Attacking and defending the McEliece cryptosystem

Daniel J. Bernstein[1], Tanja Lange[2], and Christiane Peters[2]

[1] Department of Mathematics, Statistics, and Computer Science (M/C 249)
University of Illinois at Chicago, Chicago, IL 60607–7045, USA
djb@cr.yp.to
[2] Department of Mathematics and Computer Science
Technische Universiteit Eindhoven, P.O. Box 513, 5600 MB Eindhoven, Netherlands
tanja@hyperelliptic.org, c.p.peters@tue.nl

Abstract. This paper presents several improvements to Stern's attack on the McEliece cryptosystem and achieves results considerably better than Canteaut et al. This paper shows that the system with the originally proposed parameters can be broken in just 1400 days by a single 2.4GHz Core 2 Quad CPU, or 7 days by a cluster of 200 CPUs. This attack has been implemented and is now in progress.

This paper proposes new parameters for the McEliece and Niederreiter cryptosystems achieving standard levels of security against all known attacks. The new parameters take account of the improved attack; the recent introduction of list decoding for binary Goppa codes; and the possibility of choosing code lengths that are not a power of 2. The resulting public-key sizes are considerably smaller than previous parameter choices for the same level of security.

Keywords: McEliece cryptosystem, Stern attack, minimal weight code word, list decoding binary Goppa codes, security analysis.

1 Introduction

The McEliece cryptosystem was proposed by McEliece in 1978 [10] and the original version, using Goppa codes, remains unbroken. Quantum computers do not seem to give any significant improvements in attacking code-based systems, beyond the generic improvements possible with Grover's algorithm, and so the McEliece encryption scheme is one of the interesting candidates for post-quantum cryptography.

A drawback of the system is the comparably large key size — in order to hide the well-structured and efficiently decodable Goppa code in the public key, the full generator matrix of the scrambled code needs to be published. Various

[*] Permanent ID of this document: 7868533f20f51f8d769be2aa464647c9. Date of this document: 2008.08.07. This work has been supported in part by the National Science Foundation under grant ITR–0716498.

attempts to reduce the key size have used other codes, most notably codes over larger fields instead of subfield codes; but breaks of variants of the McEliece system have left essentially only the original system as the strongest candidate.

The fastest known attacks on the original system are based on information set decoding as implemented by Canteaut and Chabaud [4] and analyzed in greater detail by Canteaut and Sendrier [5].

In this paper we reconsider attacks on the McEliece cryptosystem and present improvements to Stern's attack [17] (which predates the Canteaut–Chabaud attack) and demonstrate that our new attack outperforms any previous ones. The result is that an attack on the originally proposed parameters of the McEliece cryptosystem is feasible on a moderate computer cluster. Already Canteaut and Sendrier had pointed out that the system does not hold up to current security standards but no actual attack was done before. We have implemented our new method and expect results soon.

On the defense side our paper proposes new parameters for the McEliece cryptosystem, selected from a much wider range of parameters than have been analyzed before. The codes we suggest are also suitable for the Niederreiter cryptosystem [11], a variant of the McEliece cryptosystem. The new parameters are designed to minimize public-key size while achieving 80-bit, 128-bit, or 256-bit security against known attacks — and in particular our attack. (Of course, by a similar computation, we can find parameters that minimize costs other than key size.) These new parameters exploit the ability to choose code lengths that are not powers of 2. They also exploit a recently introduced list-decoding algorithm for binary Goppa codes — see [2]; list decoding allows senders to introduce more errors into ciphertexts, leading to higher security with the same key size, or alternatively the same security with lower key size.

2 Review of the McEliece cryptosystem

McEliece in [10] introduced a public-key cryptosystem based on error-correcting codes. The public key is a hidden generator matrix of a binary linear code of length n and dimension k with error-correcting capability t. McEliece suggested using classical binary Goppa codes. We will briefly describe the main properties of these codes before describing the set-up of the cryptosystem.

Linear codes. A *binary* $[n, k]$ *code* is a binary linear code of length n and dimension k, i.e., a k-dimensional subspace of $\mathbf{F}_2^n$. All codes considered in this paper are binary.

The *Hamming weight* of an element $\mathbf{c} \in \mathbf{F}_2^n$ is the number of nonzero entries of $\mathbf{c}$. The *minimum distance* of an $[n, k]$ code C with $k > 0$ is the smallest Hamming weight of any nonzero element of C.

