syndrome computation. The process is detailed as follows.

6.3.1 Alternant-Syndrome Computation

1. Input received word $\boldsymbol{c}$ to be decoded.
2. Compute the vector $\boldsymbol{s} = \mathrm{Diag}(\boldsymbol{y}) \cdot \boldsymbol{c}^T$.
3. Form intermediate matrix $\tilde{H}$. To do this:
   (a) Set first row equal to $\boldsymbol{s}$.
   (b) Obtain row i, for $i = 1, \ldots, st-1$, by multiplying the j-th element of row $i-1$ by v_j, for $j = 0, \ldots, n-1$.
4. Sum elements in each row and output resulting vector.

6.4 Time and Space Requirements

The implementation is in ANSI C, as requested for a generic, reference implementation. For the measurements we used a processor x64 Intel core i5-5300U@2.30GHz with 16GiB of RAM compiled with GCC version 6.3.020170516 without any optimization and running on Debian 9.2.

We start by considering space requirements. In Table 5 we recall the flow between two parties P_1 and P_2 in a standard Key Exchange protocol derived from a KEM.

Table 5: KEM-based Key Exchange flow

P_1		P_2
$(pk, sk) \leftarrow KEM.KeyGen$		
	$\xrightarrow{pk}$	
		$(\boldsymbol{k}, \boldsymbol{c}) \leftarrow KEM.Encaps(pk)$
	$\xleftarrow{\boldsymbol{c}}$	
$\boldsymbol{k}/\perp \leftarrow KEM.Decaps(\boldsymbol{c}, sk)$		
	Shared Key $:= \boldsymbol{k}$	

When instantiated with DAGS, the public key is given by the generator matrix G. The non-identity block M^T is $k \times (n-k) = k \times mst$ and is dyadic of order s, thus requires only $kmst/s = kmt$ elements of the base field for storage. The private key is simply the pair $(\boldsymbol{v}, \boldsymbol{y})$, consisting of $2n$ elements of $\mathbb{F}_{q^m}$. Finally, the ciphertext is

the pair $(\boldsymbol{c}, \boldsymbol{d})$, that is, a q-ary vector of length n plus 256 bits. This leads to the following measurements (in bytes).

Table 6: Memory Requirements.

Parameter Set	Public Key	Private Key	Ciphertext
DAGS_1	8112	2496	656
DAGS_3	11264	4864	1248
DAGS_5	19712	6400	1632

Table 7: Communication Bandwidth.

Message Flow	Transmitted Message	Size		
		DAGS_1	DAGS_3	DAGS_5
$\mathsf{P}_1 \rightarrow \mathsf{P}_2$	G	8112	11264	19712
$\mathsf{P}_2 \rightarrow \mathsf{P}_1$	$(\boldsymbol{c}, \boldsymbol{d})$	656	1248	1632

Note that in our reference code, which is not optimized, we currently allocate a full byte for each element of $\mathbb{F}_{2^6}$ and two bytes for each element of $\mathbb{F}_{2^{12}}$ thus effectively wasting some memory. However, we expect to be able to represent elements more efficiently, namely using three bytes to store either four elements of $\mathbb{F}_{2^6}$ or two elements of $\mathbb{F}_{2^{12}}$. The measurements in Tables 6 and 7, above, are taken with respect to the latter method. This is not a problem for DAGS_3 and DAGS_5, obviously.

We now move on to analyzing time measurements. We are using x64 architecture and our measurements use an assembly instruction to get the time counter. We do this by calling "rdtsc" before and after the instruction, which gives us the cycles used by each function. Table 8 gives the results of our measurements represented by the mean after running the code 50 times.

Note About Implementations. Our reference implementation and code have been compiled for DAGS_5. However, it is possible to adapt both to run with any set of parameters simply by calling the Makefile with the parameters: DAGS_1, DAGS_3 or DAGS_5.

Table 8: Timings.

Algorithm	Cycles		
	DAGS_1	DAGS_3	DAGS_5
Key Generation	2,540,311,986	4,320,206,006	7,371,897,084
Encapsulation	12,108,373	26,048,972	96,929,832
Decapsulation	215,710,551	463,849,016	1,150,831,538

We would like to remark that our reference implementation is designed for clarity, rather than performance. However, we found that, as a consequence of NIST's platform and language of choice, there wouldn't be many significant performance differences by presenting an optimized version of our reference code. Thus, our Optimized Implementation is the same as the Reference Implementation, and all the measurements presented in this section refer to the reference code.

Our team is currently at work to complete additional implementations that could better showcase the potential of DAGS in terms of performance. These include code prepared with x86 assembly instructions (AVX) as well as a hardware implementation (FPGA) etc. We plan to include such additional implementations in time for the second evaluation period. A hint at the effectiveness of DAGS can be had by looking at the performance of the scheme presented in [19], which also features an implementation for embedded devices. In particular, we expect DAGS to perform especially well in hardware, due to the nature of the computations of the McEliece framework.

