Classic McEliece: conservative code-based cryptography: design rationale

23 October 2022

Contents

1 One-wayness

There is a long history of trapdoor systems (in modern terminology: PKEs) that are designed to be one-way (in modern terminology: OW-CPA). One-wayness means that it is difficult to invert the map from input to ciphertext, given the public key, when the input is chosen uniformly at random.

The McEliece system is one of the oldest proposals, almost as old as RSA. RSA has suffered dramatic security losses, while the McEliece system has maintained a spectacular security track record unmatched by any other proposals for post-quantum encryption. This is the fundamental reason to use the McEliece system.

Here is more detail to explain what "spectacular security track record" means.

With the key-size optimizations discussed below, the McEliece system uses a key size of $(c_0 + o(1))b^2(\log_2 b)^2$ bits to achieve 2^b security against all inversion attacks that were known in 1978, when the system was introduced. Here $o(1)$ means something that converges to 0 as $b \to \infty$, and $c_0 \approx 0.7418860694$.

The best attack at that time was from 1962 Prange [43]. After 1978 there were 30 publications studying the one-wayness of the system and introducing increasingly sophisticated non-quantum attack algorithms:

1. 1981 Clark–Cain [20], crediting Omura.
2. 1988 Lee–Brickell [33].
3. 1988 Leon [34].
4. 1989 Krouk [32].
5. 1989 Stern [46].
6. 1989 Dumer [25].
7. 1990 Coffey–Goodman [21].
8. 1990 van Tilburg [48].
9. 1991 Dumer [26].
10. 1991 Coffey–Goodman–Farrell [22].
11. 1993 Chabanne–Courteau [18].
12. 1993 Chabaud [19].
13. 1994 van Tilburg [49].
14. 1994 Canteaut–Chabanne [14].
15. 1998 Canteaut–Chabaud [15].
16. 1998 Canteaut–Sendrier [16].

17. 2008 Bernstein–Lange–Peters [9].
18. 2009 Bernstein–Lange–Peters–van Tilborg [11].
19. 2009 Finiasz–Sendrier [29].
20. 2011 Bernstein–Lange–Peters [10].
21. 2011 May–Meurer–Thomae [35].
22. 2012 Becker–Joux–May–Meurer [2].
23. 2013 Hamdaoui–Sendrier [31].
24. 2015 May–Ozerov [36].
25. 2016 Canto Torres–Sendrier [47].
26. 2017 Both–May [12].
27. 2018 Both–May [13].
28. 2020 Debris-Alazard–Ducas–van Woerden [24].
29. 2021 Esser–May–Zweydinger [27].
30. 2022 Carrier–Debris-Alazard–Meyer-Hilfiger–Tillich [17].

What is the cumulative impact of all this work? Answer: With the same key-size optimizations, the McEliece system uses a key size of $(c_0 + o(1))b^2(\log_2 b)^2$ bits to achieve 2^b security against all non-quantum attacks known today, where c_0 is exactly the same constant. All of the improvements have disappeared into the $o(1)$.

This does not mean that the required key size is precisely the same—that dozens of attack papers over 40 years have not accomplished *anything*. What it means is that the required change in key size is below 1% once b is large enough; below 0.1% once b is large enough; etc. This is a remarkably stable security story.

What about quantum attacks? Grover's algorithm is applicable, reducing the attack cost to asymptotically its square root (essentially the same situation as for strong symmetric-key ciphers); see generally [4]. In other words, the key now needs $(4c_0 + o(1))b^2(\log_2 b)^2$ bits. As before, further papers on the topic have merely improved the $o(1)$.

The most effective attack strategy known, the central focus of this long list of papers, is "information-set decoding". This strategy does not exploit any particular structure of a matrix G: it recovers a low-weight error vector e given a *uniform random* matrix G and $Ga + e$ for some a. Experiments are consistent with the theory that McEliece's matrices G behave like uniform random matrices in this context.

There are also many papers studying attacks that instead recover McEliece's private key—a polynomial g and distinct elements $\alpha_0, \ldots, \alpha_{n-1}$ of a finite field $\mathbb{F}_q$—from the public key G. Recovering the private key also breaks one-wayness, since the attacker can then use the receiver's decryption algorithm. These attacks can be much faster than a brute-force search through private keys: for example, Sendrier's "support splitting" algorithm [44] quickly finds $\alpha_0, \ldots, \alpha_{n-1}$ given g provided that $n = q$. More generally, whether or not $n = q$, support splitting finds $\alpha_0, \ldots, \alpha_{n-1}$ given g and given the *set* $\{\alpha_0, \ldots, \alpha_{n-1}\}$. (This can be viewed as a reason to keep n somewhat smaller than q, since then there are many possibilities for the set, along with many possibilities for g; most of the selected parameter sets provide this extra defense.) However, despite this and other interesting speedups, the state-of-the-art key-recovery attacks are vastly slower than information-set decoding.

Various authors have proposed replacing the binary Goppa codes in McEliece's system with other families of codes: see, e.g., [1, 3, 37, 41, 42, 38]. Often these replacements are advertised as allowing smaller public keys. Unfortunately, many of these proposals have turned out to allow unacceptably fast recovery of the private key (or of something equivalent to the private key, something that allows fast inversion of the supposedly one-way function). Some small-key proposals are unbroken, but Classic McEliece uses binary Goppa codes, the traditional, conservative, well-studied choice.

Authors of attacks on other codes often study the performance of their attacks against binary Goppa codes. These studies consistently show that McEliece's system is far beyond all known attacks. For example, 2013 Faugère–Gauthier-Umaña–Otmani–Perret–Tillich [28] showed that "high-rate" binary Goppa codes can be distinguished from random codes. The worst-case possibility is that this distinguisher somehow allows an inversion attack faster than attacks for random codes. However, the distinguisher stops working

- at 8 errors for $n = 1024$ (where McEliece's original parameters used 50 errors),
- at 20 errors for $n = 8192$ (where the selected parameters use between 96 and 128 errors),

etc. As another example, the attack in [17] saves time for code rates below 0.3; McEliece's original parameters used code rate 0.5, and the selected parameters use rates between 0.7 and 0.8. As yet another example, the attack in [23] reaches degree $m = 2$; McEliece's original parameters used degree $m = 10$, and the selected parameters use degree $m = 12$ or $m = 13$.

2 Better efficiency for the same one-wayness

The main focus of Classic McEliece is security, but Classic McEliece also takes reasonable steps to improve efficiency when this clearly does not compromise security. In particular, Classic McEliece includes the following three modifications, the first two of which are well known.

