August 9, 2016

MASTER THESIS

ACHIEVING 128-BIT SECURITY AGAINST QUANTUM ATTACKS IN OPENVPN

Simon de Vries

Faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS)
Services, Cybersecurity and Safety (SCS)

Graduation committee:
dr. A. Peter
dr. M.H. Everts
N. Duif, MSc (Ministerie BZK)

UNIVERSITY OF TWENTE.

Achieving 128-bit Security against Quantum Attacks in OpenVPN

Simon de Vries

Abstract—Niederreiter is a candidate post-quantum cryptosystem. Its large public key size currently discourages its use in practice. We demonstrate and evaluate how Niederreiter can be used for quantum-secure key exchanges by implementing it in OpenVPN. We contribute an analysis of how much Grover's algorithm can speed up existing attacks on Niederreiter and McEliece and what code parameters can protect against these attacks. We provide parameters for 128-bit quantum security that result in almost 35% smaller keys than parameters currently available in literature.

Index Terms—OpenVPN, post-quantum cryptography, McEliece, Niederreiter, key exchange

1 Introduction

OpenVPN is a software application which can be used to set up a *Virtual Private Network* (VPN) [1]. It allows connecting two private networks over a public untrusted network, creating a secure channel between the private networks. In order to provide this secure link, TLS is used for key exchange.

1.1 A new threat: quantum computers

The most used algorithms for key exchange in TLS are RSA, Diffie-Hellman and Elliptic Curve Diffie-Hellman. These algorithms respectively rely on the integer factorization problem, the Diffie-Hellman problem, and the elliptic curve Diffie-Hellman problem. No classical algorithms to solve these problems in polynomial time are known. However, these problems can be solved in polynomial time on a quantum computer, using Shor's algorithm [2]. Almost all asymmetric cryptography in use today can be efficiently broken by quantum algorithms.

Symmetric cryptography, on the other hand, seems to be able to survive quantum computer attacks. The best quantum attacks on symmetric ciphers and hash functions currently known use Grover's algorithm [2]. In order to find a 256-bit symmetric key from a number of plaintexts and ciphertexts, or to find a pre-image for a 256-bit hash function, Grover's algorithm needs approximately 2^{128} iterations. In practice there can additionally be significant overhead from quantum error correction [3], [4].

Although powerful quantum computers do not exist yet, it is expected to be only a matter of time before they can be used to break current key exchange algorithms [2]. In February 2016, NIST published a draft report on Post-Quantum Cryptography which states that "many scientists now believe it [building a large quantum computer] to be merely a significant engineering challenge", although still substantial long-term efforts are needed to actually build one [5]. NIST is reluctant to provide concrete estimates of when scalable quantum computers will be available, although they do include an estimation from Matteo Mariantoni, a scientist who has been working on quantum computer research. He estimates that "it is likely that a quantum computer capable of breaking RSA-2048 in a matter of hours could be built by 2030 for a budget of about a billion dollars" [5], [6]. Quantum computers have already successfully factorized small integers [7], [8]. Worryingly, an attacker can store intercepted key exchanges and ciphertexts today and decrypt them when a large-scale quantum computer will be available. Depending on when (and if) powerful quantum computers will become available, this may make current asymmetric cryptography unsuitable for encryption of long-term secrets.

1.2 A new defense: post-quantum cryptography

A number of defenses against quantum computer attacks are known. A possible solution is to use quantum key distribution, which uses quantum communication to exchange a key between two parties and can be mathematically proven secure assuming some physical laws hold. A significant disadvantage of this solution is that it is incompatible with current networking hardware and therefore cannot be used across the Internet. Another solution is to use classical cryptosystems that are not known to be vulnerable to quantum computer attacks, for example by using only symmetric cryptography. However, this would significantly change the OpenVPN protocol and forward secrecy will be lost. A third and most promising solution is to replace vulnerable asymmetric cryptography in OpenVPN by cryptosystems that are still considered secure in a quantum-world. These cryptosystems are part of *post-quantum cryptography*.

1.3 Our contribution

We will investigate how post-quantum cryptography can be used to achieve 128-bit quantum security in OpenVPN. In a quantum world, OpenVPN's current Diffie-Hellman key exchange will be broken. Because quantum computers capable of breaking current asymmetric cryptography do not exist yet, they cannot be used to break authenticity of key exchanges protected by digital signatures today. Therefore in this paper we focus on the confidentiality of the key exchange.

We will implement the McEliece cryptosystem in OpenVPN, extending the existing key exchange by one secure against quantum attackers. We use McEliece because it is the oldest asymmetric post-quantum primitive and is one of the most trusted candidates for post-quantum cryptography. It was published in 1978 by Robert McEliece [9]. No efficient quantum attack on McEliece is known [10]. The best known attack uses Grover's algorithm to speed up existing information-set decoding attacks [11]. McEliece's main disadvantage is its large public key size, which is why it is currently not being used in practice. At time of writing, only very few implementations of McEliece are publicly available.

We provide two complementary measures to mitigate the challenge of McEliece's large public keys. Firstly, we analyse how much Grover's algorithm can speed up existing attacks on McEliece and what parameters of McEliece can protect against these attacks. We provide parameters for 128-bit quantum security that result in almost 35% smaller keys compared to parameters currently suggested in literature. Secondly, we introduce a mechanism to minimize OpenVPN's handshake time when using the large McEliece keys. We demonstrate and evaluate the usability of our solution in practice by benchmarking it against regular OpenVPN.

Our main contributions are new McEliece parameters that result in almost 35% smaller keys for 128-bit quantum security and an implementation of McEliece in OpenVPN which is publicly available and usable in practice. For easy verification and for further research, we make our source files and computations publicly available in the supplementary material provided along with this paper.

1.4 Outline

This paper is organized as follows. Section 2 provides background information on the McEliece and Niederreiter cryptosystems. We give definitions of the key generation, encryption and decryption methods, review what codes are suitable for McEliece and give an overview of existing attacks against McEliece. In Section 3, we analyse the impact of Grover's algorithm on existing attacks against McEliece in order to optimize parameters for 128-bit quantum security. In Section 4, we explain how McEliece can be implemented in OpenVPN and how a key caching mechanism can be used to minimize OpenVPN handshake time. We discuss security guarantees and robustness of the key caching mechanism. In Section 5, we evaluate our solution by benchmarking the quantum-secure OpenVPN against regular OpenVPN. Finally, in Section 6 we provide conclusions and recommendations.

2 Review of McEliece and Niederreiter

McEliece is a post-quantum public-key cryptosystem published in 1978 by Robert McEliece [9]. The security of McEliece is based on the problem of decoding linear codes. If the code is indistinguishable from a random linear code, this is an NP-hard problem [12]. Even though this does not imply hardness for the average case or even for specific codes (many specific codes have been broken by structural attacks), it still gives confidence that this cryptosystem will not be broken by generic attacks. McEliece is one of the oldest public-key cryptosystems and has fast encryption and decryption functions.

The Niederreiter cryptosystem is a variant of the McEliece cryptosystem published in 1986 [13]. From a security point of view, the Niederreiter variant is equivalent to the original McEliece cryptosystem [14]. It is called the *dual variant* of McEliece because it uses the parity check matrix instead of the generator matrix, and ciphertexts consist of syndromes as opposed to codewords to which an error vector has been added. The advantage of this variant is that the public key is slightly smaller. For a binary linear code of length n, rank k and minimum distance d, the public key matrix has dimensions $(n-k) \times n$ instead of $k \times n$ for McEliece. For parameters suggested by the EU PQCRYPTO project [15] ($n = 6960, k = 5413$), this results in a public key of size 1.3 MiB for Niederreiter and of size 4.6 MiB for McEliece. Transforming a key into standard form, where the leftmost part of the public key equals the identity matrix, reduces the effective key size to $k(n-k)$ or 1.0 MiB for the parameters mentioned above. Using Niederreiter for encryption and decryption is slightly slower than McEliece because messages have to be encoded into error vectors. We briefly describe the Niederreiter cryptosystem by giving a definition of the KeyGen, Encrypt and Decrypt procedures:

Key generation

1) Given security parameters n, k, d, randomly select a linear code $\mathcal{C}$ of length n, rank k and minimum distance d. We will call such a code an $[n, k, d]$ *code*. An efficient decoding algorithm for $\mathcal{C}$ has to be known.
2) Generate an $(n-k) \times n$ parity check matrix H.
3) Choose a random $(n-k) \times (n-k)$ invertible binary matrix S and a random $n \times n$ permutation matrix P. The public key consists of the product $H_{pub} = SHP$, along with the number of correctable errors $t = \lfloor \frac{d-1}{2} \rfloor$. The private key consists of (S, P) and the decoding algorithm of $\mathcal{C}$.

Remark *(Choosing S and P).* Instead of choosing S randomly it can be selected such that SHP is in standard form. Also, P can be set to the identity matrix, because when the cryptosystem is based on a binary Goppa code selecting a random permutation matrix is equivalent to using a uniformly random support vector. Permuting the elements of the support vector gives the same result as permuting the columns of the parity check matrix of a Goppa code. *We will assume this choice of S and P in the following.*

Encryption

1) Given a public key $H_{pub} = SHP$ and the number of correctable errors t, and a message encoded as an error vector $e \in \mathbb{F}_2^n$ of weight t, compute the syndrome of e:

$$c = H_{pub} e^T.$$

Decryption

1) Given a ciphertext c and a Niederreiter private key, undo the multiplication by S:
$$S^{-1}c = HPe^T.$$
2) Use the syndrome decoding algorithm to decode HPe^T to Pe^T.
3) Invert the permutation of the decoded error vector Pe^T to obtain the original error vector $P^{-1}Pe^T = e^T$.
4) Typically, e will be decoded to the original message m.