A *generator matrix* of an $[n, k]$ code C is a $k \times n$ matrix G such that $C = \left\{\mathbf{x}G : \mathbf{x} \in \mathbf{F}_2^k\right\}$. A *parity-check matrix* of an $[n, k]$ code C is an $(n-k) \times n$ matrix H such that $C = \left\{\mathbf{c} \in \mathbf{F}_2^n : H\mathbf{c}^T = 0\right\}$. Here $\mathbf{c}^T$ means the transpose of $\mathbf{c}$; we view elements of $\mathbf{F}_2^n$ as $1 \times n$ matrices, so $\mathbf{c}^T$ is an $n \times 1$ matrix.

A *systematic generator matrix* of an $[n, k]$ code C is a generator matrix of the form $(I_k|Q)$ where I_k is the $k \times k$ identity matrix and Q is a $k \times (n-k)$ matrix. The matrix $H = (Q^T|I_{n-k})$ is then a parity-check matrix for C. There might not exist a systematic generator matrix for C, but there exists a systematic generator matrix for an equivalent code obtained by permuting columns of C.

The classical decoding problem is to find the closest codeword $\mathbf{x} \in C$ to a given $\mathbf{y} \in \mathbf{F}_2^n$, assuming that there is a unique closest codeword. Here *close* means that the difference has small Hamming weight. Uniqueness is guaranteed if there exists a codeword $\mathbf{x}$ whose distance from $\mathbf{y}$ is less than half the minimum distance of C.

Classical Goppa codes. Fix a finite field $\mathbf{F}_{2^d}$, a basis of $\mathbf{F}_{2^d}$ over $\mathbf{F}_2$, and a set of n distinct elements $\alpha_1, \ldots, \alpha_n$ in $\mathbf{F}_{2^d}$. Fix an irreducible polynomial $g \in \mathbf{F}_{2^d}[x]$ of degree t, where $2 \le t \le (n-1)/d$. Note that, like [15, page 151] and unlike [10], we do not require n to be as large as 2^d.

The Goppa code $\Gamma = \Gamma(\alpha_1, \ldots, \alpha_n, g)$ consists of all elements $\mathbf{c} = (c_1, \ldots, c_n)$ in $\mathbf{F}_2^n$ satisfying

$$\sum_{i=1}^{n} \frac{c_i}{x - \alpha_i} = 0 \qquad \text{in } \mathbf{F}_{2^d}[x]/g.$$

The dimension of Γ is at least $n - td$ and typically is exactly $n - td$. For cryptographic applications one assumes that the dimension is exactly $n - td$. The $td \times n$ matrix

$$H = \begin{pmatrix} 1/g(\alpha_1) & \cdots & 1/g(\alpha_n) \\ \alpha_1/g(\alpha_1) & \cdots & \alpha_n/g(\alpha_n) \\ \vdots & \ddots & \vdots \\ \alpha_1^{t-1}/g(\alpha_1) & \cdots & \alpha_n^{t-1}/g(\alpha_n) \end{pmatrix},$$

where each element of $\mathbf{F}_{2^d}$ is viewed as a column of d elements of $\mathbf{F}_2$ in the specified basis of $\mathbf{F}_{2^d}$, is a parity-check matrix of Γ.

The minimum distance of Γ is at least $2t + 1$. Patterson in [13] gave an efficient algorithm to correct t errors.

The McEliece cryptosystem. The *McEliece secret key* consists of an $n \times n$ permutation matrix P; a nonsingular $k \times k$ matrix S; and a generator matrix G for a Goppa code $\Gamma(\alpha_1, \ldots, \alpha_n, g)$ of dimension $k = n - td$. The sizes n, k, t are public system parameters, but $\alpha_1, \ldots, \alpha_n, g, P, S$ are randomly generated secrets. McEliece suggests in his original paper to choose a $[1024, 524]$ classical binary Goppa code Γ with irreducible polynomial g of degree $t = 50$.

The *McEliece public key* is the $k \times n$ matrix SGP.

McEliece encryption of a message $\mathbf{m}$ of length k: Compute $\mathbf{m}SGP$ and add a random error vector $\mathbf{e}$ of weight t and length n. Send $\mathbf{y} = \mathbf{m}SGP + \mathbf{e}$.