7 Advantages and Limitations (2.B.6)

We presented DAGS, a Key Encapsulation Mechanism based on Quasi-Dyadic Generalized Srivastava codes. We proved that DAGS is IND-CCA secure in the Random Oracle Model, and in the Quantum Random Oracle Model. Thanks to this feature, it is possible to employ DAGS not only as a key-exchange protocol (for which IND-CPA would be a sufficient requirement), but also in other contexts such as Hybrid Encryption, where IND-CCA is of paramount importance.

Like any scheme based on structured algebraic codes, DAGS is susceptible to algebraic attacks (FOPT etc.); this can be seen as a limitation of the scheme. In

fact, to defeat the attacks, we need to respect stringent conditions on the minimal choices of values for the scheme, in particular the size of the fields in use (both the base field q and the extension degree m) and the values t and s. We remark that in many cases an accurate complexity analysis of the attack is not available. This forces us to choose conservative parameters, and this can also been seen as a disadvantage of the scheme.

Nevertheless, DAGS is competitive and compares well with other other code-based schemes. These include the classic McEliece approach [4], as well as more recent proposals such as BIKE [3] and BIG QUAKE [2]. The "Classic McEliece" project is an evolution of the well-known McBits [14](based on the work of Persichetti [39]), and benefits from a well-understood security assessment but suffers from the usual public key size issue. BIG QUAKE continues the line of work of [11], and proposes to use quasi-cyclic Goppa codes. Due to the particular nature of the algebraic attacks, it seems harder to provide security with this approach, and the protocol has to use very large parameters in order to do so. Finally, BIKE, a protocol based on QC-MDPC codes, is the result of a merge between two independently published works with similar background, namely CAKE [9] and Ouroboros [21]. The scheme possesses some very nice features like compact keys and an easy implementation approach, but has currently some potential drawbacks. In fact, the QC-MDPC encryption scheme on which it is based is susceptible to a reaction attack by Guo, Johansson and Stankovski (GJS) [27], and thus the protocol is forced to employ ephemeral keys. Moreover, due to its non-trivial Decoding Failure Rate (DFR), achieving IND-CCA security is currently infeasible, so that the BIKE protocol only claims to be IND-CPA secure.

Indeed, another advantage of our proposal is that it doesn't involve any decoding error. This is particularly favorable in a comparison with some lattice-based schemes like [18], [5] and [17], as well as BIKE. No decoding error allows for a simpler formulation and better security bounds in the IND-CCA security proof.

Our public key size is considerably smaller than Classic McEliece and BIG QUAKE, and similar to that of BIKE. With regards to the latter, we point out that while, for the same security level, DAGS public keys are indeed bigger, our ciphertexts are a lot smaller. This is because DAGS uses much shorter codes than BIKE, and the size of ciphertexts is a direct consequence of this fact. Thus, in the end, the total communication bandwidth is of the same order of magnitude.
All the objects involved in the scheme are vectors of finite fields elements, which in turn are represented as binary strings; thus computations are very fast. The cost

of computing the hash functions is minimized thanks to the parameter choice that makes sure the input $\boldsymbol{m}$ is only 256 bits. As a result, we expect that it will be possible to implement our scheme efficiently on multiple platforms.

Finally, we would like to highlight that a DAGS-based Key Exchange features an "asymmetric" structure, where the bandwidth cost and computational effort of the two parties are considerably different. In particular, in the flow described in Table 5, the party P_2 benefits from a much smaller message and faster computation (the encapsulation operation), whereas P_1 has to perform a key generation and a decapsulation (which includes a run of the decoding algorithm), and transmit a larger message (the public matrix). This is suitable for traditional client-server applications where the server side is usually expected to respond to a large number of requests and thus benefits from a lighter computational load. On the other hand, it is easy to imagine an instantiation, with reversed roles, which could be suitable for example in Internet-of-Things (IoT) applications, where it would be beneficial to lesser the burden on the client side, due to its typical processing, memory and energy constraints.

All in all, our scheme offers great flexibility in key exchange applications, which is not the case for traditional key exchange protocols like Diffie-Hellman.

8 Acknowledgments

Cheikh Thiecoumba Gueye, Brice Odilon Boidje, Jean Belo Klamti, Ousmane Ndiaye and Gilbert Ndollane Dione were supported by the National Commission of Cryptology via the ISPQ project and by the CEA-MITIC via the CBC project. Jefferson E. Ricardini is Supported by the joint São Paulo Research Foundation (FAPESP)/Intel Research grant 2015/50520-6 "Efficient Post-Quantum Cryptography for Building Advanced Security". Gustavo Banegas has received funding under the European Union's Horizon 2020 research and innovation program (Marie Sklodowska-Curie grant agreement 643161 ECRYPT-NET).