2.1 Systematic-form public keys

The goal of the public key in McEliece's system is to communicate an $[n, k]$ linear code C over $\mathbb{F}_2$: a k-dimensional linear subspace of $\mathbb{F}_2^n$. This means communicating the ability to generate uniform random elements of C. McEliece accomplished this by choosing the public key to be a uniform random generator matrix G for C: specifically, multiplying any generator matrix for C by a uniform random invertible matrix. A generator matrix G for C means a matrix $G \in \mathbb{F}_2^{n \times k}$ such that $C = \left\{Ga : a \in \mathbb{F}_2^k\right\}$.

The first modification accomplishes this by instead choosing the public key to be the unique systematic-form generator matrix for C if one exists. This means a generator matrix of the form $\left(\frac{T}{I_k}\right)$ where T is some $(n-k) \times k$ matrix and I_k is the $k \times k$ identity matrix. Approximately 29% of choices of C have this form, so key generation requires about 3.4 attempts on average, but now the public key occupies only $k(n-k)$ bits instead of kn bits. Note that sending a systematic-form generator matrix also implies sending a parity-check matrix H for C, namely $(I_{n-k} \mid T)$.

This modification changes security by at most 2 bits. See the separate "guide for security reviewers" document.

2.2 Syndromes as ciphertexts

McEliece's ciphertext has the form $Ga + e$. Here G is a random $n \times k$ generator matrix for a code C as above; a is a column vector of length k; e is a weight-t column vector of length n; and the ciphertext is a column vector of length n. McEliece's inversion problem is to compute (a, e) given G and the ciphertext $Ga + e$, where a is a uniform random column vector of length k; e is a uniform random weight-t column vector of length n; and a, e are independent.

Niederreiter [41] instead suggested a ciphertext of the form He. Here H is a parity-check matrix for C used as a public key, and e is a weight-t column vector of length n, so the ciphertext is a column vector of length just $n-k$, shorter than McEliece's ciphertext. Niederreiter's inversion problem is to compute e given H and the ciphertext He, where e is a uniform random weight-t vector of length n.

This modification makes negligible difference in security. See the separate "guide for security reviewers" document.

2.3 Semi-systematic form

As a generalization (introduced by Chou) of the idea of systematic form, consider any key obtained as follows:

- Starting from the secret parity-check matrix for the code C, compute the unique parity-check matrix in reduced row-echelon form.
- Start over with a new code if this matrix is not acceptable. This generalization is parameterized by the definition of acceptability: e.g., one can define an acceptable matrix as a matrix in (μ, ν)-semi-systematic form.

- Permute the matrix columns to reach systematic form, while permuting the code accordingly. This requires all acceptable matrices to have full rank.

It is important here for the second and third steps to depend only on the reduced row-echelon form. This guarantees that any attack against the resulting public key can be converted into an attack against McEliece's public key: anyone can convert McEliece's public key into the parity-check matrix in reduced row-echelon form, and then follow the second and third steps.

Accepting only systematic-form matrices—i.e., $(0, 0)$-semi-systematic-form matrices—is the simplest possibility, making implementations as easy as possible to write and audit. One can argue that accepting more matrices produces a tighter security proof, but the original tightness loss was at most 2 bits. The primary argument for accepting more matrices is a performance argument, namely that this increases the success probability of each key-generation attempt.

Accepting *any* full-rank matrix maximizes the success probability. On the other hand, the analysis in [30] suggests that constant-time implementations of the first step will then be very slow. Presumably this means that the overall key-generation time will be slower on average, despite the improved success probability.

The concept of (μ, ν)-semi-systematic form is designed to take both the time and the success probability into account. Compared to $(\mu, \nu) = (0, 0)$, a small increase in μ and $\nu - \mu$ reduces and stabilizes the number of key-generation attempts. It is reasonable to estimate, for example, that $(\mu, \nu) = (32, 64)$ reduces the failure probability of each attempt below 2^{-30}, so most of the time one needs only 1 key-generation attempt. This attempt requires extra work for a constant-time echelon-form computation, but only within ν columns, which is not a large issue when ν is kept reasonably small.

Classic McEliece continues to include $(0, 0)$-semi-systematic-form computations for three reasons. First, the Classic McEliece software also speeds up those computations, skipping most of the work in Gaussian elimination in the failure cases and thus reducing the average key-generation time. Second, applications where key generation is not a bottleneck do not need the speedups from $(32, 64)$-semi-systematic form. Third, there is value in simplicity.

3 Indistinguishability against chosen-ciphertext attacks

Assume that McEliece's system is one-way. Niederreiter's system is then also one-way: the attacker, given Niederreiter's public key H and the ciphertext He for a uniform random weight-t vector e, cannot efficiently compute e.

What the user actually needs is more than one-wayness. The user is normally sending a plaintext with structure, perhaps a plaintext that can simply be guessed. Furthermore, the attacker can try modifying ciphertexts to see how the receiver reacts. McEliece's original PKE was not designed to resist, and does not resist, such attacks. In modern terminology, the user needs IND-CCA2 security.

There is a long literature studying the IND-CCA2 security of various PKE constructions, and in particular constructions built from an initial PKE assumed to have OW-CPA security. An increasingly popular simplification here is to encrypt the user's plaintext with an authenticated cipher such as AES-GCM. The public-key problem is then simply to send an unpredictable session key to use as the cipher key. Formally, the design goal here is to build a KEM with IND-CCA2 security; "KEM-DEM" composition [45] then produces a PKE with IND-CCA2 security, assuming a secure DEM. More complicated PKE constructions can pack some plaintext bytes into the ciphertext but are more difficult to audit and would be contrary to the Classic McEliece goal of producing high confidence in security.

The Classic McEliece KEM construction follows the best practices established in the literature:

- Classic McEliece uses a uniform random PKE input e, and computes the session key as a hash of e.
- After using the private key to compute e from a ciphertext, the Classic McEliece decapsulation algorithm checks that reencrypting e matches the ciphertext.
- If decryption fails (i.e., if computing e fails or reencryption does not match), Classic McEliece does not return a KEM failure: instead it returns a pseudorandom function of the ciphertext, specifically a cryptographic hash of a separate private key and the ciphertext.

Classic McEliece uses a standard, thoroughly studied cryptographic hash function, and ensures that the two hashes mentioned above are obtained by applying this function to input spaces that are visibly disjoint. The input details are chosen to simplify implementations that run in constant time, in particular not leaking whether decryption failed.

The fact that the underlying PKE is deterministic enables *tight* proofs of ROM IND-CCA2 security and even QROM IND-CCA2 security for this KEM construction, assuming one-wayness of the PKE. See the separate "guide for security reviewers" document. The fact that decryption always works for legitimate ciphertexts simplifies the proofs.