The Niederreiter cryptosystem is especially suitable for key exchange, since it is easy to generate a random error vector of a given weight (as opposed to encoding a message into an error vector of given weight). When using Niederreiter for key exchange, during encryption an error vector is randomly generated. By applying a key-derivation function, a shared secret can be established from the error vector. We will do so in our implementation in OpenVPN. This approach was introduced by Shoup in 2001 [16], proven secure for use with Niederreiter by Persichetti in 2012 [17] and previously used in McBits [18].

2.1 Codes suitable for McEliece

Robert McEliece originally proposed to use *binary Goppa codes* for the McEliece cryptosystem [9]. In an attempt to reduce McEliece's key size, researchers have been looking for other linear codes. Although selecting a different code often significantly reduces the public key size, the resulting cryptosystem is often much easier to break as well. Table 1 lists various attempts of using different codes or subclasses of binary Goppa codes, showing the key size reduction and whether is was broken. Most of these codes were broken by structural decoding [19]. A lot of research has been done to breaking McEliece with binary Goppa codes yielding new attacks [20], [21]. This has resulted in a revision of secure parameters [21], but with these parameters the scheme is secure against all known attacks. There are two other unbroken proposals for codes that can be used with McEliece [22], [23]. However, these proposals are so recent that more research should be done to gain confidence in the cryptosystem's security. We choose to use the original binary Goppa code because it is unbroken for almost 40 years.

2.2 Attacks on Niederreiter

The security of the Niederreiter cryptosystem depends on the following two security assumptions: [1]

1) The selected linear code is *indistinguishable* from a random linear code, and
2) Decoding a random linear code is hard.

As all attacks need to break at least one of these assumptions, two types of attacks on Niederreiter can be distinguished: *structural* (or general) attacks and *decoding* attacks. Structural attacks aim to break the first assumption by discovering the original structure of the scrambled public key, in order to obtain the unscrambled private key. The feasibility of structural attacks mainly depends on the type of code used and the code parameters. On the other hand, decoding attacks aim to decode a codeword, using only the public key. If the first assumption holds, the feasibility of these attacks depends only on the length, rank and minimum distance of the code. The decoding problem is an NP-hard problem [12] and the best general decoding algorithms have their complexity exponential in n. Although the distinguishability problem for binary Goppa codes has not yet been reduced to a hard problem, the only efficient distinguisher known is a distinguisher for high-rate Goppa codes [38].

One of the main reasons Niederreiter is studied is because there are no efficient quantum computer attacks known [10]. The only known attack is a general attack using Grover's algorithm to improve a classical information-set decoding (ISD) attack on McEliece [11]. This has yielded an attack with a complexity of $c^{(1/2+o(1))n/\log n}$, where $c = 1/(1-\frac{k}{n})^{1-\frac{k}{n}}$. This attack, however, only uses a 'basic' information-set decoding attack as described by McEliece [9] and originally introduced in 1962 by Prange [39]. A number of more advanced classical information-set decoding attacks exist and can be found in [21], [40], [41], [42], [43], [44], [45]. These attacks have resulted in a revision of McEliece's parameters [21]. The best of these attacks has a classical complexity of $2^{0.0494n}$ [43]. The advanced forms of information set decoding attacks decrease the number of iterations at the cost of increasing complexity per iteration. An overview of these attacks is given in Table 3. In the next sections, we will analyse the complexity of attacks when the iterative search step is replaced by Grover's algorithm. Apart from speeding up existing attacks on McEliece using Grover's algorithm, no quantum attacks against McEliece are currently known.

Grover's algorithm is a quantum computer algorithm published in 1996 [46]. It was introduced as an algorithm for efficiently searching a database: given an unsorted list of n items, Grover's algorithm can find the index of a given item in $\frac{\pi}{4}\sqrt{n}$ steps with high probability using $\log_2(n)$ qubits. The best classical algorithm needs $\frac{1}{2}n$ steps on average. Although the description of Grover's algorithm as a way of searching a database suggests all items need to be kept in memory, this is not the case. In fact, the algorithm only needs access to a black box function that outputs a value of 0 for the matching item and 1 for all other inputs. This leads to the following alternative description of Grover's algorithm as a 'quantum root-finding circuit' (as stated in [11]). Given any black box function f which is implemented in a quantum circuit:
$$f : \{0,1\}^b \to \{0,1\} \tag{1}$$
Grover's algorithm finds (with high probability) a b-bit string x such that $f(x) = 0$, or determines that such a string x does not exist. Note that in this case the search space is a set containing $n = 2^b$ bit strings. Even though f needs to be implemented as a quantum circuit, any classical algorithm can be implemented as a reversible quantum circuit. This can be done by first converting the classical

1. See [37], section 2.4, for a more precise definition of these assumptions

TABLE 1
Overview of different codes used for McEliece.

Type of code	Proposed by	Current status
Binary Goppa codes	R. McEliece, 1978 [9]	Unbroken as of 2016
Generalized Reed-Solomon (GRS) codes	H. Niederreiter, 1986 [13]	Broken in 1992 [24]
Maximum rank distance (MRD) codes	E. Gabidulin *et al.*, 1991 [25] and 1993 [26]	Broken in 1994 [27] and 1996 [28]
Reed-Muller codes	V.M. Sidelnikov, 1994 [29]	Broken in 2007 [30]
Quasi-cyclic subcodes of a primitive BCH code	P. Gaborit, 2005 [31]	Broken in 2008 [32]
Quasi-cyclic low density parity-check codes	M. Baldi *et al.*, 2007 [33]	Broken in 2008 [32]
Wild McEliece	D.J. Bernstein *et al.*, 2010 [19]	Specific instances broken in 2014 [34], [35]
Wild McEliece Incognito	D.J. Bernstein *et al.*, 2011 [36]	Specific instances broken in 2014 [35]
Quasi-cyclic moderate density parity-check codes	R. Misoczki *et al.*, 2013 [22]	Unbroken as of 2016
Random linear codes	Y. Wang, 2016 [23]	Unbroken as of 2016

algorithm to an algorithm containing only NAND-gates (which is always possible because NAND is *functionally complete*), and then converting this circuit to a quantum circuit built from Toffoli gates, as described in [11]. For example, f can be implemented as a function that returns 0 if and only if the input is an AES-key for a certain plaintext-ciphertext combination [3], or a SHA-256 pre-image for a certain hash [4].

Grover's algorithm has two limitations. One limitation is that although Grover's algorithm can be parallelized, this does not result in a linear speedup. Given m quantum computers searching in a set of n items, the time complexity is $\frac{\pi}{4}\sqrt{\frac{n}{m}}$ [47]. A straightforward way of achieving this speedup is by splitting the search space of n items into m sets of $\frac{n}{m}$ items, and having each quantum computer search in one of these sets. The other limitation is that Grover's algorithm cannot be applied iteratively, meaning that the black box function f cannot contain an instance of Grover's algorithm.

If there are k matching items in a set of n (as opposed to only one matching item), Grover can find an arbitrary matching item in $\frac{\pi}{4}\sqrt{\frac{n}{k}}$ steps. This optimization can even be used if the number of matching items is not known ahead of time [48].

Grover's algorithm is optimal in at least three ways. Firstly, there does not exist a quantum algorithm that can solve the search problem faster than $\frac{\pi}{4}\sqrt{n}$ [49]. Secondly, when Grover's algorithm is stopped after some number of oracle queries, the probability that it has found the correct result is maximal [47]. Thirdly, the time complexity of finding any solution when there are multiple matching items is optimal [48].

For symmetric ciphers, an attack using Grover's algorithm reduces the cipher's security to at most half of the intended number of bits. For example, it takes about 2^{128} Grover iterations to find a 256-bit AES-key. This means that for 128-bit quantum security, setting the key size or hash output length to 256 bits is sufficient. Similarly, for code-based cryptosystems selecting system parameters (n, k, d) which provide 256-bit security against classical attacks is sufficient to provide 128-bit security against quantum attacks. However, Grover only reduces the number of iterations but does not reduce the cost per iteration. In order to optimize parameters, we should therefore analyse the impact of Grover's algorithm on attacks on McEliece more carefully. We will do so in the following section.

3 Selecting parameters for Niederreiter

All information-set decoding algorithms listed in Table 3 are probabilistic: they consist of an algorithm $\mathcal{A}$ that, with some probability p, decodes a codeword with errors (or finds a codeword of small weight). This algorithm is then iterated until the attack is successful, which is expected to happen after $\frac{1}{p}$ iterations. When adapting a classical information-set decoding algorithm to a quantum information-set decoding algorithm with Grover, we set the function f (from Equation 1) to a function that returns 0 when the algorithm $\mathcal{A}$ was successful given the input x of f, and 1 otherwise. Now the attack will be successful with high probability after only $\frac{\pi}{4}\sqrt{\frac{1}{p}}$ iterations.

This transformation has a small cost: firstly, Grover's algorithm supplies a string of qubits x of certain length to the function f, whereas information-set decoding algorithms typically have input values of different forms (for example a vector of certain weight). The bitstring x needs to be transformed to a suitable input for $\mathcal{A}$. Also, since $\mathcal{A}$ needs to be implemented as a *reversible quantum circuit* [3], [11], it is no longer possible to skip an iteration if a submatrix from the parity check matrix is not invertible. To determine the the minimum number of *qubit* operations, for each decoding algorithm we need to determine the following quantities:

- p_{inv}; the probability that a random binary matrix is invertible. As the matrix size increases, this probability quickly converges to $p_{inv} \approx 0.29$.
- $p_{success}$; the probability that one iteration of the classical decoding algorithm is successful, given that the selected columns of the matrix are invertible.
- c_{decode}; the cost in qubit operations of decoding the input qubits to the inputs of the classical algorithm.
- c_{inv}; the cost in bit operations of inverting the selected columns of the matrix.
- c_{it}; the number of bit operations operations needed to perform one iteration of the classical decoding algorithm, without the cost of inverting the submatrix.