References

[1] http://christianepeters.wordpress.com/publications/tools/.

[2] https://bigquake.inria.fr/.

[3] https://bikesuite.org.

[4] https://classic.mceliece.org/.

[5] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange - a new hope. Cryptology ePrint Archive, Report 2015/1092, 2015. `http://eprint.iacr.org/2015/1092`.

[6] Elise Barelli and Alain Couvreur. An efficient structural attack on nist submission dags. *arXiv preprint arXiv:1805.05429*, 2018.

[7] A. Barg. Some new NP-complete coding problems. *Probl. Peredachi Inf.*, 30:23–28, 1994. (in Russian).

[8] A. Barg. Complexity issues in coding theory. *Electronic Colloquium on Computational Complexity (ECCC)*, 4(46), 1997.

[9] Paulo SLM Barreto, Shay Gueron, Tim Gueneysu, Rafael Misoczki, Edoardo Persichetti, Nicolas Sendrier, and Jean-Pierre Tillich. Cake: Code-based algorithm for key encapsulation.

[10] Paulo SLM Barreto, Richard Lindner, and Rafael Misoczki. Monoidic codes in cryptography. *PQCrypto*, 7071:179–199, 2011.

[11] T. P. Berger, P.-L. Cayrel, P. Gaborit, and A. Otmani. Reducing Key Length of the McEliece Cryptosystem. In *AFRICACRYPT*, pages 77–97, 2009.

[12] E. Berlekamp, R. McEliece, and H. van Tilborg. On the inherent intractability of certain coding problems (corresp.). *Information Theory, IEEE Transactions on*, 24(3):384 – 386, may 1978.

[13] Daniel J. Bernstein. *Grover vs. McEliece*, pages 73–80. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[14] Daniel J. Bernstein, Tung Chou, and Peter Schwabe. Mcbits: Fast constant-time code-based cryptography. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 8086 LNCS, pages 250–272, 12 2013.

[15] Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. Kangarootwelve: fast hashing based on keccak-p. *IACR Cryptology ePrint Archive*, 2016:770, 2016.

[16] B. Biswas and N. Sendrier. Mceliece cryptosystem implementation: Theory and practice. In *PQCrypto*, pages 47–62, 2008.

[17] Joppe Bos, Craig Costello, Léo Ducas, Ilya Mironov, Michael Naehrig, Valeria Nikolaenko, Ananth Raghunathan, and Douglas Stebila. Frodo: Take off the ring! practical, quantum-secure key exchange from LWE. Cryptology ePrint Archive, Report 2016/659, 2016. `http://eprint.iacr.org/2016/659`.

[18] Joppe W Bos, Craig Costello, Michael Naehrig, and Douglas Stebila. Post-quantum key exchange for the tls protocol from the ring learning with errors problem. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pages 553–570. IEEE, 2015.

[19] Pierre-Louis Cayrel, Gerhard Hoffmann, and Edoardo Persichetti. Efficient implementation of a cca2-secure variant of McEliece using generalized Srivastava codes. In *Proceedings of PKC 2012, LNCS 7293, Springer-Verlag*, pages 138–155, 2012.

[20] N. Courtois, M. Finiasz, and N. Sendrier. How to achieve a mceliece-based digital signature scheme. In *ASIACRYPT*, pages 157–174, 2001.

[21] Jean-Christophe Deneuville, Philippe Gaborit, and Gilles Zémor. Ouroboros: A simple, secure and efficient key exchange protocol based on coding theory. In Tanja Lange and Tsuyoshi Takagi, editors, *PQCrypto 2017*, volume 10346 of *LNCS*, pages 18–34. Springer, 2017.

[22] J.-C. Faugère, V. Gauthier-Umaña, A. Otmani, L. Perret, and J.-P. Tillich. A distinguisher for high rate mceliece cryptosystems. In *Information Theory Workshop (ITW), 2011 IEEE*, pages 282 –286, oct. 2011.

[23] J.-C. Faugère, A. Otmani, L. Perret, and J.-P. Tillich. Algebraic cryptanalysis of mceliece variants with compact keys. In *EUROCRYPT*, pages 279–298, 2010.

[24] J.-C. Faugère, A. Otmani, L. Perret, and J.-P. Tillich. Algebraic Cryptanalysis of McEliece Variants with Compact Keys – Towards a Complexity Analysis. In *SCC '10: Proceedings of the 2nd International Conference on Symbolic Computation and Cryptography*, pages 45–55, RHUL, June 2010.