IND-CCA2 for encodings. Ciphertexts are normally encoded as byte strings and then further encoded as objects in higher-level protocols. Encodings in network protocols and in other applications are, in general, not unique: there are many objects that will decode to the same ciphertext.

Shoup [45] refers to this non-uniqueness property as "benign malleability". It is possible to construct applications where this property loses security. There is a debate in the literature as to whether this should be addressed

- in applications, by the rule of acting on encoded ciphertexts solely by decoding and decapsulating them, or
- in decoders, by the rule of enforcing unique encodings for ciphertexts.

A reencoding wrapper converts a deterministic decoder (and deterministic encoder) into a decoder following the second rule, by rejecting any encoded ciphertext $\underline{C}$ different from the encoding of the decoding of $\underline{C}$. A similar wrapper that rewrites ciphertexts, without rejecting them, converts an application into an application following the first rule.

Given any encoding, one can extend the definition of IND-CCA2 for ciphertexts into a definition of IND-CCA2 for encoded ciphertexts. The rule of enforcing unique encodings makes the second definition equivalent to the first. The rule of acting on encoded ciphertexts solely by decoding and decapsulating them makes the second definition unnecessary.

Encodings as byte strings are within the scope of Classic McEliece. To provide clarity for applications that want to enforce unique encodings as byte strings, the specification distinguishes between

- Narrowly Decoded Classic McEliece, which requires padding bits (not just for encoded ciphertexts but also for encoded public keys) to be 0 on decoding, and
- Simply Decoded Classic McEliece, which ignores padding bits on decoding.

A wrapper that requires the padding bits to be 0 converts an implementation of Simply Decoded Classic McEliece into an implementation of Narrowly Decoded Classic McEliece. A wrapper that clears the padding bits converts an implementation of Narrowly Decoded Classic McEliece into an implementation of Simply Decoded Classic McEliece.

Simply Decoded Classic McEliece and Narrowly Decoded Classic McEliece are equivalent for the selected non-`6960` parameter sets, since only the `6960` parameter sets use padding bits. For the `6960` parameter sets, the Classic McEliece software implements Narrowly Decoded Classic McEliece by checking the appropriate bits of public keys and ciphertexts. This check is handled in constant time, for applications where "public" keys and ciphertexts are actually confidential (e.g., obtained as outputs of another layer of decryption). In case applications fail to check return values, the encapsulation software sets all bits to 0 in its ciphertext and session-key output buffers in case of bad padding, and the decapsulation software sets all bits to 1 in its session-key output buffer in case of bad padding. The difference between 0 and 1 here is designed so that a cascade of several possible failures (bad padding in a public key, ignoring the encapsulation failure, bad padding in a ciphertext, and ignoring the decapsulation failure) will produce two different session keys that will not interoperate in typical DEMs, increasing the chance of the failures being caught by tests.

Security goals beyond IND-CCA2. The literature contains many other extensions of IND-CCA2. There are arguments that the extended properties are easy for cryptosystems to achieve and can protect applications. There are counterarguments saying that the same security can be achieved in a simpler way by another layer of the system. Consider, e.g., Shoup's argument in [45, Section 3.3] that KEMs should avoid hashing "labels" such as identities since "it is easier to implement labels in the data encapsulation mechanism".

Classic McEliece follows the principle that any generic transformation aiming at a goal beyond IND-CCA2 is out of scope for a KEM specification. Factoring the transformation out of KEM specifications simplifies the cryptographic ecosystem, making design and review easier, because the transformation is modularized instead of being handled redundantly by each cryptosystem. Each component is simpler, without any change in the composition provided to the end user.

One could extend the principle to say that generic conversions from from OW-CPA to IND-CCA2 should be factored out of cryptosystem specifications. Note, however, that the Classic McEliece conversion from OW-CPA to IND-CCA2 is *not* generic Fujisaki–Okamoto: it uses cryptosystem-specific modifications for extra efficiency, in particular avoiding any need to store the public key inside the private key. The output is always the same, so IND-CCA2 security is guaranteed to be the same, but the data flow and specification are different.

Here are examples of obvious properties of Classic McEliece that have no effect on the IND-CCA2 security goal but that can affect other goals: private keys are malleable (because of padding bits, multiple sequences of control bits that represent the same permutation, unused α values for $n < q$, etc.); modifying one bit in a public key has a significant chance of not affecting any particular ciphertext; various linear-algebra operations on public keys have predictable effects on ciphertexts; the first $n - k$ bits of a plaintext for the internal PKE are simply added to the ciphertext. Application designers are encouraged to assume solely the standard IND-CCA2 property, and in any case to be clear regarding the properties that they assume.

4 Generation of random objects

A widely deployed RSA prime-generation algorithm was broken by ROCA [40]. Nothing was shown to be wrong with the underlying source of random bytes, or the primality of each output p, or the interval containing p, or the entropy of p, namely 256 bits. The problem was that, for efficiency, the algorithm generated primes with a special structure that could be exploited by the attacker.

This algorithm was compliant with RSA specifications that asked for "random" primes p in an interval. RSA specifications that asked for "uniform random" primes p in an interval would have prohibited this algorithm, but also would have prohibited almost all RSA implementations: any PRNG converting (say) 256 bits of entropy into a 1024-bit prime p is producing a non-uniform output distribution.

PRNGs are good for testability, so standards should allow cryptographic modules to generate a prime p from a PRNG. How does the cryptographic module validator assess whether the prime p is sufficiently random?

If RSA security reviewers have considered uniform random private keys (p, q), then there is no loss of security from any distribution of (p, q) that is indistinguishable from uniform. It therefore suffices for the validator to ask whether the algorithm to generate (p, q) is generating a distribution indistinguishable from uniform. A weaker divergence property (see, e.g., [5]) suffices for "search" security properties such as signature security and OW-CPA.

The standard PRNG objective is to generate a byte string indistinguishable from a uniform random byte string. This is analogous to generating primes indistinguishable from uniform random primes, but it is not the same. To close the gap, some standards specify algorithms to deterministically convert byte strings into private keys (p, q); see, e.g., [39, Appendix B.3.3]. The security goal for such an algorithm is to convert a uniform random byte string into (p, q) indistinguishable from uniform. If the algorithm meets this goal, and is applied to a byte string indistinguishable from uniform, then it produces (p, q) indistinguishable from uniform. A weaker divergence property again suffices for some applications.

This structure means that there is a process of specifying, reviewing, and approving algorithms to generate RSA keys: not just tests for primality, but complete algorithms to convert random bytes into private keys. The cryptographic module validator checks whether the implementation is using an approved private-key-generation algorithm and an approved source of random bytes.