If we are given these quantities, it can be seen that the minimum number of qubit operations, or binary workfactor WF, for each quantum decoding algorithm can be computed as follows:

$$\mathrm{WF} = \frac{\pi}{4}\sqrt{\frac{1}{p_{inv}\cdot p_{success}}}\left(\mathrm{c}_{decode} + \mathrm{c}_{inv} + \mathrm{c}_{it}\right). \quad (2)$$

The factor $\left(\frac{\pi}{4}\sqrt{\frac{1}{p_{inv}\cdot p_{success}}}\right)$ gives the expected number of iterations each algorithm needs to decode a codeword. Each iteration consists of 1) decoding the iteration's parameters from the input qubits, 2) inverting the selected columns, and 3) the cost to finish the iteration. Therefore the sum $(\mathrm{c}_{decode} + \mathrm{c}_{inv} + \mathrm{c}_{it})$ gives the cost of each iteration. We will explain how we obtain the value of $p_{success}$, c_{decode}, c_{inv} and c_{it} in the next subsection.

3.1 Non-asymptotic complexity of quantum attacks

The probability $p_{success}$ and cost c_{it} are equal to the classical iteration success probability (given that the selected columns are invertible) and classical iteration cost, respectively. These are already known; for 'basic ISD', we extract them from [11], for Stern's algorithm, we use the values mentioned in Stern's paper [41], and for MMT and BJMM we obtain them from [50]. We assume the cost of Gaussian elimination of a matrix of dimension $k \times n$ is $\frac{1}{2}k^3 + (n-k)k^2$ qubit operations. This is equal to the classical cost given by Stern [41] and is consistent with the quantum cost Bernstein assumes [11]. We use this cost to compute c_{inv}.

The cost in qubit operations of decoding the input qubits, c_{decode}, is different for each algorithm. An operation which is used multiple times is the transformation of a bitstring of length $\log_2\binom{n}{k}$ into a selection of k out of n elements. To carry out this operation we use the algorithm from [51] with complexity $\mathcal{O}(n^2 \log n)$ to decode an integer into a vector of length n and weight k. We assume the cost in qubit operations is equal to $n^2 \log n$ and denote the cost of this transformation as $\mathrm{c}_{\mathrm{select}}(k, n) = n^2 \log_2 n$. Now the precise decoding cost c_{decode} can be computed for each quantum decoding algorithm in the following way:

- For 'basic ISD', the only parameter that needs to be decoded in each iteration of Grover's algorithm is a selection of k out of n columns. Therefore for basic ISD, $\mathrm{c}_{decode} = \mathrm{c}_{\mathrm{select}}(k, n)$.
- In each iteration of quantum Stern's algorithm, the following three items need to be decoded from the input qubits: a selection of $n-k$ out of n pivot columns, a partition of k integers into two sets X and Y, and a set J containing a selection of l indices out of $n-k$ options. Here l is a parameter of Stern which given n and k can be optimized and set to a fixed value. This respectively costs $\mathrm{c}_{\mathrm{select}}(n-k, n)$, k and $\mathrm{c}_{\mathrm{select}}(l, n-k)$ qubit operations, giving a total cost of $\mathrm{c}_{decode} = \mathrm{c}_{\mathrm{select}}(n-k, n) + k + \mathrm{c}_{\mathrm{select}}(l, n-k)$.
- The input of the quantum MMT algorithm is a selection of $(n-k-l_1)$ out of n columns and a selection of sets $\mathcal{E}_{1,2,3,4}$, containing $\binom{(k+l_1)/2}{m/4}$ vectors of weight $\frac{1}{4}m$ and length $k+l_1$. Sets $\mathcal{E}_{1,2}$ have disjoint supports, as do sets $\mathcal{E}_{3,4}$. The variables m and l_1 are parameters which are set to a fixed value. The selection of columns costs $\mathrm{c}_{\mathrm{select}}(n-k-l_1, n)$ qubit operations. For decoding a string of qubits into sets $\mathcal{E}_{1,2}$ with disjoint supports, we first decode the support of $\mathcal{E}_1$, requiring $\mathrm{c}_{\mathrm{select}}(\frac{k+l_1}{2}, k+l_1)$ qubit operations. The support of $\mathcal{E}_2$ consists of the positions *not* selected for the support of $\mathcal{E}_2$, so the sets indeed have disjoint supports. Next, for both sets, we select $\binom{(k+l_1)/2}{m/4}$ vectors of weight $\frac{1}{4}m$ and length $(k+l_1)/2$ instead of $k+l_1$, because only the vector positions selected in the support may be used. In fact, we have only one possibility here, by selecting *all* vectors of the aforementioned length and weight. This means for this selection of vectors no decoding is needed. However, in order for these vectors to be usable to the algorithm, we do need to decode each of the $2\binom{(k+l_1)/2}{m/4}$ vectors of weight $\frac{1}{4}m$ and length $(k+l_1)/2$. The cost of this is $\binom{(k+l_1)/2}{m/4}\left(\mathrm{c}_{\mathrm{select}}(m/4, (k+l_1)/2)\right)$ qubit operations. Combining these costs, we obtain a decoding cost of:

$$\begin{aligned}\mathrm{c}_{decode} = {} & \mathrm{c}_{\mathrm{select}}(n-k-l_1, n) \\ & + 2\left(\mathrm{c}_{\mathrm{select}}\left(\frac{k+l_1}{2}, k+l_1\right)\right. \\ & \left. + 2\binom{\frac{k+l_1}{2}}{\frac{m}{4}}\mathrm{c}_{\mathrm{select}}\left(\frac{m}{4}, \frac{k+l_1}{2}\right)\right)\end{aligned}$$

- The quantum BJMM algorithm has a selection of $(n-k-l)$ out of n columns, and 8 sets $\mathcal{E}_{1\cdots 8}$ each containing $\binom{(k+l)/2}{p_2/2}$ vectors of weight $\frac{1}{2}p_2$ and length $k+l$ as input. The variables l and p_2 are fixed algorithm parameters. Sets $\mathcal{E}_{2i-1}$ and $\mathcal{E}_{2i}$ have disjoint supports. Using the same reasoning as for the decoding of quantum MMT's inputs, it can be seen that the decoding cost of quantum BJMM is given by:

$$\begin{aligned}\mathrm{c}_{decode} = {} & \mathrm{c}_{\mathrm{select}}(n-k-l, n) \\ & + 4\left(\mathrm{c}_{\mathrm{select}}\left(\frac{k+l}{2}, k+l\right)\right. \\ & \left. + 2\binom{\frac{k+l}{2}}{\frac{p_2}{2}}\mathrm{c}_{\mathrm{select}}\left(\frac{p_2}{2}, \frac{k+l}{2}\right)\right)\end{aligned}$$

The exact formulas for $p_{success}$, c_{decode}, c_{inv} and c_{it} for each algorithm are listed in Appendix A.

3.2 Optimizing parameters for 128-bit quantum security

In order to choose the code parameters such that the public key size is as small as possible, we compute the non-asymptotic complexities of Basic ISD, Stern's algorithm, MMT and BJMM as described above for codes with $3000 \leq n, k \leq 10000$. It turns out that for these codes, quantum Stern is the fastest of these attacks. The complexity of quantum Stern is depicted in Figure 1, and we provide the complexity of the other algorithms in the supplemental materials. The green line in the graph for quantum Stern represents optimal code parameters: for these parameters, there do not exist other parameters that have at least the same attack cost but smaller public key sizes. We compute the key sizes for keys in standard form, so the key size of an $[n, k, d]$ binary code equals $k(n-k)$ bits. There are

two transitions visible in this graph, near $n = 4096$ and $n = 8192$. These transitions occur because the code needs to use a larger field near these values: for binary Goppa codes we need to switch from $\mathbb{GF}(2^{12})$ to $\mathbb{GF}(2^{13})$ and $\mathbb{GF}(2^{14})$ when n becomes greater than 4096 and 8192, respectively.

From the optimal code parameters from Figure 1 we can compute the Niederreiter public key size for a given security threshold. Figure 2 shows the public key size in kilobytes for a given security in bits against both a classical and quantum computer attacker. For example, when a key provides 128 bits of classical (or quantum) security, the best attack needs at least 2^{128} binary operations on a classical (or quantum) computer. In these graphs again some bumps are visible: the key size suddenly increases from approximately 400 KB up to approximately 460 KB. This is caused by a transition from $\mathbb{GF}(2^{12})$ to $\mathbb{GF}(2^{13})$. The 'PQCRYPTO recommendation' graph displays the public key size if Grover can speed up classical attacks exactly by taking the square root. This assumption is made for the initial recommendations of the EU PQCRYPTO project [15]. We have shown that although Grover can significantly reduce the attack cost, this worst-case assumption is overly conservative. A more careful analysis of the cost of quantum information-set decoding yields keys that are almost 35% smaller compared to the size of parameters under this assumption. For comparison, in Table 2 we show the parameters as suggested by the EU PQCRYPTO project and compare them with new parameters which also aim to provide 128-bit security against quantum computer attackers. Interestingly, the EU PQCRYPTO parameters have a security of approximately 240 bits against a classical attacker instead of the expected 256 bits. This can be explained by the fact that the parameters suggested by the EU PQCRYPTO initial recommendation are taken from a paper published in 2008 [21]. Only the BJMM attack, which was published in 2012 [43], can break this algorithm in less than 2^{256} bit operations.

We point out that our new security estimates are still based on conservative assumptions. Firstly, in practice, qubits suffer decoherence and quantum error correction is needed to overcome this. This can add significant overhead, as has been demonstrated by [4]. They estimate the cost of performing a pre-image on SHA-256 at 2^{162} logical-qubit-cycles using a surface code, even though Grover needs only $\frac{\pi}{4}2^{128}$ iterations. Secondly, the advanced information-set decoding algorithms make use of lookup tables. We do not know if this can be efficiently implemented in a quantum algorithm. For example when using Grover to break AES, the authors choose to explicitly calculate the AES `SubBytes` step, because it was considered more 'resource friendly' than using a lookup table [3]. Thirdly, we did not analyse the number of required qubits and quantum gates but just assumed that it is feasible to build a quantum computer large enough to run the algorithms. Especially for algorithms such as BJMM, which may require a substantial amount of memory, this may be problematic in practice. Finally, the quantum attacks only allow for limited parallelization. With Grover, running the algorithm on n quantum computers only makes the algorithm run $\sqrt{n}$ times faster. For these reasons we consider our quantum security estimates as a lower bound on the actual security.