McEliece decryption: Compute $\mathbf{y}P^{-1} = \mathbf{m}SG + \mathbf{e}P^{-1}$. Note that $\mathbf{m}SG$ is a codeword in Γ, and that the permuted error vector $\mathbf{e}P^{-1}$ has weight t. Use Patterson's algorithm to find $\mathbf{m}S$ and thereby $\mathbf{m}$.

The Niederreiter cryptosystem. We also consider a variant of the McEliece cryptosystem published by Niederreiter in [11]. Niederreiter's system, with the same Goppa codes used by McEliece, has the same security as McEliece's system, as shown in [9].

Niederreiter's system differs from McEliece's system in public-key structure, encryption mechanism, and decryption mechanism. Beware that the specific system in [11] also used different codes — Goppa codes were replaced by generalized Reed-Solomon codes — but generalized Reed-Solomon codes were broken by Sidelnikov and Shestakov in 1992; see [16].

The *Niederreiter secret key* consists of an $n \times n$ permutation matrix P; a nonsingular $(n-k) \times (n-k)$ matrix S; and a parity-check matrix H for a Goppa code $\Gamma(\alpha_1, \ldots, \alpha_n, g)$ of dimension $k = n - td$. As before, the sizes n, k, t are public system parameters, but $\alpha_1, \ldots, \alpha_n, g, P, S$ are randomly generated secrets.

The *Niederreiter public key* is the $(n-k) \times n$ matrix SHP.

Niederreiter encryption of a message $\mathbf{m}$ of length n and weight t: Compute and send $\mathbf{y} = SHP\mathbf{m}^T$.

Niederreiter decryption: By linear algebra find $\mathbf{z}$ such that $H\mathbf{z}^T = S^{-1}\mathbf{y}$. Then $\mathbf{z} - \mathbf{m}P^T$ is a codeword in Γ. Apply Patterson's algorithm to find the error vector $\mathbf{m}P^T$ and thereby $\mathbf{m}$.

CCA2-secure variants. McEliece's system as described above does not resist chosen-ciphertext attacks; i.e., it does not achieve "IND-CCA2 security." For instance, encryption of the same message twice produces two different ciphertexts which can be compared to find out the original message since it is highly unlikely that errors were added in the same positions.

There are several suggestions to make the system CCA2-secure. Overviews can be found in [6, Chapters 5–6] and [12]. All techniques share the idea of scrambling the message inputs. The aim is to destroy any relations of two dependent messages which an adversary might be able to exploit.

If we secure McEliece encryption against chosen-ciphertext attacks then we can use a *systematic* generator matrix as a public key. This reduces the public-key size from kn bits to $k(n-k)$ bits: it is sufficient to store the $k \times (n-k)$ matrix Q described above. Similarly for Niederreiter's system it suffices to store the non-trivial part of the parity check matrix, reducing the public-key size from $(n-k)n$ bits to $k(n-k)$ bits.

3 Review of the Stern attack algorithm

The most effective attack known against the McEliece and Niederreiter cryptosystems is "information-set decoding." There are actually many variants of this attack. A simple form of the attack was introduced by McEliece in [10, Section III]. Subsequent variants were introduced by Leon in [8], by Lee and Brickell in [7], by Stern in [17], by van Tilburg in [18], by Canteaut and Chabanne in [3], by Canteaut and Chabaud in [4], and by Canteaut and Sendrier in [5].

The new attack presented in Section 4 of this paper is most easily understood as a variant of Stern's attack. This section reviews Stern's attack.

How to break McEliece and Niederreiter. Stern actually states an attack on a different problem, namely the problem of finding a low-weight codeword.

However, as mentioned by Canteaut and Chabaud in [4, page 368], one can decode a linear code — and thus break the McEliece system — by finding a low-weight codeword in a slightly larger code.

Specifically, if C is a length-n code over $\mathbf{F}_2$, and $\mathbf{y} \in \mathbf{F}_2^n$ has distance w from a codeword $\mathbf{c} \in C$, then $\mathbf{y} - \mathbf{c}$ is a weight-w element of the code $C + \{0, \mathbf{y}\}$. Conversely, if C is a length-n code over $\mathbf{F}_2$ with minimum distance larger than w, then a weight-w element $\mathbf{e} \in C + \{0, \mathbf{y}\}$ cannot be in C, so it must be in $C + \{\mathbf{y}\}$; in other words, $\mathbf{y} - \mathbf{e}$ is an element of C with distance w from $\mathbf{y}$.