4.1 Random objects in Classic McEliece

The same issues arise in post-quantum cryptography. In particular, key generation in the Model Classic McEliece KEM (see the separate "guide for security reviewers" document), defined using ModelKeyGen, asks for a uniform random sequence $(\alpha_0, \ldots, \alpha_{n-1})$ of n distinct elements of $\mathbb{F}_q$, and a uniform random monic irreducible polynomial g of degree t. Even if an implementation is using an approved source of random bytes, how does the cryptographic module validator assess whether an implementation is generating a sufficiently random sequence $(\alpha_0, \ldots, \alpha_{n-1})$ and a sufficiently random g?

To answer these questions, the specification defines deterministic algorithms Irreducible and FieldOrdering designed to convert uniform random byte strings into sufficiently random g and $(\alpha_0, \ldots, \alpha_{q-1})$ respectively, and on top of this defines a deterministic algorithm SeededKeyGen that converts a 32-byte seed into a private key. KeyGen for the Classic McEliece KEM is then defined to apply SeededKeyGen to a uniform random 32-byte seed.

Specifying ModelKeyGen gives a simple definition of the Model Classic McEliece KEM for review of the IND-CCA2 security property. A separate review of the relationship between KeyGen and ModelKeyGen then transports the IND-CCA2 security property from the Model Classic McEliece KEM to the Classic McEliece KEM. The cryptographic module validator then checks that an implementation is correctly implementing SeededKeyGen and is starting from an approved source of 32 random bytes for KeyGen.

This structure is compatible with specifying, reviewing, and approving future alternatives to SeededKeyGen, for example because performance analysis finds faster secure key-generation methods.

4.2 Compression of private keys

Classic McEliece private keys are much smaller than public keys, but there may be interest in compressing them further.

The KeyGen structure explained above, deterministically mapping a 32-byte seed to a private key, implies that a private key can be compressed to these 32 bytes. Uncompression then means running SeededKeyGen again. Various details of SeededKeyGen, and of the private-key format, are designed to support a slightly different compression mechanism for which uncompression is much faster than key generation.

The main bottleneck in key generation is reducing a parity-check matrix to systematic form (or semi-systematic form), and starting over with a new key-generation attempt if the matrix reduction fails. However, as explained earlier, the resulting public key is not needed for decapsulation. The data flow from the matrix reduction to the private key consists solely of (1) knowing whether $(g, \alpha_0, \ldots, \alpha_{n-1})$ has been rejected and (2) a permutation of $(\alpha_0, \ldots, \alpha_{n-1})$ into $(\alpha'_0, \ldots, \alpha'_{n-1})$ for the generalization to semi-systematic form.

SeededKeyGen starts with a seed δ and deterministically maps δ to $(g, \alpha_0, \ldots, \alpha_{n-1}, \delta')$. If $(g, \alpha_0, \ldots, \alpha_{n-1})$ is rejected, SeededKeyGen replaces δ with δ' and starts over. This structure supports a compression mechanism that stores the *final* seed, a seed known to pass the rejection-sampling process, rather than the initial seed. Uncompression then simply maps the final seed δ to the final $(g, \alpha_0, \ldots, \alpha_{n-1})$, without any matrix operations.

For the generalization to semi-systematic form, uncompression needs to compute $(\alpha'_0, \ldots, \alpha'_{n-1})$, not just $(\alpha_0, \ldots, \alpha_{n-1})$. The permutation of $(\alpha_0, \ldots, \alpha_{n-1})$ into $(\alpha'_0, \ldots, \alpha'_{n-1})$ is fully specified by a ν-bit string of weight μ encoding $c = (c_{mt-\mu}, \ldots, c_{mt-1})$: e.g., a 64-bit string of weight 32 when $(\mu, \nu) = (32, 64)$.

The objects $(\delta, c, g, \alpha, s)$ are organized in a private key so that four natural compression mechanisms each consist of simple truncation:

- Truncation to (δ, c, g, α) saves $\lceil n/8 \rceil$ bytes in the private key, and regenerating s from δ requires simply (the first) $\lceil n/8 \rceil$ bytes of SHAKE256 output.
- Truncation to (δ, c, g) requires more work to regenerate α but saves much more space.
- Truncation to (δ, c) requires a minimal-polynomial computation to regenerate g but compresses to just 40 bytes.
- Truncation to the 32-byte seed δ suffices for systematic form.

One could encode c as 0 bytes for systematic form. The specified encoding instead uses 8 bytes so that a systematic-form private key can also be used by implementations that expect a semi-systematic-form private key. This avoids the need for key-format specifications to distinguish systematic form from semi-systematic form when all other parameters are the same: systematic-form private keys are simply the special case of semi-systematic-form private keys in which these 8 bytes are $(255, 255, 255, 255, 0, 0, 0, 0)$.

The private key is specified to record an ordering $(\alpha'_0, \ldots, \alpha'_{n-1}, \alpha_n, \ldots, \alpha_{q-1})$ of the field $\mathbb{F}_q$

as a sequence of control bits for a Beneš network. Most of the selected parameters have $n < q$, and no use is made of $(\alpha_n, \ldots, \alpha_{q-1})$, so another way to save space is to list just $(\alpha'_0, \ldots, \alpha'_{n-1})$. One can apply the corresponding permutation by sorting, which is slower than a Beneš network but avoids the need to compute control bits. One can also apply the corresponding permutation through RAM lookups, but implementors are cautioned that this leaks information through timing on many platforms. Specifying control bits as the default representation of α has the advantage of encouraging constant-time implementations. All implementations should be reviewed for timing leaks and other applicable side-channel leaks in any case.

4.3 Field ordering

The FieldOrdering algorithm interprets a uniform random $4q$-byte input string as a uniform random sequence of 32-bit integers $a_0, a_1, \ldots, a_{q-1}$. This sequence is rejected if and only if it contains fewer than q distinct elements. The sequence is accepted with probability $(1-1/2^{32})(1-2/2^{32})\cdots(1-(q-1)/2^{32})$, which is more than 0.99 if $q \le 2^{13}$, and more than 0.6 if $q \le 2^{16}$. (This description focuses on the choice $\sigma_2 = 32$. Parameters with $q > 2^{16}$ would take more than 32 bits in each integer by definition of σ_2, so the acceptance probability would still be more than 0.6.)

An accepted sequence is a uniform random sequence of distinct 32-bit integers $a_0, a_1, \ldots, a_{q-1}$. There is then a unique permutation that sorts $(a_0, a_1, \ldots, a_{q-1})$, and the output is the same permutation applied to an initial ordering of $\mathbb{F}_q$. The permutation is a uniform random permutation, so the output is a uniform random ordering of $\mathbb{F}_q$.

Omitting the rejection would produce a permutation distinguishable from uniform, but would still suffice for almost exactly the same OW-CPA security level by a divergence argument. See [5].

