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 [8], which shows to be strong against quantum attacks. Over the years, since McEliece's seminal work [27], 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. [7], who proposed Quasi-Cyclic (QC) codes, and Misoczki and Barreto [28], suggesting Quasi-Dyadic (QD) codes instead (later generalized to Quasi-Monoidic (QM) codes [6]). 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 [17] 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 [32].

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. [23], 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
$\mathbf{a}$	a vector
A	a matrix
$\mathcal{A}$	an algorithm or hash function
A	a set
$\mathrm{Diag}(\mathbf{a})$	the diagonal matrix formed by the vector $\mathbf{a}$
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 [26, 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 set to be "small". In practice, k' is such that a vector of length k' can be efficiently stored in 256 bits, depending on the base

field. This makes the hash functions (see below) easy to compute, and minimizes the overhead due to the IND-CCA2 security in the QROM.

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 $\mathbf{h} = (h_0, \ldots, h_{n-1})$ 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 $C(\mathbf{u}, \mathbf{v})$ 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 $\mathbf{h}$. 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. Set $u_i = 1/h_i + \omega$ and $v_j = 1/h_j + 1/h_0 + \omega$ for $i = 0, \ldots, s-1$ and $j = 0, \ldots, n-1$.
   iii. Set $\mathbf{u} = (u_0, \ldots, u_{s-1})$ and $\mathbf{v} = (v_0, \ldots, v_{n-1})$.
3. Form Cauchy matrix $\hat{H}_1 = C(\mathbf{u}, \mathbf{v})$.
4. Build blocks $\hat{H}_i$, $i = 2, \ldots t$, by raising each element of $\hat{H}_1$ to the power of i.
5. Superimpose blocks to form matrix $\hat{H}$.
6. Choose random elements $z_i \xleftarrow{\$} \mathbb{F}_{q^m}$ such that $z_{is+j} = z_{is}$ 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. Form $H = \hat{H} \cdot \mathrm{Diag}(\mathbf{z})$.
8. Transform H into alternant form[3]: call this H'.
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 alternant matrix H'.

The encapsulation and decapsulation algorithms make use of two hash functions[4] $\mathcal{G} : \mathbb{F}_q^{k'} \to \mathbb{F}_q^k$ 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 [23]. 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 $\mathbf{m} \xleftarrow{\$} \mathbb{F}_q^{k'}$.
2. Compute $\mathbf{r} = \mathcal{G}(\mathbf{m})$ and $\mathbf{d} = \mathcal{H}(\mathbf{m})$.
3. Parse $\mathbf{r}$ as $(\boldsymbol{\rho} \parallel \boldsymbol{\sigma})$ then set $\boldsymbol{\mu} = (\boldsymbol{\rho} \parallel \mathbf{m})$.
4. Generate error vector $\mathbf{e}$ of length n and weight w from $\boldsymbol{\sigma}$.
5. Compute $\mathbf{c} = \boldsymbol{\mu} G + \mathbf{e}$.
6. Compute $\mathbf{k} = \mathcal{K}(\mathbf{m})$.
7. Output ciphertext $(\mathbf{c}, \mathbf{d})$; the encapsulated key is $\mathbf{k}$.

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 [26, Ch. 12, §9] and requires the parity-check matrix to be in alternant form (hence the nature of the private key).

[3] See §2 and §6 of [26, Ch. 12].
[4] As specified in Section 2.1

3.1.3 Decapsulation

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

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. [31] into our scheme. This is extremely beneficial for two distinct aspects: first of all, it allows us to use a much shorter vector $\mathbf{m}$ to generate the remaining components of the scheme, greatly improving efficiency. Secondly, it allows us to get tighter security bounds. Note that our protocol differs slightly from the paradigm presented in [23], 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 $\mathbf{e}$ (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 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 [23], but note that in fact this corresponds to the notion of γ-*uniformity* given by Fujisaki and Okamoto in [20], except for a change of constants. In other words, a scheme is γ-spread if it is $2^{-\gamma}$-uniform.

It was proved in [14] 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 $\mathbf{y}$ be a generic vector of $\mathbb{F}_q^n$. Then either $\mathbf{y}$ is a word at distance w from the code, or it isn't. If it isn't, the probability of $\mathbf{y}$ being a valid ciphertext is clearly exactly 0. On the other hand, suppose $\mathbf{y}$ is at distance w from the code; then there is only one choice of $\boldsymbol{\rho}$ and one choice of $\mathbf{e}$ that satisfy the equation (since w is below the GV bound), i.e. the probability of $\mathbf{y}$ 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*

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

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

Proof The thesis is a consequence of the results presented in Section 3.3 of [23]. 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 [23]), which greatly simplifies the resulting bound. □

The value $\mathbf{d}$ 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 [23]. 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. □

[6] Same as above.