Recall that a McEliece ciphertext $\mathbf{y} \in \mathbf{F}_2^n$ is known to have distance t from a unique closest codeword $\mathbf{c}$ in a code C that has minimum distance at least $2t + 1$. The attacker knows the McEliece public key, a generator matrix for C, and can simply append $\mathbf{y}$ to the list of generators to form a generator matrix for $C + \{0, \mathbf{y}\}$. The only weight-t codeword in $C + \{0, \mathbf{y}\}$ is $\mathbf{y} - \mathbf{c}$; by finding this codeword the attacker finds $\mathbf{c}$ and easily solves for the plaintext.

Similar comments apply if the attacker is given a Niederreiter public key, i.e., a parity-check matrix for C. By linear algebra the attacker quickly finds a generator matrix for C; the attacker then proceeds as above. Similar comments also apply if the attacker is given a Niederreiter ciphertext. By linear algebra the attacker finds a word that, when multiplied by the parity-check matrix, produces the specified ciphertext. The bottleneck in all of these attacks is finding the weight-t codeword in $C + \{0, \mathbf{y}\}$.

Beware that there is a slight inefficiency in the reduction from the decoding problem to the problem of finding low-weight codewords: if C has dimension k and $\mathbf{y} \notin C$ then $C + \{0, \mathbf{y}\}$ has slightly larger dimension, namely $k+1$. The user of the low-weight-codeword algorithm knows that the generator $\mathbf{y}$ will participate in the solution, but does not pass this information to the algorithm. In this paper we focus on the low-weight-codeword problem for simplicity.

How to find low-weight words. Stern's attack has two inputs: first, an integer $w \geq 0$; second, an $(n - k) \times n$ parity-check matrix H for an $[n, k]$ code over $\mathbf{F}_2$. Other standard forms of an $[n, k]$ code, such as a $k \times n$ generator matrix, are easily converted to the parity-check form by linear algebra.

Stern randomly selects $n - k$ out of the n columns of H. He selects a random size-ℓ subset Z of those $n-k$ columns; here ℓ is an algorithm parameter optimized later. He partitions the remaining k columns into two sets X and Y by having each column decide independently and uniformly to join X or to join Y.

Stern then searches, in a way discussed below, for codewords that have exactly p nonzero bits in X, exactly p nonzero bits in Y, 0 nonzero bits in Z, and exactly $w - 2p$ nonzero bits in the remaining columns. Here p is another algorithm parameter optimized later. If there are no such codewords, Stern starts with a new selection of columns.

The search has three steps. First, Stern applies elementary row operations to H so that the selected $n - k$ columns become the identity matrix. This fails, forcing the algorithm to restart, if the original $(n - k) \times (n - k)$ submatrix of H is not invertible. Stern guarantees an invertible submatrix, avoiding the cost of a restart, by choosing each column adaptively as a result of pivots in previous columns. (In theory this adaptive choice could bias the choice of (X, Y, Z), as Stern points out, but the bias does not seem to have a noticeable effect on performance.)

Second, now that this $(n-k) \times (n-k)$ submatrix of H is the identity matrix, each of the selected $n-k$ columns corresponds to a unique row, namely the row where that column has a 1 in the submatrix. In particular, the set Z of ℓ columns corresponds to a set of ℓ rows. For every size-p subset A of X, Stern computes the sum (mod 2) of the columns in A for each of those ℓ rows, obtaining an ℓ-bit vector $\pi(A)$. Similarly, Stern computes $\pi(B)$ for every size-p subset B of Y.

Third, for each collision $\pi(A) = \pi(B)$, Stern computes the sum of the $2p$ columns in $A \cup B$. This sum is an $(n-k)$-bit vector. If the sum has weight $w - 2p$, Stern obtains 0 by adding the corresponding $w - 2p$ columns in the $(n-k) \times (n-k)$ submatrix. Those $w - 2p$ columns, together with A and B, form a codeword of weight w.

4 The new attack

This section presents our new attack as the culmination of a series of improvements that we have made to Stern's attack. The reader is assumed to be familiar with Stern's algorithm; see the previous section.

As a result of these improvements, our attack speeds are considerably better than the attack speeds reported by Canteaut, Chabaud, and Sendrier in [4] and [5]. See the next two sections for concrete results and comparisons.

Reusing existing pivots. Each iteration of Stern's algorithm selects $n-k$ columns of the parity-check matrix H and applies row operations — Gaussian elimination — to reduce those columns to the $(n-k) \times (n-k)$ identity matrix.