For codes of our interest, with $3000 \leq n \leq 10000$, the quantum Stern algorithm is the fastest quantum information-set decoding algorithm. Basic quantum information-set decoding is more expensive because it needs more iterations, while the quantum MMT and quantum BJMM algorithms need fewer iterations but have a high cost per iteration and decoding qubits into suitable inputs for the algorithm takes more operations. It is, however, possible that decoding qubits into suitable algorithm inputs is actually less expensive than we assume. For example, some algorithm parameters that should be random can actually be set to a fixed value. This will reduce the probability of success per iteration because iterations are no longer independent from each other (see for example [21], section 5), but will decrease decoding cost. It is currently unclear what the smallest possible decoding cost is when this is taken into account. If we assume an absolute lower bound on decoding cost, $c_{decode} = 0$ for all algorithms, the code we suggest in Table 2 actually has a security of 127.94 bits.

TABLE 2
Comparison of Niederreiter parameters for 128-bit security against quantum computers. The PQCRYPTO parameters can be found in [15].

	EU PQCRYTPO	**New parameters**
Parameters $[n, k, d]$	$[6960, 5413, 119]$	$[5542, 4242, 100]$
Public key size	1022 KiB	673 KiB
Quantum security	$2^{153.1}$	$2^{128.0}$
Best quantum attack	Quantum Stern	Quantum Stern
Classical security	$2^{240.4}$	$2^{198.7}$
Best classical attack	BJMM	BJMM

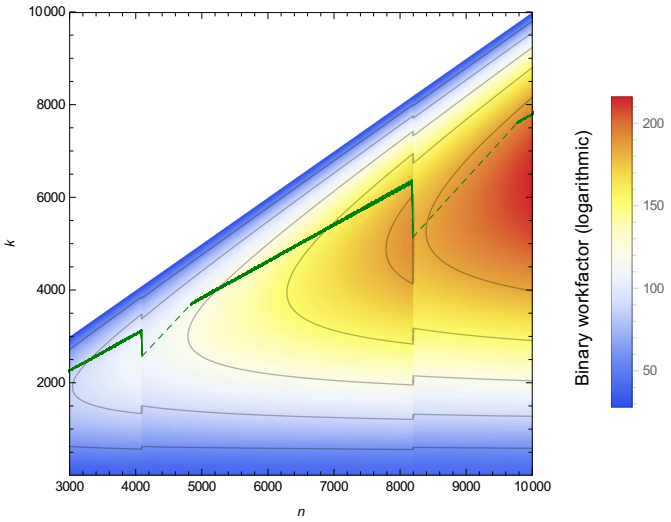

Fig. 1. Binary workfactor for breaking a linear code of length n and rank k with the best quantum information-set decoding attack. The parameters of the smallest code for a certain complexity are given by the green line.

3.3 Asymptotic complexity of quantum attacks

Since complexities of information-set decoding algorithms are all exponential, their asymptotic running times are often compared in exponential factors of the code length n only. In this asymptotic comparison polynomial factors are

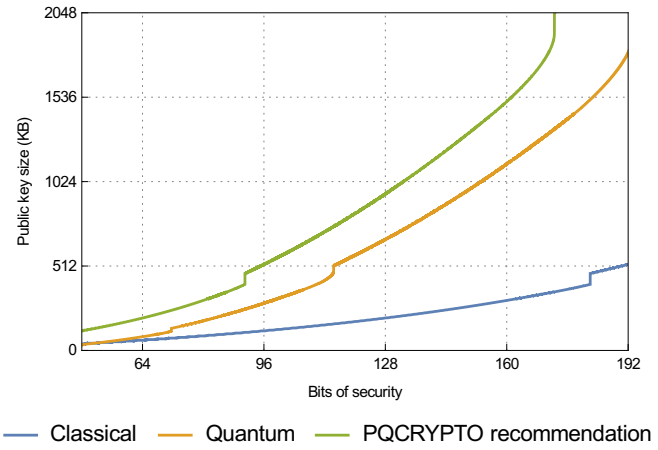

Fig. 2. Security vs. McEliece public key size.

suppressed, i.e. each of the algorithms has an α such that the complexity is in $\tilde{\mathcal{O}}(2^{\alpha n})$ [43]. We derive this complexity from the non-asymptotic workfactor of each algorithm as determined in Section 3.1. We use a variant of Stirling's approximation to approximate binomials:

$$\log_2 \binom{n}{k} \approx nH\left(\frac{k}{n}\right), \tag{3}$$

where $H(p)$ is the binary entropy function $H(p) = -p\log_2(p) - (1-p)\log_2(1-p)$. The complexity of decoding an $[n, k, d]$ linear code does not just depend on the length of the code, but also on the rank k and minimum distance d. For computing the asymptotic complexity, the Gilbert-Varshamov bound is used to compute the maximum possible minimum distance (and therefore maximum error-correcting capacity, for which decoding is most difficult) for a code of length n and rank k. We note that both random codes and binary Goppa codes asymptotically meet this bound [52]. Each attack (except Basic ISD) has a number of parameters that need to be optimized in order to achieve the lowest complexity. By optimizing these parameters for several values of k we can find the value of k for which decoding is most difficult. This gives the asymptotic worst-case complexity of decoding as a function of n. The asymptotic complexities for FS and Ball collision are adapted from [44] and [45], respectively.

A more elaborate description on how asymptotic complexities of information-set decoding algorithms are computed can be found in [43], section 1. We use *bounded distance decoding* as opposed to *full decoding*, because in Niederreiter the number of errors is limited by $\lfloor \frac{d-1}{2} \rfloor$.

The asymptotic complexities of a number of information-set decoding attacks are listed in Table 3. The classical complexities are well-known (see for instance [43]). The quantum complexities are computed in the same way as the classical complexities, but with the original number of iterations n replaced by $\frac{\pi}{4}\sqrt{n}$ and with a higher cost per iteration. Surprisingly, all quantum attacks asymptotically have exactly the same complexity as the most basic information-set decoding attack. It turns out the optimal parameters for the other attacks, for which the attacks have the lowest cost, are all zero. This effectively reduces the more advanced attacks to the most basic attack. This can be explained by the fact that the advanced attacks reduce the number of iterations at the cost of complicating each iteration. Grover, on the other hand, only reduces the number of iterations and does not reduce the cost of each iteration. Also, note that the complexity of basic quantum ISD is exactly half of the complexity of basic classical ISD. This is because the cost per iteration for this algorithm is only polynomial, and therefore the asymptotic complexity is completely determined by the number of iterations. In Figure 3, the factor α in the exponent of the asymptotic running times is shown as a function of the code rate $\frac{n}{k}$. The complexities in Table 3 correspond to the maximum value of the complexities in this graph.

TABLE 3
Overview of information-set decoding attacks and their asymptotic complexities.

	Complexity	
Name of attack	**Classical**	**Quantum**
Basic ISD, 1962 [39]	$2^{0.05752n}$	$2^{0.02876n}$
Stern's algorithm, 1988 [41]	$2^{0.05563n}$	$2^{0.02876n}$
Finiasz and Sendrier (FS), 2009 [44]	$2^{0.05558n}$	$2^{0.02876n}$
Ball collision (BC), 2011 [45]	$2^{0.05558n}$	$2^{0.02876n}$
May *et al.* (MMT), 2011 [42]	$2^{0.05364n}$	$2^{0.02876n}$
Becker *et al.* (BJMM), 2012 [43]	$2^{0.04933n}$	$2^{0.02876n}$

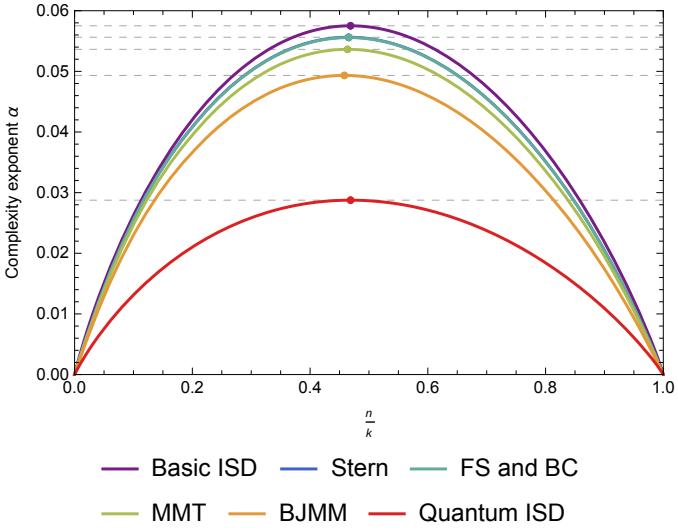

Fig. 3. Asymptotic running time for different code rates n/k and different information-set decoding attacks. Since the complexity of Finiasz and Sendrier and Ball collision differ less than 0.1% from the complexity of Stern's algorithm, these graphs are overlapping.

4 Integrating Niederreiter into OpenVPN

4.1 OpenVPN architecture

We now describe the OpenVPN protocol from a security viewpoint. The exact security controls depend on many configuration options. In this description, we focus on the security mechanisms for OpenVPN when it is used with *'key method 2'* (which is the default and preferred method in OpenVPN 2.0+).

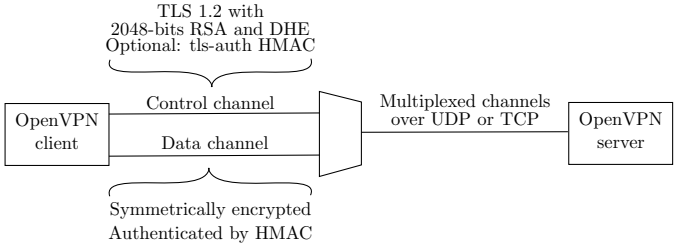

Fig. 4. OpenVPN protocol from a security perspective

As outlined in Figure 4, a connection between an OpenVPN client and OpenVPN server consists of two channels: a *control channel* and a *data channel*.