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 and a vector $\mathbf{y}$, both defined over $\mathbb{F}_q$, and a non-negative integer w, find a vector $\mathbf{e} \in \mathbb{F}_q^n$ of weight w such that $H\mathbf{e}^T = \mathbf{y}$.*

The corresponding decision problem was proved to be NP-complete in 1978 [8], but only for binary codes. In 1994, A. Barg proved that this result holds for codes over all finite fields ([3], in Russian, and [4, 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. [16]) for cryptographic protocols that make use of high-rate Goppa codes (like the CFS signature scheme [15]). 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. ([36, 37]). Thus, traditionally, the safest method (Biswas and Sendrier, [11]) 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 [35]. Among several variants and generalizations, Peters showed [34] that it is possible to apply Prange's approach to

generic q-ary codes. Other approaches such as Statistical Decoding [24, 29] are usually considered less efficient. Thus, when choosing parameters, we will focus mainly on defeating attacks of the ISD family.

Hamdaoui and Sendrier in [22] provide non-asymptotic complexity estimates for ISD in the binary case. For codes over $\mathbb{F}_q$, instead, a bound is given in [30], 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 [9] shows that Grover's algorithm applies to ISD-like algorithms, effectively halving the asymptotic exponent in the complexity estimates. Later, it was proven in [25] 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 FOPT

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 [17]. The attack (for convenience referred to as FOPT) relies on the simple property (valid for every linear code) $H \cdot G^T = 0$ to build an algebraic system, using then Gröbner bases techniques to solve it. The special properties of alternant codes are fundamental, as they contribute to considerably reduce the number of unknowns of the system.

The attack was originally aimed at two variants of McEliece, introduced respectively in [7] and [28]. The first variant, using quasi-cyclic codes, was completely broken, and falls out of the scope of this paper. The second variant, instead, only considered quasi-dyadic Goppa codes. In this case, most of the parameters proposed have also been broken very easily, 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 the attack, as it defines the dimension of the

solution space, which is equal, in fact, exactly to $m-1$. In a successive paper [18], the authors provide a theoretical complexity bound for the attack, and 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. In the case of GS codes, though, there is one important difference to keep in mind. In fact, as shown in [32], the dimension of the solution space is defined by $mt-1$, rather than $m-1$ as for Goppa codes. This provides greater flexibility when designing parameters for the code, and it allows, for example, to keep the extension degree m small.

Recently, an extension of the FOPT attack appeared in [19]. 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 much 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. The paper features two tables with several sets of parameters, respectively for signature schemes, and encryption schemes. The parameters are either taken from the original papers, or generated ad hoc. 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. Despite a refinement in the techniques used to solve the algebraic system, some of the parameters could not be solved in practice, and even the binary Goppa codes of [28], with their relatively low dimension of 15, require a considerably high computational effort (at least 2^{150} operations).

It is not clear how the attack performs against GS codes, since the authors didn't present any explicit result against this particular family of codes, nor attempted to decode GS codes specifically. Thus, an attack against GS codes would use generic techniques for Alternant codes, and wouldn't benefit from the speedups which are specific to (binary) Goppa codes. Furthermore, the authors do not propose a concrete bound, but only provide experimental results. For these reasons, and until an accurate complexity analysis for an attack on GS codes is available, we choose to attain to the latest measurable guidelines (those suggested in [18]) and choose our parameters such that the dimension of the solution space for the algebraic system is strictly greater than 20.

5.4 Parameter Selection

To choose our parameters, we have to first keep in mind all of the remarks from the previous sections about decoding and structural attacks. For FOPT, we have the condition $mt \geq 21$. This guarantees at least 128 bits of security according to the bound presented in [18]. On the other hand, 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 an optimal choice in this sense seems to be $q = 2^6, m = 2$. 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.

We also included the estimated complexity of the structural attack (column FOPT), which is at least greater than 128 bits in all cases.

Table 1: Suggested DAGS Parameters.

Security Level	FOPT	q	m	n	k	k'	s	t	w
1	≥ 128	2^5	2	832	416	32	2^4	13	104
3	≥ 128	2^6	2	1216	512	32	2^5	11	176
5	≥ 128	2^6	2	2112	704	32	2^6	11	352

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

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

6.1 Components

DAGS computations are detailed as follows:

Key generation:

1. Two polynomial multiplications in $\mathbb{F}_{q^m}$ and two in $\mathbb{F}_q$.
2. Six polynomial inversions in $\mathbb{F}_{q^m}$ and four in $\mathbb{F}_q$.
3. Two polynomial squarings in $\mathbb{F}_{q^m}$ and two in $\mathbb{F}_q$.
4. Two polynomial additions in $\mathbb{F}_{q^m}$ and two in $\mathbb{F}_q$.
5. One random generation of a polynomial in $\mathbb{F}_{q^m}$.

Encapsulation:

1. One polynomial multiplication in $\mathbb{F}_q$.
2. One polynomial addition in $\mathbb{F}_q$.
3. One random generation of a polynomial in $\mathbb{F}_q$.
4. One hash function computation.

Decapsulation:

1. Three polynomial multiplications in $\mathbb{F}_{q^m}$.
2. One polynomial power in $\mathbb{F}_{q^m}$.
3. One polynomial addition in $\mathbb{F}_{q^m}$.
4. One random generation of a polynomial in $\mathbb{F}_q$.
5. One hash function computation.