Any parity-check matrix for the same code will produce the same results here. In particular, instead of starting from the originally supplied parity-check matrix, we start from the parity-check matrix produced in the previous iteration — which, by construction, already has an $(n-k) \times (n-k)$ identity submatrix. About $(n-k)^2/n$ of the newly selected columns will match previously selected columns, and are simply permuted into identity form with minimal effort, leaving real work for only about $n - k - (n-k)^2/n = (k/n)(n-k)$ of the columns.

Stern says that reduction involves about $(1/2)(n-k)^3 + k(n-k)^2$ bit operations; for example, $(3/16)n^3$ bit operations for $k = n/2$. To understand this formula, observe that the first column requires $\leq n-k$ reductions, each involving $\leq n-1$ additions (mod 2); the second column requires $\leq n-k$ reductions, each involving $\leq n-2$ additions; and so on through the $(n-k)$th column, which requires $\leq n-k$ reductions, each involving $\leq k$ additions; for a total of $(1/2)(n-k)^3 + (k-1/2)(n-k)^2$.

We improve the bit-operation count to $k^2(n-k)(n-k-1)(3n-k)/4n^2$: for example, $(5/128)n^2(n-2)$ for $k = n/2$. Part of the improvement is from eliminating the work for the first $(n-k)^2/n$ columns. The other part is the standard observation that the number of reductions in a typical column is only about $(n-k-1)/2$.

Forcing more existing pivots. More generally, one can artificially reuse exactly $n - k - c$ column selections, and select the remaining c new columns randomly from among the other k columns, where c is a new algorithm parameter. Then only c columns need to be newly pivoted. Reducing c below $(k/n)(n - k)$ saves time correspondingly.

Beware, however, that smaller values of c introduce a dependence between iterations and require more iterations before the algorithm finds the desired weight-w word. See Section 5 for a detailed discussion of this effect.

The extreme case $c = 1$ has appeared before: it was used by Canteaut et al. in [3, Algorithm 2], [4, Section II.B], and [5, Section 3]. This extreme case minimizes the time for Gaussian elimination but maximizes the number of iterations of the entire algorithm.

Illustrative example from the literature: Canteaut and Sendrier report in [5, Table 2] that they need $9.85 \cdot 10^{11}$ iterations to handle $n = 1024$, $k = 525$, $w = 50$ with their best parameters $(p, \ell) = (2, 18)$. Stern's algorithm, with the same $(p, \ell) = (2, 18)$, needs only $5.78 \cdot 10^{11}$ iterations. Note that these are not the best parameters for Stern's algorithm; the parameters $p = 3$ and $\ell = 28$ are considerably better.

Another illustrative example: Canteaut and Chabaud recommend $(p, \ell) = (2, 20)$ for $n = 2048$, $k = 1025$, $w = 112$ in [4, Table 2]. These parameters use $5.067 \cdot 10^{29}$ iterations, whereas Stern's algorithm with the same parameters uses $3.754 \cdot 10^{29}$ iterations.

Canteaut and Chabaud say that Gaussian elimination is the "most expensive step" in previous attacks, justifying the switch to $c = 1$. We point out, however, that this switch often loses speed compared to Stern's original attack. For example, Stern's original attack (without reuse of existing pivots) uses only $2^{124.06}$ bit operations for $n = 2048$, $k = 1025$, $w = 112$ with $(p, \ell) = (3, 31)$, beating the algorithm by Canteaut et al.; in this case Gaussian elimination is only 22% of the cost of each iteration.

Both $c = 1$, as used by Canteaut et al., and $c = (k/n)(n - k)$, as used (essentially) by Stern, are beaten by intermediate values of c. See Section 5 for some examples of optimized choices of c.

Faster pivoting. Adding the first selected row to various other rows cancels all remaining 1's in the first selected column. Adding the second selected row to various other rows then cancels all remaining 1's in the second selected column.

It has frequently been observed — see, e.g., [1] — that there is an overlap of work in these additions: about 25% of the rows will have *both* the first row and the second row added. One can save half of the work in these rows by simply precomputing the sum of the first row and the second row. The precomputation involves at most one vector addition (and is free if the first selected column originally began $1, 1$).