Control channel – OpenVPN uses the control channel for authentication of both the server and the client and to negotiate keys for the data channel. A TLS session is set up to authenticate both parties and provide a secure channel to exchange keys for the data channel. OpenVPN supports a wide range of TLS ciphers and security mechanisms. It is currently recommended to use a Diffie-Hellman key exchange to provide perfect forward secrecy [53]. The Diffie-Hellman shares are signed with an RSA key. The keys for the data channel are generated using the TLS PRF mechanism, which is based on HMAC.

Data channel – The actual data packets which are sent or received through the OpenVPN tunnel are transmitted over the data channel. Two keys are used for encryption of data; one for each direction. Two other keys are used for authentication of packets in both directions.

These two channels are multiplexed and transmitted over a single UDP or TCP stream. Since UDP is an unreliable transport protocol but TLS requires a reliable transport protocol, an intermediate reliability layer is used for the control channel. Optionally, the control channel can be protected by the `tls-auth` directive, which specifies a pre-shared key by which all incoming packets on the control channel shall be authenticated (using an HMAC). This feature is not essential for security but it makes it more difficult to exploit software vulnerabilities in OpenVPN or the TLS implementation. This is because unauthenticated packets are discarded instead of being processed by the TLS library.

The reason for having a special data channel instead of just using the TLS control channel for data as well is motivated by performance. Since OpenVPN can tunnel TCP packets, and TCP includes a reliability layer, tunneling TCP over the TLS connection will stack two reliability layers. The tunneled TLS connection will never experience any packet loss and will not be able to set its window sizes to correct values. After a timeout happens, packets will be retransmitted even though this is not necessary. To eliminate this performance problem, data is transmitted over an unreliable channel using UDP. A combination of UDP and TCP might have allowed for better performance, but was probably considered unnecessary given the small size of a regular handshake.

Table 4 shows an overview of the security of each OpenVPN security mechanism assuming quantum algorithms are usable in practice. Two components are broken in a quantum world: the RSA-based mutual authentication and Diffie-Hellman key exchange. Both of these components are part of TLS. The fact that mutual authentication is insecure in a quantum world will only be a problem once quantum computers are actually available and can be used to forge signatures. However, since an attacker without a quantum computer can already store key exchanges in order to break them when a quantum computer becomes usable, the key exchange problem is much more urgent. For our research, we will therefore focus on creating secure post-quantum key exchange functionality in OpenVPN.

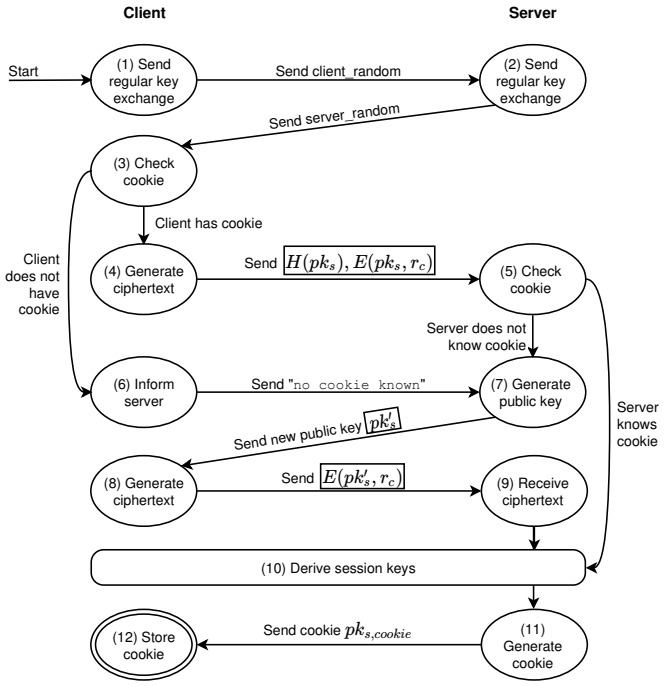

Fig. 5. State diagram of the new key exchange protocol. In this diagram, H is a cryptographically secure hash function and $E(pk_s, r_c)$ is an encryption of message r_c by public key pk_s. Messages inside a box shall be signed by the sender if the protocol is not executed within an authenticated channel.

4.2 Extending OpenVPN

From the description above, the most sensible way for extending the OpenVPN protocol with a post-quantum key exchange is to change the control channel protocol. There are multiple options to accomplish this:

1) Implementing an entirely new key method, 'key method 3'.
2) Changing the current TLS key exchange such that it uses a Niederreiter key exchange instead of (elliptic curve) Diffie-Hellman.
3) Performing a Niederreiter key exchange once the TLS control channel is set up.

A disadvantage of the first method is that it requires duplicating much of the existing control channel logic, such as client and server authentication and exchanging routing information. The second method may seem to be the most elegant and efficient method, but in practice TLS has not been designed for keys as large as Niederreiter's public

TABLE 4
Summary of OpenVPN security in a post-quantum world. We consider a mechanism secure if it is not known to be broken by a quantum computer in less than 2^{128} qubit operations.

Security mechanism	Quantum security
TLS mutual authentication	**Broken.** The RSA key of the root certificate can be factored using Shor's algorithm, allowing an attacker to create rogue certificates.
TLS key exchange	**Broken.** Shor's algorithm can compute the discrete logarithm of the Diffie-Hellman shares, allowing an attacker to obtain the secret keys.
TLS symmetric encryption	**Secure** if AES-256 is used for encryption.
Data channel encryption	**Secure** if AES-256 is used as symmetric cipher.
Data channel authentication	**Secure** if an HMAC with SHA-256 is used for authenticating messages.
`tls-auth` control channel authentication	**Secure** since an HMAC with SHA-256 is used.

keys. Although the TLS handshake protocol specifies a maximum message length of 16 MiB (2^{24} bytes), the TLS records that contain handshake messages actually have a maximum length of 64 KiB (2^{16} bytes) [54]. TLS extension messages are also limited to 64 KiB. We note that there exist recent proposals for post-quantum key exchanges in TLS [55], [56], but these are using cryptosystems with smaller public keys. A disadvantage of the third option regarding performance is that actually two key exchanges are done: a Diffie-Hellman key exchange to set up the TLS connection and then a Niederreiter key exchange. With respect to security, this is in fact an advantage, because an attacker will need to break both key exchanges. In the unlikely event that the Niederreiter cryptosystem is broken, this will not affect security against non-quantum adversaries. Also, a new control channel is set up automatically after a predefined number of seconds or transmitted packets to ensure forward secrecy. By building upon the control channel we can achieve post-quantum forward secrecy. Because this option allows most flexibility to define a custom protocol we choose this method to implement the new key exchange.

As explained in Section 2, Niederreiter can easily be adapted into a key exchange protocol between Alice and Bob in the following way: Alice constructs a Niederreiter keypair and sends the public key to Bob. Bob constructs a random error vector, encrypts it and sends the ciphertext to Alice. After decryption, Alice and Bob both have the same secret error vector. By supplying the error vector as input to a key derivation function a shared key is obtained. This is the basis of our key exchange protocol. In principle, this protocol is vulnerable to adaptive chosen-ciphertext attacks. If a protocol run for a non-decodable ciphertext is distinguishable from a protocol run for a decodable ciphertext an attacker can iteratively decode a syndrome by testing whether the syndrome is still decodable when the syndrome of a single error is added. If the ciphertext is still decodable, the new error actually nullified an existing error and the attacker learns that the original ciphertext contains an error at the same position. Persichetti [17] suggests to solve this by continuing the protocol even if a ciphertext is not decodable. The key should then be derived from the unscrambled ciphertext. However, we use an alternative solution by signing ciphertexts so an attacker cannot adapt them. Since our protocol is run over TLS, all messages are signed already anyway. This also protects against other man-in-the-middle attacks, such as attacks replacing public keys by forged ones.

4.3 Caching Niederreiter keys

With this approach, one challenge still remains. Since a large Niederreiter public key has to be transferred before the handshake can be finished, setting up a VPN connection takes a lot of time. To solve this problem for most practical situations, we have implemented a cache for the public key. Once the handshake is finished the server will generate a new public key and send it to the client. The client will store the public key and use it for the next key exchange. We call this cached public key a *cookie*. If during the next key exchange the server still has knowledge of the corresponding private key, there is no need to send a large public key before the handshake can be finished. In Figure 5, the new key exchange protocol is described in more detail. This protocol is executed over the control channel after the TLS connection has been set up. First, in State (1) and (2), the regular key exchange is done. If both the client and server already have a cached key (which is very likely if they connected with each other before), the client immediately uses the cached public key to encrypt a random error vector in State (4). This drastically speeds up the handshake. If either the server or the client does not have knowledge of a previous key, a new keypair will be generated by the server in State (7). It will take some time to transfer the large public key, but this is unavoidable and should only be necessary when the client and server connect for the first time or when they lose possession of their cached key. In State (10), the shared secret is established. The data channel can then be used and the key exchange is finished. Finally, the server generates a new keypair and sends the new public key to the client. This new keypair will be cached by the server and client so it can be used for the next key exchange. In a way this is similar to the TLS Cached Information Extension, which has been published in July 2016 [57].

There are several reasons why we choose to do key generation on the server. Firstly, the server is in a controlled environment and therefore it is easier to arrange for secure cookie storage on the server. Secondly, key generation is a resource-demanding operation and servers often have more processing power than clients, especially for mobile clients. This does open an opportunity for denial-of-service attacks, but since a keypair is generated only after the TLS control channel is set up, only authenticated clients can carry out this attack. Blocking it is as simple as per-user rate limiting.