For DAGS_3 and DAGS_5, 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 $x^2+x+\alpha^{34}$, where α is a primitive element of $\mathbb{F}_{2^6}$. For DAGS_1, we build $\mathbb{F}_{2^5}$ using x^5+x^2+1 and then extend it to $\mathbb{F}_{2^{10}}$ via $x^2+\alpha^4 x+\alpha$.

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 Time and Space Requirements

The implementation is in ANSI C. 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 2 we recall the flow between two parties P_1 and P_2 in a standard Key Exchange protocol derived from a KEM.

Table 2: KEM-based Key Exchange flow

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

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 given by the alternant matrix H' which is composed of stn elements of $\mathbb{F}_{q^m}$. Finally, the ciphertext C is the pair $(\mathbf{c}, \mathbf{d})$, that is, a q-ary vector of length n plus 256 bits. This leads to the following measurements (in bytes).

Table 3: Memory Requirements.

Parameter Set	Public Key	Private Key	Ciphertext
DAGS_1	6760	432640	552
DAGS_3	8448	1284096	944
DAGS_5	11616	2230272	1616

Table 4: Communication Bandwidth.

Message Flow	Transmitted Message	Size		
		DAGS_1	DAGS_3	DAGS_5
$\mathsf{P}_1 \rightarrow \mathsf{P}_2$	G	6760	8448	11616
$\mathsf{P}_2 \rightarrow \mathsf{P}_1$	$(\mathbf{c}, \mathbf{d})$	552	944	1616

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 3 and 4, above, are taken with respect to the latter method.

Furthermore, we would like to mention that the representation of the private key offers a significant tradeoff between time and space. In fact, it would be possible to store as private key the (quasi-dyadic) matrix H or even the defining vectors $\mathbf{u}, \mathbf{v}$ and $\mathbf{z}$, and then compute the alternant form during decapsulation (following [26, Ch. 12, §2,6]); this, however, would significantly slow down the decapsulation algorithm. Thus, we have opted to store H' instead and save computation time, although this obviously results in a very large private key. It is debatable which of the two routes is preferable, and we signal this as an implementor's choice.

We now move on to analyze 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 5 gives the results of our measurements represented by the mean after running the code 50 times.

Table 5: Timings.

Algorithm	Cycles		
	DAGS_1	DAGS_3	DAGS_5
Key Generation	49394032811	106876216775	136497712522
Encapsulation	20109354	26109354	49029613
Decapsulation	23639371	24639371	260829051

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 editing the file *param.h*.

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 [14], 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 the FOPT attack and its successive improvements; this can be seen as a limitation of the scheme. In fact, to defeat the attack, we need to respect stringent conditions on the minimal choices of values for the scheme, in particular the extension degree m and the value t. We remark that an accurate complexity analysis of the attack is, to date, not available, and a version of the attack targeting GS codes hasn't yet been provided. This forces us to choose conservative parameters, according to the theoretical bound of [18].

Nevertheless, DAGS is competitive and compares well with other other code-based schemes. These include the well-known McBits [10], as well as more recent proposals such as CAKE [5]. The former follows the work of Persichetti [33], and is built using binary Goppa codes, thus benefiting from a well-understood security assessment. The scheme however suffers from the same public key size issue as "classic" McEliece-like cryptosystems. On the other hand, CAKE, a protocol based on QC-MDPC codes, possesses some very nice features like compact keys and an easy implementation approach, but the QC-MDPC encryption scheme on which it is based suffers from a security-related drawback. This means that, in order to circumvent the Guo-Johansson-Stankovski (GJS) attack [21], the protocol is forced to employ ephemeral keys. While CAKE key generation is indeed very fast, this still causes an increase in computation time. Moreover, due to its non-trivial Decoding Failure Rate (DFR), achieving IND-CCA security becomes very hard, so that the CAKE 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 [13], [2] and [12], as well as CAKE. No decoding error allows for a simpler formulation and better security bounds in the IND-CCA security proof.

Our public key is much smaller than the McBits family, and of the same order of magnitude of CAKE. With respect to CAKE, it is possible to notice that, for the

same security level, DAGS requires lower overall communication bandwidth. This is because, while the size of a CAKE public key is slightly less than a DAGS key, DAGS uses much shorter codes, and as a consequence the ciphertext is quite small compared to a CAKE ciphertext. 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 $\mathbf{m}$ 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 2, 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 (the most expensive operations in the scheme), 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] 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`.

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

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

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

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

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

[8] 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.

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

[10] 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.

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

[12] 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`.

[13] 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.

[14] 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.

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

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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`.

[23] 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.

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

[25] 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.

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

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

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

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

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

[31] 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.

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

[33] 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.

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

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

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

[37] 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

In this section we point out some considerations about the choice of the offset ω during the key generation process.

The usual decoding algorithm for alternant codes, for example as in [26], relies on the special form of the parity-check matrix ($H_{ij} = y_j x_j^{i-1}$). The first step is to recover the error locator polynomial $\sigma(x)$, by means of the euclidean algorithm for polynomial division; then it 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 L_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.