[25] Jean-Charles Faugere, Ayoub Otmani, Ludovic Perret, Frédéric De Portzamparc, and Jean-Pierre Tillich. Structural cryptanalysis of mceliece schemes with compact keys. *Designs, Codes and Cryptography*, 79(1):87–112, 2016.

[26] E. Fujisaki and T. Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99*, volume 1666 of *LNCS*, pages 537–554. Springer, 1999.

[27] Qian Guo, Thomas Johansson, and Paul Stankovski. *A Key Recovery Attack on MDPC with CCA Security Using Decoding Errors*, pages 789–815. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016.

[28] Yann Hamdaoui and Nicolas Sendrier. A non asymptotic analysis of information set decoding. Cryptology ePrint Archive, Report 2013/162, 2013. `http://eprint.iacr.org/2013/162`.

[29] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the Fujisaki-Okamoto transformation. Cryptology ePrint Archive, Report 2017/604, 2017. `http://eprint.iacr.org/2017/604`.

[30] A. Al Jabri. *A Statistical Decoding Algorithm for General Linear Block Codes*, pages 1–8. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[31] Ghazal Kachigar and Jean-Pierre Tillich. Quantum information set decoding algorithms. In Tanja Lange and Tsuyoshi Takagi, editors, *PQCrypto 2017*, volume 10346 of *LNCS*, pages 69–89. Springer, 2017.

[32] F. J. MacWilliams and N. J. A. Sloane. *The Theory of Error-Correcting Codes*, volume 16. North-Holland Mathematical Library, 1977.

[33] R. J. McEliece. A public-key cryptosystem based on algebraic coding theory. *Deep Space Network Progress Report*, 44:114–116, January 1978.

[34] R. Misoczki and P. S. L. M. Barreto. Compact mceliece keys from goppa codes. In *Selected Areas in Cryptography*, pages 376–392, 2009.

[35] R. Niebuhr. *Statistical Decoding of Codes over* $\mathbb{F}_q$, pages 217–227. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[36] R. Niebuhr, E. Persichetti, P.-L. Cayrel, S. Bulygin, and J. Buchmann. On lower bounds for information set decoding over $\mathfrak{U}_q$ and on the effect of partial knowledge. *Int. J. Inf. Coding Theory*, 4(1):47–78, January 2017.

[37] R. Nojima, H. Imai, K. Kobara, and K. Morozov. Semantic security for the McEliece cryptosystem without random oracles. *Des. Codes Cryptography*, 49(1-3):289–305, 2008.

[38] Edoardo Persichetti. Compact mceliece keys based on quasi-dyadic srivastava codes. *Journal of Mathematical Cryptology*, 6(2):149–169, 2012.

[39] Edoardo Persichetti. Secure and anonymous hybrid encryption from coding theory. In Philippe Gaborit, editor, *Post-Quantum Cryptography: 5th International Workshop, PQCrypto 2013, Limoges, France, June 4-7, 2013. Proceedings*, pages 174–187, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[40] C. Peters. Information-set decoding for linear codes over $\mathbb{F}_q$. In *PQCrypto*, LNCS, pages 81–94, 2010.

[41] E. Prange. The use of information sets in decoding cyclic codes. *IRE Transactions*, IT-8:S5–S9, 1962.

[42] F. Strenzke. A timing attack against the secret permutation in the mceliece pkc. In *PQCrypto*, pages 95–107, 2010.

[43] F. Strenzke, E. Tews, H. G. Molter, R. Overbeck, and A. Shoufan. Side channels in the mceliece pkc. In *PQCrypto*, pages 216–229, 2008.

A Note on the Choice of ω

As discussed in Section 6.3, in our scheme we use a standard alternant decoder. After computing the syndrome of the word to be decoded, the next step is to recover the error locator polynomial $\sigma(x)$, by means of the euclidean algorithm for polynomial division; the algorithm then proceeds by finding the roots of σ. There is a 1-1 correspondence between these roots and the error positions: in fact, there is an error in position i if and only if $\sigma(1/x_i) = 0$. Of course, if one of the x_i's is equal to 0, it is not possible to find the root, and to detect the error.

Now, the generation of the error vector is random, hence we can assume the probability of having an error in position i to be around $st/2n$; since the codes give the best performance when mst is close to $n/2$, we can estimate this probability as $1/4m$, which is reasonably low for any nontrivial choice of m; however, we still argue that the code is not fully decodable and we now explain how to adapt the key generation algorithm to ensure that all the x_i's are nonzero.

As part of the key generation algorithm we assign to each x_i the value v_i, hence it is enough to restrict the possible choices for ω to the set $\{\alpha \in \mathbb{F}_{q^m} | \alpha \neq 1/h_i + 1/h_0, i = 0, \ldots, n-1\}$. In doing so, we considerably restrict the possible choices for ω but we ensure that the decoding algorithm works properly.