Thirdly, clients often have more download bandwidth than upload bandwidth. When the server generates the keypair the public key has to be transferred from the server to the client, which is often faster than the other way around.

4.4 Cookie mechanism robustness and security

The protocol automatically recovers from lost cookies by generating a new keypair on the server. When the cookie storage for the *client* is compromised by an attacker the key exchange is still secure: in case the attacker obtains read access, the attacker can only read public keys. If the attacker obtains write access, the attacker can store new public keys, but the server will not know these keys and therefore reject ciphertexts encrypted by these keys. Because ciphertexts are signed, an attacker cannot perform a man-in-the-middle attack even when he has write access to the client cookie storage. An attacker who obtains either read or write access to *server* cookie storage will be able to break the security of the key exchange. If the cookie cache is shared among multiple servers, an attacker might conceivably be able to remove cookies or restore cookies that have already been used. Apart from being able to collect multiple ciphertexts for a single public key because cookies can be made valid multiple times, this does not affect the key exchange security.

If either the server or the client has a compromised or predictable random number generator, the shared secret can be learned by an eavesdropper. This is because either the private key or the random error vector will be known by the attacker. We are not aware of a way to circumvent this and believe this is acceptable. A single predictable random number generator compromises the Diffie-Hellman and RSA key exchange methods in TLS as well.

Although the presence of a cached private key on the server arguably makes the forward secrecy no longer 'perfect', because potentially more than a single session can be decrypted, the impact of an event in which an attacker grabs hold of all secrets on the server is still very limited. At most two sessions can be decrypted, namely the current session and the next session. To limit the consequences of a security breach, cookies are valid for a limited time. They may be used only for a predefined number of times or be valid within a certain time interval. The latter may be useful for unstable connections with frequent reconnects. Because we combine the Niederreiter key exchange with a regular Diffie-Hellman key exchange, perfect forward secrecy is preserved against non-quantum adversaries.

4.5 Storage

The decoding procedure needs knowledge of the inverse of the scrambling matrix S, the Goppa polynomial and the support vector. Since the scrambling matrix can be computed from the Goppa polynomial and support vector, a space-time trade-off is possible. The following options for storing private keys exist:

1) Store only a seed for a CSPRNG. Storage size: 32 bytes (256 bits). It takes approximately 0.6 seconds to load a private key.
2) Store only the Goppa polynomial and support vector. Storage size: $\frac{1}{8}m(t+n)$ bytes, or about 11 KiB for our parameters. It takes approximately 0.4 seconds to load a private key.
3) Store S^{-1} as well. Storage size: $\frac{1}{8}m(t+n+mt^2)$, or about 215 KiB for our parameters. The private key is immediately available for use.

Because the best way to store private keys depends on how much storage space is available and the number of clients, we implement the last two options and allow the server administrator to make a choice.

4.6 Implementational details and optimizations

We implement Patterson's algorithm for decoding binary Goppa codes [58]. Patterson's algorithm is a specialized algorithm which can correct up to $\lfloor\frac{d-1}{2}\rfloor$ errors in codewords of binary Goppa codes. In our implementation, we have applied the following optimizations. We use lookup tables for multiplications, inversions and square roots in $\mathbb{GF}(2^m)$. These operations only cost a single table lookup, but may make the implementation vulnerable to cache-timing attacks. Additions in $\mathbb{GF}(2^m)$ are simply done by `xor`-instructions. In the key generation procedure, we use the algorithm from Shoup [59] for quickly constructing a random Goppa polynomial, given a precomputed irreducible polynomial of the same degree. Only about 29% of Goppa code parity check matrices can be transformed to standard form. Instead of trying again with a new code when this transformation fails, we swap columns in the parity check matrix and support vector such that the transformation will be successful. In the decoding algorithm, we use an optimization previously applied by Risse [60] to quickly compute the square root of a polynomial modulo the Goppa polynomial with only a single multiplication. We use Horner's method to evaluate the error polynomial. There exist faster methods for finding all roots of the error polynomial, such as the method using additive FFT's which is used by McBits [18].

5 Performance in practice

5.1 Benchmark setup

We evaluate the performance of OpenVPN with Niederreiter key exchange under different network conditions. We connect two virtual machines in a virtual network and run an OpenVPN server on one machine and an OpenVPN client on the other machine. The host machine has an Intel Core i5-3230M CPU. During the benchmarks, the CPU is only the bottleneck for experiments with (nearly) ideal network conditions. When packet drops or additional latency are introduced, the network becomes the bottleneck. We use the Linux kernel Queuing Disciplines Traffic Control, that can be managed by the `tc qdisc` command, to control networking characteristics. We evaluate OpenVPN's performance for packet losses between 0 and 2% for the classical OpenVPN client and the OpenVPN client with Niederreiter key exchange, with and without the cookie mechanism. The results are displayed in Figure 6.

In Table 5, Niederreiter's performance in practice is summarized. The timing measurements are done on the same machine on which the benchmarks are done, and are averaged over 1.000 runs. The time for the initial setup consists

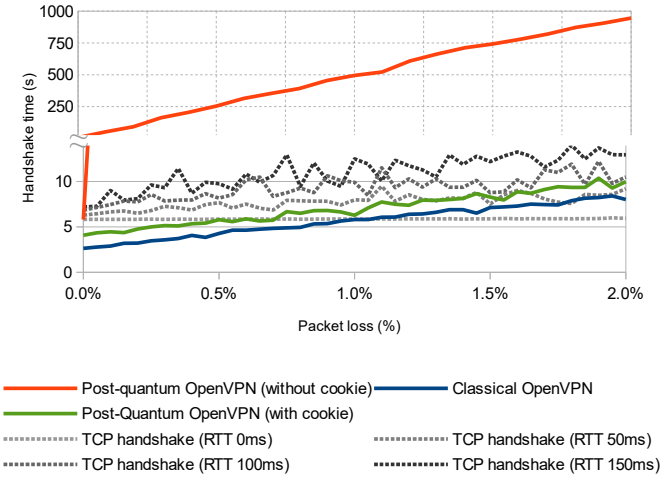

Fig. 6. The time of a full handshake for different packet loss rates. Each data point represents the average of at least 100 runs, except the 'Post-quantum OpenVPN no cookie' graph, which is averaged over 20 runs. The TCP graph shows how post-quantum OpenVPN would perform for various round-trip times when TCP would be used as reliability layer.

TABLE 5
Niederreiter in practice: amount of time and randomness required for cryptographic operations, key and ciphertext sizes and actual handshake times.

$[n, k, d]$	[5730, 4430, 100]
Time required for cryptographic operations	
Initial setup	1.404 seconds
Key generation	0.632 seconds
Encryption	0.002 seconds
Decryption	0.185 seconds
Randomness needed	
For key generation	22.8 KiB
For encryption	0.4 KiB
Size of cryptographic data structures	
Public key size	703.0 KiB
Ciphertext length	163 bytes
Handshake times	
Classical OpenVPN	2.64 seconds
Post-quantum OpenVPN (with cookie)	4.08 seconds
Post-quantum OpenVPN (without cookie)	5.78 seconds

of constructing and precomputing operations in the $\mathbb{GF}(2^{13})$ finite field. This is only needed once. For key generation, transforming the matrix into standard form takes most time: approximately 0.2 seconds. For decoding, the most expensive operation is evaluating the error polynomial, which takes approximately 0.1 seconds. The rows under 'randomness needed' show the number of (pseudo)random bytes needed to generate a key and generation of a shared secret. This is not necessarily the same as the required amount of entropy, since using 256 bits of entropy is sufficient if a CSPRNG is used. The handshake time is the time needed for a complete handshake, measured as the time interval between starting the client and server and the moment the OpenVPN tunnel is set up under ideal network conditions.

5.2 Analysis

From Figure 6, we can conclude that the handshake time for post-quantum OpenVPN without a cookie explodes when network conditions degrade. This is caused by the transfer of the public key of almost 700 KiB and by OpenVPN's reliability layer for the control channel, which is very inefficient. The reliability layer has a very small window size of 8 packets, and only detects a packet loss when a timeout of 2 seconds occurs. Effectively this means every time a packet loss happens, the handshake time is increased by 2 seconds. A better reliability layer can significantly improve OpenVPN's performance for networks with high packet loss. For comparison, we also show the OpenVPN handshake time for the case where TCP would be used as reliability layer for the control channel in Figure 6. We have not actually implemented this, but simulated this scenario by sending exactly the same packets over a TCP connection (taking into account that each party can only respond when certain packets have been received) and added the cost of cryptographic operations. We did so for various round-trip times, ranging from 0 to 150 ms. As we can see from Figure 6, replacing OpenVPN's reliability layer by TCP drastically reduces the handshake time when no cookie exists. Unfortunately, even though OpenVPN supports running the entire OpenVPN protocol over TCP, its internal reliability layer will then still be used.

On the other hand, the cookie-mechanism, which exchanges a shared secret using a public key previously sent, works quite well. It is only about 1.6 seconds slower than classical OpenVPN. This is due to the time initial setup, encryption and decryption takes. The cost of initial setup only applies to the server, and only once for all clients, so in practice the difference in handshake time between classical OpenVPN and OpenVPN with a cookie is much lower. Because a ciphertext is only 163 bytes and only few other additional messages are needed for the post-quantum handshake, the impact of packet loss is very comparable with classical OpenVPN. However, when the handshake is finished, it is still needed to transfer a large public key for the next key exchange. This may still prevent this solution from being suitable for devices that need to do frequent quantum-secure key exchanges with very limited bandwidth or a low data cap. We also note that the current implementation of OpenVPN lacks support for multithreading. Therefore, the data channel cannot be used during key generation and public key transfer. Because Niederreiter keys are completely independent from clients and specific connections, it is possible to optimize key generation by generating keys beforehand in a separate thread for all clients and maintaining a 'pool' of Niederreiter keypairs on the server.