More generally, suppose that we defer additions of r rows; here r is another algorithm parameter. After precomputing all $2^r - 1$ sums of nonempty subsets of these rows, we can handle each remaining row with, on average, $1 - 1/2^r$ vector additions, rather than $r/2$ vector additions. For example, after precomputing

15 sums of nonempty subsets of 4 rows, we can handle each remaining row with, on average, 0.9375 vector additions, rather than 2 vector additions; the precomputation in this case uses at most 11 vector additions. The optimal choice of r is roughly $\lg(n-k) - \lg\lg(n-k)$ but interacts with the optimal choice of c.

See [14] for a much more thorough optimization of subset-sum computations.

Multiple choices of $\boldsymbol{Z}$. Recall that Stern's algorithm finds a particular weight-w word if that word has exactly $p, p, 0$ errors in the column sets X, Y, Z respectively. We generalize Stern's algorithm to allow m disjoint sets $Z_1, Z_2, \ldots, Z_m$ with the same X, Y, each of $Z_1, Z_2, \ldots, Z_m$ having cardinality ℓ; here $m \geq 1$ is another algorithm parameter.

The cost of this generalization is an m-fold increase in the time spent in the second and third steps of the algorithm — but the first step, the initial Gaussian elimination, depends only on X, Y and is done only once. The benefit of this generalization is that the chance of finding any particular weight-w word grows by a factor of nearly m.

For example, if $(n, k, w) = (1024, 525, 50)$ and $(p, \ell) = (3, 29)$, then one set Z_1 works with probability approximately 6.336%, while two disjoint sets Z_1, Z_2 work with probability approximately 12.338%. Switching from one set to two produces a $1.947\times$ increase in effectiveness at the expense of replacing steps $1, 2, 3$ by steps $1, 2, 3, 2, 3$. This is worthwhile if step 1, Gaussian elimination, is more than about 5% of the original computation.

Reusing additions of the $\boldsymbol{\ell}$-bit vectors. The second step of Stern's algorithm considers all p-element subsets A of X and all p-element subsets B of Y, and computes ℓ-bit sums $\pi(A), \pi(B)$. Stern says that this takes $2\ell p\binom{k/2}{p}$ bit operations for average-size X, Y. Similarly, Canteaut et al. say that there are $\binom{k/2}{p}$ choices of A and $\binom{k/2}{p}$ choices of B, each using $p\ell$ bit operations.

We comment that, although computing $\pi(A)$ means $p-1$ additions of ℓ-bit vectors, usually $p-2$ of those additions were carried out before. Simple caching thus reduces the average cost of computing $\pi(A)$ to only marginally more than ℓ bit operations for each A. This improvement becomes increasingly important as p grows.

Faster additions after collisions. The third step of Stern's algorithm, for the pairs (A, B) with $\pi(A) = \pi(B)$, adds all the columns in $A \cup B$.

We point out that, as above, many of these additions overlap. We further point out that it is rarely necessary to compute *all* of the rows of the result. After computing $2(w - 2p + 1)$ rows one already has, on average, $w - 2p + 1$ errors; in general, as soon as the number of errors exceeds $w - 2p$, one can safely abort this pair (A, B).

5 Attack optimization and comparison

Canteaut, Chabaud, and Sendrier announced ten years ago that the original parameters for McEliece's cryptosystem were not acceptably secure: specifically, an attacker can decode 50 errors in a $[1024, 524]$ code over $\mathbf{F}_2$ in $2^{64.1}$ bit operations.

Choosing parameters $p = 2$, $m = 2$, $\ell = 20$, $c = 7$, and $r = 7$ in our new attack shows that the same computation can be done in only $2^{60.55}$ bit operations, almost a $12\times$ improvement over Canteaut et al. The number of iterations drops from $9.85 \cdot 10^{11}$ to $4.21 \cdot 10^{11}$, and the number of bit operations per iteration drops from $20 \cdot 10^6$ to $4 \cdot 10^6$. As discussed in Section 6, we have achieved even larger speedups in software.

The rest of this section explains how we computed the number of iterations used by our attack, and then presents similar results for many more sizes $[n, k]$.

Analysis of the number of iterations. Our parameter optimization relies on being able to quickly and accurately compute the average number of iterations required for our attack.