4.4 Irreducible polynomial

The Irreducible algorithm extracts mt input bits from a uniform random $2t$-byte input string, and interprets the mt bits as a uniform random element β of $\mathbb{F}_q[y]/F(y)$. The algorithm returns the minimal polynomial g of β over $\mathbb{F}_q$ if g has degree t; otherwise it fails. (This description focuses on the choice $\sigma_1 = 16$, with $m \le 16$.)

Any particular monic irreducible degree-t polynomial g has exactly t roots in $\mathbb{F}_q[y]/F(y)$, and is thus found by exactly t of the 2^{mt} possibilities for β, i.e., exactly $t2^{16t-mt}$ of the possibilities for the $16t$-bit input string. The distribution of g is thus uniform. Uniformity is again overkill here: having low divergence would suffice.

Well-known formulas for the number of irreducible polynomials (equivalently, the observation that the algorithm fails exactly when β is in a proper subfield of $\mathbb{F}_q[y]/F(y)$) imply that this algorithm succeeds with probability more than 99% when $t \ge 16$.

4.5 Fixed-weight vector in encapsulation

ModelEncap begins by generating a uniform random n-bit vector of weight t. This again raises a question for cryptographic module validators regarding how these vectors are generated. Encap instead calls FixedWeight, which calls a traditional RNG that produces a stream of bits.

Applications might plug in an RNG that generates output from a series of seeds as in key generation, allowing a ciphertext to be compressed to the final seed. See, e.g., [8]. The same compression approach works for rejection sampling in much more generality.

FixedWeight generates a uniform random n-bit vector $e = (e_0, e_1, \ldots, e_{n-1})$ of weight t by generating a uniform random sequence $(a_0, a_1, \ldots, a_{t-1})$ of t distinct integers in $\{0, 1, \ldots, n-1\}$, and using those integers as the support of e.

FixedWeight generates this uniform random sequence by generating a uniform random sequence $(a_0, a_1, \ldots, a_{t-1})$ of integers in $\{0, 1, \ldots, n-1\}$, and then starting over if the integers are not distinct. Each try succeeds with probability $(1 - 1/n)(1 - 2/n) \cdots (1 - (t-1)/n)$, which is above $1/4$ for each of the selected parameter sets. (The alternative e-generation method in [5] guarantees its run time, but FixedWeight is essentially always faster.)

Generating a uniform random sequence $(a_0, a_1, \ldots, a_{t-1})$ of integers in $\{0, 1, \ldots, q-1\}$ is a simple matter of collecting uniform random bits. For $n < q$, one well-known way to generate a uniform random stream of integers in $\{0, 1, \ldots, n-1\}$ is by rejection sampling on a uniform random stream of integers in $\{0, 1, \ldots, q-1\}$. If n is below $q/2$ then it is more efficient to begin with integers in $\{0, 1, \ldots, q/2 - 1\}$, and similar comments apply if n is below $q/4$ etc., but FixedWeight skips these refinements since all of the selected parameters have $n > q/2$.

FixedWeight uses a batch of τ integers in $\{0, 1, \ldots, q-1\}$, and applies rejection sampling to generate a batch of integers in $\{0, 1, \ldots, n-1\}$, say u integers, where u is between 0 and τ. For each u, the integers in $\{0, 1, \ldots, n-1\}$ are uniform. Consequently, if $u \geq t$, the first t integers in $\{0, 1, \ldots, n-1\}$ are uniform as desired. Batch processing simplifies parallelization and vectorization, and some standard RNGs are much more efficient at generating a large batch of random bits than at generating the same volume of data in small chunks.

The probability of $u \geq t$ is the sum of the coefficients of $x^t, x^{t+1}, \ldots, x^\tau$ in $(1 - n/q + (n/q)x)^\tau$. This probability is above 0.96 for $(m, n, t) = (13, 4608, 96)$, and much closer to 1 for the other selected parameters.

There would be a slight savings in time from reducing τ below $2t$. With smaller batch sizes it would also save time to handle the case $u < t$ differently: instead of discarding the u integers already found in $\{0, 1, \ldots, n-1\}$, keep those integers and use the next batch to extend the list of integers. This would reduce the number of iterations required, at the expense of tracking state between iterations. As long as the selection process sees only whether integers are in $\{0, 1, \ldots, n-1\}$ or not, the resulting integers in $\{0, 1, \ldots, n-1\}$ are uniform.

5 Selected parameter sets

The Classic McEliece parameter space naturally allows a wide range of choices of the field size q, the code length n, and the number of errors t. This raises the question of how to choose parameters.

5.1 Maximizing security within 1MB

The selected Classic McEliece parameters have always been chosen to maximize security subject to size constraints.

This implies, as a preliminary matter, the traditional choice of $m = \log_2 q$ as $\lceil \log_2 n \rceil$. Then n and t dictate the code dimension (namely $k = n - mt$; Goppa codes of other dimensions can occur, but are extremely rare, and are rejected by the key-generation process), the number of ciphertext bits (namely $n - k = mt$), and the number of public-key bits (namely $k(n - k) = kmt$), along with security levels.

In particular, the `6960119` parameter set, with $(n, t) = (6960, 119)$, is chosen to maximize security for public keys fitting into 2^{20} bytes. The choice $n = 6960$ goes back to [9, Section 7], which documents this choice as maximizing security subject to the 2^{20}-byte size limit:

> For keys limited to $2^{16}, 2^{17}, 2^{18}, 2^{19}, 2^{20}$ bytes, we propose Goppa codes of lengths $1744, 2480, 3408, 4624, 6960$ and degrees $35, 45, 67, 95, 119$ respectively, with $36, 46, 68, 97, 121$ errors added by the sender. These codes achieve security levels $84.88, 107.41, 147.94, 191.18, 266.94$ against our attack. In general, for any particular limit on public-key size, codes of rate approximately 0.75 appear to maximize the difficulty of our attack.

The only difference between the choice in [9] and the specified `6960119` parameter set is that Classic McEliece adds 119 errors instead of 121 errors; this is structurally required because Classic McEliece does not include the complications of "list decoding". This marginally reduces the security level.

5.2 The robustness of maximizing security

Choosing parameter sets to maximize security subject to a specified size constraint is much more robust than choosing parameter sets to minimize size for a specific target security level.

The issue here is that there are variations in exactly how the literature measures security, partly because of small attack improvements over many years and partly because of different models for the costs of operations inside attacks. See, e.g., Table 1 in the separate "guide for security reviewers" document. A parameter set chosen to exactly match a particular security level in one metric is likely to be overkill in another metric and too small in a third metric.