6 Conclusions and Recommendations

Our main contributions are a more careful analysis of quantum computer attacks on McEliece, resulting in much smaller parameters providing 128-bit security against quantum computers, a public implementation of McEliece in

an open source product and a way to cope with the large public keys in practice. We have shown that although the complexities of quantum information-set decoding attacks are asymptotically exactly the same, non-asymptotically the binary workfactor is different for each algorithm and the quantum information set decoding variant of Stern's algorithm is the most efficient quantum decoding algorithm currently known. We demonstrated and evaluated the usability of McEliece in practice. We conclude that in ideal network conditions McEliece can be used in practical applications to establish a shared secret key. In networks with high packet loss, OpenVPN's inefficient reliability mechanism is unsuitable for sending large public keys.

We recommend to replace OpenVPN's reliability layer by a more efficient one, such as TCP. We also recommend to estimate the number of logical qubit cycles and gate operations more precisely, taking overhead from quantum error correction and transforming the circuit into a reversible circuit into account.

APPENDIX A
NON-ASYMPTOTIC COST OF QUANTUM INFORMATION-SET DECODING ALGORITHMS

In this appendix, we list the exact cost of several quantum information-set decoding algorithms. For algorithms that have additional parameters, these parameters are optimized to minimize the cost of the algorithm. The final workfactor can be computed using Equation 2 on page 5. The cost of transforming a bitstring of length $\log_2 \binom{n}{k}$ into a vector of length n and weight k is defined as $\mathrm{c}_{\mathrm{select}}(k,n) = n^2 \log_2 n$. For a description on how to derive these formulas, see Section 3.1.

A.1 Quantum information-set decoding

The basic information-set decoding algorithm does not have any parameters which can be optimized. The only parameter that needs to be decoded in each iteration of Grover's algorithm is the selection of k out of n columns.

$$p_{success} = \frac{\binom{n-d}{k}}{\binom{n}{k}}, \quad \mathrm{c}_{inv} = \frac{1}{2}k^3 + (n-k)k^2,$$
$$\mathrm{c}_{decode} = \mathrm{c}_{\mathrm{select}}(k,n), \quad \mathrm{c}_{it} = k.$$

A.2 Quantum Stern's algorithm

Stern's algorithm has two parameters: l and m. In each iteration of Grover's algorithm, the following three items need to be decoded from the input qubits: the selection of $n-k$ out of n pivot columns, a partition of k integers into two sets X and Y, and a set J containing a selection of l indices out of $n-k$ options.

$$p_{success} = \frac{\binom{d}{2m}\binom{n-d}{k-2m}\binom{2m}{m}\binom{n-k-d+2m}{l}}{4^m\binom{n}{k}\binom{n-k}{l}},$$
$$\mathrm{c}_{decode} = \mathrm{c}_{\mathrm{select}}(n-k,n) + k + \mathrm{c}_{\mathrm{select}}(l,n-k),$$
$$\mathrm{c}_{it} = 2lm\binom{\frac{k}{2}}{m} + 2m(n-k)\binom{\frac{k}{2}}{m}^2 2^{-l},$$
$$\mathrm{c}_{inv} = \frac{1}{2}(n-k)^3 + k(n-k)^2.$$

A.3 Quantum MMT's algorithm

MMT's algorithm has four parameters: m, l_1, l_2 and $|A|$. The input of this algorithm is a selection of $(n-k-l_1)$ out of n columns and a selection of sets $\mathcal{E}_{1,2,3,4}$, containing $\binom{(k+l_1)/2}{m/4}$ vectors of weight $\frac{1}{4}m$ and length $k+l_1$. Sets $\mathcal{E}_{1,2}$ have disjoint supports, as do sets $\mathcal{E}_{3,4}$.

$$p_{success} = 1 - \left(1 - \varepsilon\, 2^{l_1}\right)^{|\mathcal{E}|},$$
$$\mathrm{c}_{inv} = \frac{1}{2}(n-k-l)^3 + (k+l)(n-k-l)^2,$$
$$\begin{aligned}\mathrm{c}_{decode} = {} & \mathrm{c}_{\mathrm{select}}(n-k-l_1,n) \\ & + 2\left(\mathrm{c}_{\mathrm{select}}\left(\frac{k+l_1}{2}, k+l_1\right)\right. \\ & \left. + 2\binom{\frac{k+l_1}{2}}{\frac{m}{4}}\mathrm{c}_{\mathrm{select}}\left(\frac{m}{4}, \frac{k+l_1}{2}\right)\right)\end{aligned}$$
$$\mathrm{c}_{it} = |A|(n-k)\left(4L_0 + 2L_0^2\, 2^{-l_2} + 2L_0^4\, 2^{-l_1-l_2}\right),$$
$$\varepsilon = \frac{\binom{n-k-l_1}{d-m}}{\min\left(2^{n-k}, \binom{n}{d}\right)}, \quad L_0 = \binom{\frac{k+l_1}{2}}{\frac{m}{4}},$$
$$|\mathcal{E}| = |A|\, L_0^4\, 2^{-l_1-l_2}.$$

A.4 Quantum BJMM's algorithm

The BJMM algorithm has six parameters: m, l, r_1, r_2, e_1 and e_2. The inputs that need to be decoded for each iteration of Grover's algorithm are a selection of $(n-k-l)$ out of n columns, and 8 sets $\mathcal{E}_{1\cdots8}$ each containing $\binom{(k+l)/2}{p_2/2}$ vectors of weight $\frac{1}{2}p_2$ and length $k+l$. Sets $\mathcal{E}_{2i-1}$ and $\mathcal{E}_{2i}$ have disjoint supports.

$$p_{success} = 1 - \left(1 - \varepsilon\, 2^{l}\right)^{S_0},$$
$$\mathrm{c}_{inv} = \frac{1}{2}(n-k-l)^3 + (k+l)(n-k-l)^2,$$
$$\mathrm{c}_{it} = (n-k)(8S_3 + 4C_3 + 2C_2 + 2C_1),$$
$$\begin{aligned}\mathrm{c}_{decode} = {} & \mathrm{c}_{\mathrm{select}}(n-k-l,n) \\ & + 4\left(\mathrm{c}_{\mathrm{select}}\left(\frac{k+l}{2}, k+l\right)\right. \\ & \left. + 2\binom{\frac{k+l}{2}}{\frac{p_2}{2}}\mathrm{c}_{\mathrm{select}}\left(\frac{p_2}{2}, \frac{k+l}{2}\right)\right)\end{aligned}$$
$$\varepsilon = \frac{\binom{n-k-l}{d-m}}{\min\left(2^{n-k}, \binom{n}{d}\right)}, \quad m_1 = \frac{m}{2} + e_1, \quad m_2 = \frac{m_1}{2} + e_2,$$
$$\mu_1 = \frac{\binom{m_1}{e_1}\binom{k+l-m_1}{m_1-e_1}}{\binom{k+l}{m_1}}, \quad \mu_2 = \frac{\binom{m_2}{e_2}\binom{k+l-m_2}{m_2-e_2}}{\binom{k+l}{m_2}},$$
$$S_0 = \min\left(\mu_1 C_1,\, 2^{-l}\binom{k+l}{m}\right),$$
$$S_1 = \min\left(\mu_2 C_2,\, 2^{-r_1-r_2}\binom{k+l}{m_1}\right),$$
$$S_2 = C_3 = 2^{-r_2} S_3^2, \quad S_3 = \binom{\frac{k+l}{2}}{\frac{m_2}{2}},$$
$$C_1 = 2^{r_1+r_2-l}\, S_1^2, \quad C_2 = 2^{-r_1}\, S_2^2.$$

REFERENCES

[1] OpenVPN Technologies, Inc., "What is OpenVPN?" https://openvpn.net/index.php/open-source/333-what-is-openvpn.html, 2016, accessed: 08-01-2016.

[2] D. J. Bernstein, "Introduction to post-quantum cryptography," in *Post-quantum cryptography*. Springer, 2009, pp. 1 – 14.

[3] M. Grassl, B. Langenberg, M. Roetteler, and R. Steinwandt, "Applying Grover's Algorithm to AES: Quantum Resource Estimates," in *Post-Quantum Cryptography: 7th International Workshop, PQCrypto 2016, Fukuoka, Japan, February 24-26, 2016, Proceedings*, vol. 9606. Springer, 2016, p. 29.

[4] M. Amy, O. Di Matteo, V. Gheorghiu, M. Mosca, A. Parent, and J. Schanck, "Estimating the cost of generic quantum pre-image attacks on SHA-2 and SHA-3," *arXiv preprint arXiv:1603.09383*, 2016.

[5] L. Chen, S. Jordan, Y.-K. Liu, D. Moody, R. Peralta, R. Perlner, and D. Smith-Tone, "Report on Post-Quantum Cryptography," *National Institute of Standards and Technology Internal Report*, feb 2016.

[6] M. Mariantoni. (2014, oct) Building a Superconducting Quantum Computer. https://www.youtube.com/watch?v=wWHAs--HA1c.

[7] N. Xu, J. Zhu, D. Lu, X. Zhou, X. Peng, and J. Du, "Quantum Factorization of 143 on a Dipolar-Coupling Nuclear Magnetic Resonance System," *Phys. Rev. Lett.*, vol. 108, p. 130501, Mar 2012. [Online]. Available: http://link.aps.org/doi/10.1103/PhysRevLett.108.130501

[8] N. S. Dattani and N. Bryans, "Quantum factorization of 56153 with only 4 qubits," *arXiv preprint arXiv:1411.6758*, 2014.

[9] R. J. McEliece, "A public-key cryptosystem based on algebraic coding theory," *DSN progress report*, vol. 42, no. 44, pp. 114 – 116, 1978.

[10] H. Dinh, C. Moore, and A. Russell, "McEliece and Niederreiter Cryptosystems That Resist Quantum Fourier Sampling Attacks," in *Annual Cryptology Conference*. Springer, 2011, pp. 761–779.

[11] D. J. Bernstein, "Grover vs. McEliece," in *Post-Quantum Cryptography*. Springer, 2010, pp. 73 – 80.