These variations turn out to have much less impact on parameter sets chosen to maximize security level for a specified size. Occasionally two nearby parameter sets have very close security levels and will vary in ordering depending on exactly which metric is chosen, but the closeness also means that the exact choice between these parameter sets does not matter for the user.

5.3 Maximizing security within other size constraints

The nearby parameter set `6688128` is also chosen to maximize security for public keys fitting into 2^{20} bytes, but with the following extra size constraints: n is a multiple of 32, and t is a multiple of 32.

Requiring multiples of 32 generally produces worse tradeoffs between size and security in various metrics, but only slightly worse, as illustrated by [30, Figure 11.1]. Meanwhile this slightly simplifies implementations. In particular, there are no padding bits in the encodings of public keys and ciphertexts, so IND-CCA2 for ciphertexts automatically implies IND-CCA2 for encoded ciphertexts; the distinction between Simply Decoded Classic McEliece and Narrowly Decoded Classic McEliece (see Section 3) disappears.

The larger `8192128` parameter set puts heavier constraints on the sizes n and t, both being powers of 2, limiting the available choices of security levels and key sizes. One can argue that n should always be taken as q, a power of 2, since this is what McEliece's original cryptosystem did; however, the safety of $n < q$ can be deduced from the safety of $n = q$. See "OW-CPA security of length below field size" in the separate "guide for security reviewers" document.

The smaller parameter sets `348864` and `460896` were chosen the same way as `6688128`, except with key-size limits of 2^{18} bytes and 2^{19} bytes respectively instead of 2^{20} bytes. These three parameter sets were introduced in 2019 in [7], which documented the choice of these parameter sets as maximizing security subject to these size limits. To quote [7]:

> We are therefore expanding the list of parameter sets (n, k, t) as follows:
>
> - $(8192, 6528, 128)$, 240-byte ciphertexts: taking both n and t to be powers of 2; as in round-1 submission.
> - $(6960, 5413, 119)$, 226-byte ciphertexts: optimal security within 2^{20} bytes for public key; as in round-1 submission.
> - $(6688, 5024, 128)$, 240-byte ciphertexts: optimal security within 2^{20} bytes if n and t are required to be multiples of 32.
> - $(4608, 3360, 96)$, 188-byte ciphertexts: optimal security within 2^{19} bytes if n and t are required to be multiples of 32.
> - $(3488, 2720, 64)$, 128-byte ciphertexts: optimal security within 2^{18} bytes if n and t are required to be multiples of 32.

5.4 Possibilities for further parameter sets

The powers of 2 chosen above (e.g., 2^{20} bytes as a key-size limit, or t being a multiple of 32) are not required by the design of Classic McEliece. The primary argument for taking powers of 2 is simplicity. Even when there is no impact on implementations, a rule of preferring simpler parameter descriptions helps limit opportunities for attackers to manipulate parameter choices; see generally [6].

It would be straightforward to generate intermediate parameter sets: for example, parameter sets strictly between 2^{18} bytes and 2^{19} bytes for the corner case of an application that (1) can afford `348864`, (2) cannot afford `460896`, and (3) wants a higher security level than `348864`. However, the benefits of providing such options need to be weighed against the simplicity argument from the previous paragraph.

Another option is to choose parameters that maximize security for a ciphertext-size limit rather than for a key-size limit. This generally means taking smaller values of t, possibly in the range where [28] can distinguish public keys from uniform random matrices. This option would not necessarily damage IND-CCA2 security, but it would complicate the security analysis.

6 Advantages and limitations

The most important advantage of the McEliece system is security, covered throughout this document and the separate "guide for security reviewers" document.

Regarding efficiency, the use of random-looking linear codes with no visible structure forces public-key sizes to be on the scale of a megabyte for quantitatively high security: the public key is a full (generator/parity-check) matrix. Key-generation software is also not very fast. Applications must continue using each public key for long enough to handle the costs of generating and distributing the key.

There are, however, some compensating efficiency advantages. Encapsulation takes a single pass over a public key, so one can stream public keys through tiny coprocessors and tiny devices, taking advantage of the simple nature of the objects (binary vectors) and operations (binary dot products). Encapsulation and decapsulation are reasonably fast in software, and impressively fast in hardware. See the separate "guide for implementors" document for speeds of all operations.

Furthermore, the ciphertexts are unusually small for post-quantum cryptography: under 256 bytes for the selected high-security parameter sets. This allows ciphertexts to fit comfortably inside single network packets. In applications that transmit enough ciphertexts per key, the disadvantage of large public keys is outweighed by the advantage of small ciphertexts: Classic McEliece has the lowest per-ciphertext cost of any post-quantum system.

References

[1] Marco Baldi, Franco Chiaraluce, Roberto Garello, and Francesco Mininni. Quasi-cyclic low-density parity-check codes in the McEliece cryptosystem. In *Proceedings of IEEE International Conference on Communications, ICC 2007, Glasgow, Scotland, 24-28 June 2007*, pages 951–956. IEEE, 2007. https://doi.org/10.1109/ICC.2007.161.

[2] Anja Becker, Antoine Joux, Alexander May, and Alexander Meurer. Decoding random binary linear codes in $2^{n/20}$: How $1 + 1 = 0$ improves information set decoding. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology—EUROCRYPT 2012—31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15–19, 2012. Proceedings*, volume 7237 of *Lecture Notes in Computer Science*, pages 520–536. Springer, 2012. https://www.cits.ruhr-uni-bochum.de/imperia/md/content/may/paper/isd-extended.pdf.

[3] Thierry P. Berger, Pierre-Louis Cayrel, Philippe Gaborit, and Ayoub Otmani. Reducing key length of the McEliece cryptosystem. In Bart Preneel, editor, *Progress in Cryptology—AFRICACRYPT 2009, Second International Conference on Cryptology in Africa, Gammarth, Tunisia, June 21-25, 2009. Proceedings*, volume 5580 of *Lecture Notes in Computer Science*, pages 77–97. Springer, 2009. https://hal.archives-ouvertes.fr/hal-01081727/file/ACTI-BERGER-2009-2.pdf.

[4] Daniel J. Bernstein. Grover vs. McEliece. In Nicolas Sendrier, editor, *Post-Quantum Cryptography, Third International Workshop, PQCrypto 2010, Darmstadt, Germany, May 25–28, 2010. Proceedings*, volume 6061 of *Lecture Notes in Computer Science*, pages 73–80. Springer, 2010. https://cr.yp.to/papers.html#grovercode.

[5] Daniel J. Bernstein. Divergence bounds for random fixed-weight vectors obtained by sorting, 2018. https://cr.yp.to/papers.html#divergence.

[6] Daniel J. Bernstein, Tung Chou, Chitchanok Chuengsatiansup, Andreas Hülsing, Eran Lambooij, Tanja Lange, Ruben Niederhagen, and Christine van Vredendaal. How to manipulate curve standards: A white paper for the black hat. In Liqun Chen and Shin'ichiro Matsuo, editors, *Security Standardisation Research—Second International Conference, SSR 2015, Tokyo, Japan, December 15–16, 2015, Proceedings*, volume 9497 of *Lecture Notes in Computer Science*, pages 109–139. Springer, 2015. https://bada55.cr.yp.to.

[7] Daniel J. Bernstein, Tung Chou, Tanja Lange, Ingo von Maurich, Rafael Misoczki, Ruben Niederhagen, Edoardo Persichetti, Christiane Peters, Peter Schwabe, Nicolas Sendrier, Jakub Szefer, and Wen Wang. Classic McEliece: conservative code-based cryptography: modifications for round 2, 2019. https://classic.mceliece.org/nist/mceliece-20190331-mods.pdf.

[8] Daniel J. Bernstein and Tanja Lange. McTiny: Fast high-confidence post-quantum key erasure for tiny network servers. In Srdjan Capkun and Franziska Roesner, editors,

29th USENIX Security Symposium, USENIX Security 2020, August 12–14, 2020, pages 1731–1748. USENIX Association, 2020. https://mctiny.org.

[9] Daniel J. Bernstein, Tanja Lange, and Christiane Peters. Attacking and defending the McEliece cryptosystem. In Johannes A. Buchmann and Jintai Ding, editors, *Post-Quantum Cryptography, Second International Workshop, PQCrypto 2008, Cincinnati, OH, USA, October 17–19, 2008, Proceedings*, volume 5299 of *Lecture Notes in Computer Science*, pages 31–46. Springer, 2008. https://eprint.iacr.org/2008/318.

[10] Daniel J. Bernstein, Tanja Lange, and Christiane Peters. Smaller decoding exponents: Ball-collision decoding. In Phillip Rogaway, editor, *Advances in Cryptology—CRYPTO 2011—31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14–18, 2011. Proceedings*, volume 6841 of *Lecture Notes in Computer Science*, pages 743–760. Springer, 2011. https://eprint.iacr.org/2010/585.

[11] Daniel J. Bernstein, Tanja Lange, Christiane Peters, and Henk C. A. van Tilborg. Explicit bounds for generic decoding algorithms for code-based cryptography. In *Pre-proceedings of WCC 2009*, pages 168–180, 2009.

[12] Leif Both and Alexander May. Optimizing BJMM with nearest neighbors: Full decoding in $2^{2n/21}$ and McEliece security, 2017. International Workshop on Coding and Cryptography (WCC 2017). https://www.cits.ruhr-uni-bochum.de/imperia/md/content/may/paper/bjmm+.pdf.

[13] Leif Both and Alexander May. Decoding linear codes with high error rate and its impact for LPN security. In Tanja Lange and Rainer Steinwandt, editors, *Post-Quantum Cryptography—9th International Conference, PQCrypto 2018, Fort Lauderdale, FL, USA, April 9–11, 2018, Proceedings*, volume 10786 of *Lecture Notes in Computer Science*, pages 25–46. Springer, 2018. https://eprint.iacr.org/2017/1139.

[14] Anne Canteaut and Herve Chabanne. A further improvement of the work factor in an attempt at breaking McEliece's cryptosystem. In Pascale Charpin, editor, *Livre des résumés—EUROCODE 94, Abbaye de la Bussière sur Ouche, France, October 1994*, 1994. https://hal.inria.fr/inria-00074443.

[15] Anne Canteaut and Florent Chabaud. A new algorithm for finding minimum-weight words in a linear code: Application to McEliece's cryptosystem and to narrow-sense BCH codes of length 511. *IEEE Trans. Information Theory*, 44(1):367–378, 1998. https://www.rocq.inria.fr/secret/Anne.Canteaut/Publications/Canteaut_Chabaud98.pdf.

[16] Anne Canteaut and Nicolas Sendrier. Cryptanalysis of the original McEliece cryptosystem. In Kazuo Ohta and Dingyi Pei, editors, *Advances in Cryptology—ASIACRYPT '98, International Conference on the Theory and Applications of Cryptology and Information Security, Beijing, China, October 18–22, 1998, Proceedings*, volume 1514 of *Lecture Notes in Computer Science*, pages 187–199. Springer, 1998. https://www.rocq.inria.fr/secret/Anne.Canteaut/Publications/Canteaut_Sendrier98.pdf.

[17] Kevin Carrier, Thomas Debris-Alazard, Charles Meyer-Hilfiger, and Jean-Pierre Tillich. Statistical decoding 2.0: Reducing decoding to LPN. *CoRR*, abs/2208.02201, 2022. https://doi.org/10.48550/arXiv.2208.02201.

[18] Herve Chabanne and Bernard Courteau. Application de la méthode de décodage itérative d'Omura à la cryptanalyse du système de McEliece, 1993. Université de Sherbrooke, Rapport de Recherche, number 122.

[19] Florent Chabaud. Asymptotic analysis of probabilistic algorithms for finding short codewords. In Paul Camion, Pascale Charpin, and Sami Harari, editors, *Eurocode '92: proceedings of the international symposium on coding theory and applications held in Udine, October 23–30, 1992*, pages 175–183. Springer, 1993.

[20] George C. Clark, Jr. and J. Bibb Cain. *Error-correcting coding for digital communication*. Plenum, 1981.

[21] John T. Coffey and Rodney M. Goodman. The complexity of information set decoding. *IEEE Transactions on Information Theory*, 35:1031–1037, 1990.

[22] John T. Coffey, Rodney M. Goodman, and P. Farrell. New approaches to reduced complexity decoding. *Discrete and Applied Mathematics*, 33:43–60, 1991. https://core.ac.uk/reader/81155220.

[23] Alain Couvreur, Ayoub Otmani, and Jean-Pierre Tillich. Polynomial time attack on Wild McEliece over quadratic extensions. *IEEE Trans. Information Theory*, 63(1):404–427, 2017. https://eprint.iacr.org/2014/112.

[24] Thomas Debris-Alazard, Léo Ducas, and Wessel P. J. van Woerden. An algorithmic reduction theory for binary codes: LLL and more. *IEEE Trans. Inf. Theory*, 68(5):3426–3444, 2022. https://eprint.iacr.org/2020/869.

[25] Ilya I. Dumer. Two decoding algorithms for linear codes. *Problemy Peredachi Informatsii*, 25:24–32, 1989. http://www.mathnet.ru/eng/ppi635.

[26] Ilya I. Dumer. On minimum distance decoding of linear codes. In Grigori A. Kabatianskii, editor, *Fifth joint Soviet-Swedish international workshop on information theory, Moscow, 1991*, pages 50–52, 1991.