[12] E. R. Berlekamp, R. J. McEliece, and H. C. Van Tilborg, "On the inherent intractability of certain coding problems," *IEEE Transactions on Information Theory*, vol. 24, no. 3, pp. 384 – 386, 1978.

[13] H. Niederreiter, "Knapsack-type cryptosystems and algebraic coding theory," *PROBLEMS OF CONTROL AND INFORMATION THEORY*, vol. 15, no. 2, pp. 159 – 166, 1986.

[14] Y. X. Li, R. H. Deng, and X. M. Wang, "On the equivalence of McEliece's and Niederreiter's public-key cryptosystems," *IEEE Transactions on Information Theory*, vol. 40, no. 1, pp. 271 – 273, 1994.

[15] D. Augot, L. Batina, D. J. Bernstein, J. Bos, J. Buchmann, W. Castryck, O. Dunkelman, T. Güneysu, S. Gueron, A. Hülsing *et al.*, "Initial recommendations of long-term secure post-quantum systems," 2015.

[16] V. Shoup, "A proposal for an ISO standard for public key encryption (version 2.1)," *IACR E-Print Archive*, vol. 112, 2001.

[17] E. Persichetti, "Improving the efficiency of code-based cryptography," Ph.D. dissertation, Department of Mathematics, University of Auckland, 2012.

[18] D. J. Bernstein, T. Chou, and P. Schwabe, "McBits: fast constant-time code-based cryptography," in *Cryptographic Hardware and Embedded Systems-CHES 2013*. Springer, 2013, pp. 250 – 272.

[19] D. J. Bernstein, T. Lange, and C. Peters, "Wild McEliece," in *Selected Areas in Cryptography*. Springer, 2011, pp. 143 – 158.

[20] A. Canteaut and N. Sendrier, "Cryptanalysis of the original McEliece cryptosystem," in *Advances in Cryptology - ASIACRYPT'98*. Springer, 1998, pp. 187 – 199.

[21] D. J. Bernstein, T. Lange, and C. Peters, "Attacking and defending the McEliece cryptosystem," in *Post-Quantum Cryptography*. Springer, 2008, pp. 31 – 46.

[22] R. Misoczki, J.-P. Tillich, N. Sendrier, and P. S. Barreto, "MDPC-McEliece: New McEliece variants from moderate density parity-check codes," in *Information Theory Proceedings (ISIT), 2013 IEEE International Symposium on*. IEEE, 2013, pp. 2069 – 2073.

[23] Y. Wang, "Random linear code based public key encryption scheme rlce," 2016.

[24] V. M. Sidelnikov and S. O. Shestakov, "On insecurity of cryptosystems based on generalized Reed-Solomon codes," *Discrete Mathematics and Applications*, vol. 2, no. 4, pp. 439 – 444, 1992.

[25] E. M. Gabidulin, A. Paramonov, and O. Tretjakov, "Ideals over a non-commutative ring and their application in cryptology," in *Advances in Cryptology - EUROCRYPT'91*. Springer, 1991, pp. 482 – 489.

[26] E. Gabidulin, "On public-key cryptosystems based on linear codes," in *In Proc. of 4th IMA Conference on Cryptography and Coding 1993, Codes and Ciphers*. IMA Press, 1995.

[27] J. Gibson, "Severely denting the Gabidulin version of the McEliece public key cryptosystem," *Designs, Codes and Cryptography*, vol. 6, no. 1, pp. 37 – 45, 1995.

[28] K. Gibson, "The security of the Gabidulin public key cryptosystem," in *Advances in Cryptology - EUROCRYPT'96*. Springer, 1996, pp. 212 – 223.

[29] V. M. Sidelnikov, "A public-key cryptosystem based on binary Reed-Muller codes," *Discrete Mathematics and Applications*, vol. 4, no. 3, pp. 191 – 208, 1994.

[30] L. Minder and A. Shokrollahi, "Cryptanalysis of the Sidelnikov cryptosystem," in *Advances in Cryptology - EUROCRYPT 2007*. Springer, 2007, pp. 347 – 360.

[31] P. Gaborit, "Shorter keys for code based cryptography," in *Proceedings of the 2005 International Workshop on Coding and Cryptography (WCC 2005)*, 2005, pp. 81 – 91.

[32] A. Otmani, J.-P. Tillich, and L. Dallot, "Cryptanalysis of two McEliece cryptosystems based on quasi-cyclic codes," *Mathematics in Computer Science*, vol. 3, no. 2, pp. 129 – 140, 2010.

[33] M. Baldi and F. Chiaraluce, "Cryptanalysis of a new instance of McEliece cryptosystem based on QC-LDPC codes," in *Information Theory, 2007. ISIT 2007. IEEE International Symposium on*. IEEE, 2007, pp. 2591 – 2595.

[34] A. Couvreur, A. Otmani, and J.-P. Tillich, "Polynomial time attack on wild McEliece over quadratic extensions," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2014, pp. 17–39.

[35] J.-C. Faugere, L. Perret, and F. De Portzamparc, "Algebraic Attack against Variants of McEliece with Goppa Polynomial of a Special Form," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2014, pp. 21–41.

[36] D. J. Bernstein, T. Lange, and C. Peters, *Wild McEliece Incognito*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 244–254. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-25405-5_16

[37] R. Dowsley, J. Müller-Quade, and A. C. Nascimento, "A CCA2 secure public key encryption scheme based on the McEliece assumptions in the standard model," in *Topics in Cryptology - CT-RSA 2009*. Springer, 2009, pp. 240 – 251.

[38] J.-C. Faugere, V. Gauthier-Umana, A. Otmani, L. Perret, and J.-P. Tillich, "A distinguisher for high-rate McEliece cryptosystems," *IEEE Transactions on Information Theory*, vol. 59, no. 10, pp. 6830 – 6844, 2013.

[39] E. Prange, "The use of information sets in decoding cyclic codes," *Information Theory, IRE Transactions on*, vol. 8, no. 5, pp. 5 – 9, 1962.

[40] P. J. Lee and E. F. Brickell, "An observation on the security of McEliece's public-key cryptosystem," in *Advances in Cryptology - EUROCRYPT'88*. Springer, 1988, pp. 275 – 280.

[41] J. Stern, "A method for finding codewords of small weight," in *Coding theory and applications*. Springer, 1988, pp. 106 – 113.

[42] A. May, A. Meurer, and E. Thomae, "Decoding Random Linear Codes in $\tilde{\mathcal{O}}(2^{0.054n})$," in *Advances in Cryptology - ASIACRYPT 2011*. Springer, 2011, pp. 107 – 124.

[43] A. Becker, A. Joux, A. May, and A. Meurer, "Decoding random binary linear codes in $2^{n/20}$: How 1 + 1 = 0 improves information set decoding," in *Advances in Cryptology - EUROCRYPT 2012*. Springer, 2012, pp. 520 – 536.

[44] M. Finiasz and N. Sendrier, "Security bounds for the design of code-based cryptosystems," in *Advances in Cryptology - ASIACRYPT 2009*. Springer, 2009, pp. 88 – 105.

[45] D. J. Bernstein, T. Lange, and C. Peters, "Smaller decoding exponents: ball-collision decoding," in *Advances in Cryptology - CRYPTO 2011*. Springer, 2011, pp. 743 – 760.

[46] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. ACM, 1996, pp. 212 – 219.

[47] C. Zalka, "Grover's quantum searching algorithm is optimal," *Physical Review A*, vol. 60, no. 4, p. 2746, 1999.

[48] M. Boyer, G. Brassard, P. Høyer, and A. Tapp, "Tight bounds on quantum searching," *arXiv preprint quant-ph/9605034*, 1996.

[49] C. H. Bennett, E. Bernstein, G. Brassard, and U. Vazirani, "Strengths and weaknesses of quantum computing," *SIAM journal on Computing*, vol. 26, no. 5, pp. 1510 – 1523, 1997.
[50] Y. Hamdaoui and N. Sendrier, "A Non Asymptotic Analysis of Information Set Decoding," *IACR Cryptology ePrint Archive*, vol. 2013, p. 162, 2013.
[51] J.-B. Fischer and J. Stern, "An efficient pseudo-random generator provably as secure as syndrome decoding," in *Advances in Cryptology - EUROCRYPT'96*. Springer, 1996, pp. 245 – 255.
[52] W. C. Huffman and V. Pless, *Fundamentals of error-correcting codes*. Cambridge university press, 2010.
[53] OpenVPN Technologies, Inc., "OpenVPN Howto," https://openvpn.net/index.php/open-source/documentation/howto.html, 2016, accessed: 28-07-2016.
[54] T. Dierks and E. Rescorla, "RFC 5246: The Transport Layer Security (TLS) Protocol Version 1.2," Tech. Rep., August 2008.
[55] J. W. Bos, C. Costello, M. Naehrig, and D. Stebila, "Post-quantum key exchange for the TLS protocol from the ring learning with errors problem," in *2015 IEEE Symposium on Security and Privacy*. IEEE, 2015, pp. 553–570.
[56] J. M. Schanck, W. Whyte, and Z. Zhang, "Quantum-Safe Hybrid (QSH) Ciphersuite for Transport Layer Security (TLS) version 1.3 (draft)," *IETF*, April 2016.
[57] S. Santesson and H. Tschofenig, "Transport Layer Security (TLS) Cached Information Extension," *IETF*, July 2016.
[58] N. Patterson, "The algebraic decoding of Goppa codes," *IEEE Transactions on Information Theory*, vol. 21, no. 2, pp. 203–207, 1975.
[59] V. Shoup, "Fast construction of irreducible polynomials over finite fields," *Journal of Symbolic Computation*, vol. 17, no. 5, pp. 371–391, 1994. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S074771718471025X
[60] T. Risse, "How SAGE helps to implement Goppa Codes and the McEliece Public Key Crypto System," *Ubiquitous Computing and Communication Journal, UbiCC, ISSN*, vol. 8424, 1992.