[27] Andre Esser, Alexander May, and Floyd Zweydinger. McEliece needs a break—solving McEliece-1284 and Quasi-Cyclic-2918 with modern ISD. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology—EUROCRYPT 2022—41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30–June 3, 2022, Proceedings, Part III*, volume 13277 of *Lecture Notes in Computer Science*, pages 433–457. Springer, 2022. https://eprint.iacr.org/2021/1634.

[28] Jean-Charles Faugère, Valérie Gauthier-Umaña, Ayoub Otmani, Ludovic Perret, and Jean-Pierre Tillich. A distinguisher for high-rate McEliece cryptosystems. *IEEE Trans. Information Theory*, 59(10):6830–6844, 2013. https://eprint.iacr.org/2010/331.

[29] Matthieu Finiasz and Nicolas Sendrier. Security bounds for the design of code-based cryptosystems. In Mitsuru Matsui, editor, *Advances in Cryptology—ASIACRYPT 2009, 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6–10, 2009. Proceedings*, volume 5912 of *Lecture Notes in Computer Science*, pages 88–105. Springer, 2009. https://eprint.iacr.org/2009/414.

[30] Classic McEliece Comparison Task Force. Classic McEliece vs. NTS-KEM. 2018. https://classic.mceliece.org/nist/vsntskem-20180629.pdf.

[31] Yann Hamdaoui and Nicolas Sendrier. A non asymptotic analysis of information set decoding, 2013. https://eprint.iacr.org/2013/162.

[32] Evgueni A. Krouk. Decoding complexity bound for linear block codes. *Problemy Peredachi Informatsii*, 25:103–107, 1989. http://www.mathnet.ru/eng/ppi665.

[33] Pil Joong Lee and Ernest F. Brickell. An observation on the security of McEliece's public-key cryptosystem. In Christoph G. Günther, editor, *Advances in Cryptology—EUROCRYPT '88, Workshop on the Theory and Application of of Cryptographic Techniques, Davos, Switzerland, May 25-27, 1988, Proceedings*, volume 330 of *Lecture Notes in Computer Science*, pages 275–280. Springer, 1988. https://doi.org/10.1007/3-540-45961-8_25.

[34] Jeffrey S. Leon. A probabilistic algorithm for computing minimum weights of large error-correcting codes. *IEEE Trans. Information Theory*, 34(5):1354–1359, 1988. https://doi.org/10.1109/18.21270.

[35] Alexander May, Alexander Meurer, and Enrico Thomae. Decoding random linear codes in $\tilde{\mathcal{O}}(2^{0.054n})$. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology—ASIACRYPT 2011—17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4–8, 2011. Proceedings*, volume 7073 of *Lecture Notes in Computer Science*, pages 107–124. Springer, 2011. https://www.cits.ruhr-uni-bochum.de/imperia/md/content/may/paper/decoding.pdf.

[36] Alexander May and Ilya Ozerov. On computing nearest neighbors with applications to decoding of binary linear codes. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology—EUROCRYPT 2015—34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26–30, 2015, Proceedings, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 203–228. Springer, 2015. https://www.cits.ruhr-uni-bochum.de/imperia/md/content/may/paper/codes.pdf.

[37] Rafael Misoczki and Paulo S. L. M. Barreto. Compact McEliece keys from Goppa codes. In Michael J. Jacobson Jr., Vincent Rijmen, and Rei Safavi-Naini, editors, *Selected Areas in Cryptography*, volume 5867 of *Lecture Notes in Computer Science*, pages 376–392. Springer, 2009. https://eprint.iacr.org/2009/187.

[38] Rafael Misoczki, Jean-Pierre Tillich, Nicolas Sendrier, and Paulo S. L. M. Barreto. MDPC-McEliece: New McEliece variants from moderate density parity-check codes. In *Proceedings of the 2013 IEEE International Symposium on Information Theory, Istanbul, Turkey, July 7–12, 2013*, pages 2069–2073. IEEE, 2013. https://eprint.iacr.org/2012/409.

[39] National Institute for Standards and Technology (NIST). Digital signature standard (DSS) FIPS 186–4, 2013. http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf.

[40] Matús Nemec, Marek Sýs, Petr Svenda, Dusan Klinec, and Vashek Matyas. The return of Coppersmith's attack: Practical factorization of widely used RSA moduli. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 1631–1648. ACM, 2017. https://crocs.fi.muni.cz/public/papers/rsa_ccs17.

[41] Harald Niederreiter. Knapsack-type cryptosystems and algebraic coding theory. *Problems of Control and Information Theory*, 15(2):159–166, 1986.

[42] Edoardo Persichetti. Compact McEliece keys based on quasi-dyadic Srivastava codes. *J. Mathematical Cryptology*, 6(2):149–169, 2012. https://eprint.iacr.org/2011/179.

[43] Eugene Prange. The use of information sets in decoding cyclic codes. *IRE Transactions on Information Theory*, IT-8:S5–S9, 1962.

[44] Nicolas Sendrier. Finding the permutation between equivalent linear codes: The support splitting algorithm. *IEEE Trans. Information Theory*, 46(4):1193–1203, 2000. https://doi.org/10.1109/18.850662.

[45] Victor Shoup. A proposal for an ISO standard for public key encryption, 2001. https://eprint.iacr.org/2001/112.

[46] Jacques Stern. A method for finding codewords of small weight. In Gérard D. Cohen and Jacques Wolfmann, editors, *Coding Theory and Applications, 3rd International Colloquium, Toulon, France, November 2–4, 1988, Proceedings*, volume 388 of *Lecture Notes in Computer Science*, pages 106–113. Springer, 1988. https://doi.org/10.1007/BFb0019850.

[47] Rodolfo Canto Torres and Nicolas Sendrier. Analysis of information set decoding for a sub-linear error weight. In Tsuyoshi Takagi, editor, *Post-Quantum Cryptography—7th International Workshop, PQCrypto 2016, Fukuoka, Japan, February 24–26, 2016, Proceedings*, volume 9606 of *Lecture Notes in Computer Science*, pages 144–161. Springer, 2016. https://hal.inria.fr/hal-01244886v1/document.

[48] Johan van Tilburg. On the McEliece public-key cryptosystem. In Shafi Goldwasser, editor, *Advances in Cryptology—CRYPTO '88, 8th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21–25, 1988, Proceedings*, volume

403 of *Lecture Notes in Computer Science*, pages 119–131. Springer, 1988. https://doi.org/10.1007/0-387-34799-2_10.

[49] Johan van Tilburg. *Security-analysis of a class of cryptosystems based on linear error-correcting codes*. PhD thesis, Technische Universiteit Eindhoven, 1994.