It is easy to understand the success chance of *one* iteration of the attack:

- The probability of a weight-w word having exactly $w - 2p$ errors in a uniform random set of $n - k$ columns is $\binom{w}{2p}\binom{n-w}{k-2p}/\binom{n}{k}$. The actual selection of columns is adaptive and thus not exactly uniform, but as mentioned in Section 3 this bias appears to be negligible; we have tried many attacks with small w and found no significant deviation from uniformity.
- The conditional probability of the $2p$ errors splitting as p, p between X, Y is $\binom{2p}{p}/2^{2p}$. Instead of having each column decide independently whether or not to join X, we actually make a uniform random selection of exactly $\lfloor k/2 \rfloor$ columns for X, replacing $\binom{2p}{p}/2^{2p}$ with $\binom{\lfloor k/2\rfloor}{p}\binom{\lceil k/2\rceil}{p}/\binom{k}{2p}$, but this is only a slight change.
- The conditional probability of the remaining $w - 2p$ errors avoiding Z, a uniform random selection of ℓ out of the remaining $n - k$ columns, is $\binom{n-k-(w-2p)}{\ell}/\binom{n-k}{\ell}$. As discussed in Section 4, we increase this chance by allowing disjoint sets $Z_1, Z_2, \ldots, Z_m$; the conditional probability of $w - 2p$ errors avoiding at least one of $Z_1, Z_2, \ldots, Z_m$ is

  $$m\frac{\binom{n-k-(w-2p)}{\ell}}{\binom{n-k}{\ell}} - \binom{m}{2}\frac{\binom{n-k-(w-2p)}{2\ell}}{\binom{n-k}{2\ell}} + \binom{m}{3}\frac{\binom{n-k-(w-2p)}{3\ell}}{\binom{n-k}{3\ell}} - \cdots$$

  by the inclusion-exclusion principle.

The product of these probabilities is the chance that the *first* iteration succeeds.

If iterations were independent, as in Stern's original attack, then the average number of iterations would be simply the reciprocal of the product of the probabilities. But iterations are not, in fact, independent. The difficulty is that the number of errors in the selected $n - k$ columns is correlated with the number of errors in the columns selected in the next iteration. This is most obvious in the extreme case $c = 1$ considered by Canteaut et al.: swapping one selected column for one deselected column is quite likely to preserve the number of errors in the selected columns. The effect decreases in magnitude as c increases, but iterations also become slower as c increases; optimal selection of c requires understanding how c affects the number of iterations.

To analyze the impact of c we compute a Markov chain for the number of errors, generalizing the analysis of Canteaut et al. from $c = 1$ to arbitrary c. Here are the states of the chain:

- 0: There are 0 errors in the deselected k columns.
- 1: There is 1 error in the deselected k columns.
- ...
- w: There are w errors in the deselected k columns.
- Done: The attack has succeeded.

An iteration of the attack moves between states as follows. Starting from state u, the attack replaces c selected columns, moving to states $u-c, \dots, u-2, u-1, u, u+1, u+2, \dots, u+c$ with various probabilities discussed below. The attack then checks for success, moving from state $2p$ to state Done with probability

$$\beta = \frac{\binom{\lfloor k/2 \rfloor}{p}\binom{\lceil k/2 \rceil}{p}}{\binom{k}{2p}} \left(m \frac{\binom{n-k-(w-2p)}{\ell}}{\binom{n-k}{\ell}} - \binom{m}{2} \frac{\binom{n-k-(w-2p)}{2\ell}}{\binom{n-k}{2\ell}} + \cdots \right)$$

and otherwise staying in the same state.

For $c = 1$, the column-replacement transition probabilities are mentioned by Canteaut et al.:

- state u moves to state $u-1$ with probability $u(n-k-(w-u))/(k(n-k))$;
- state u moves to state $u+1$ with probability $(k-u)(w-u)/(k(n-k))$;
- state u stays in state u otherwise.

For $c > 1$, there are at least three different interpretations of "select c new columns":

- "Type 1": Choose a selected column; choose a non-selected column; swap. Continue in this way for a total of c swaps.
- "Type 2": Choose c distinct selected columns. Swap the first of these with a random non-selected column. Swap the second with a random non-selected column. Etc.
- "Type 3": Choose c distinct selected columns and c distinct non-selected columns. Swap the first selected column with the first non-selected column. Swap the second with the second. Etc.

Type 1 is the closest to Canteaut et al.: its transition matrix among states $0, 1, \dots, w$ is simply the cth power of the matrix for $c = 1$. On the other hand, type 1 has the highest chance of re-selecting a column and thus ending up with fewer than c new columns; this effectively decreases c. Type 2 reduces this chance, and type 3 eliminates this chance.

The type-3 transition matrix has a simple description: state u moves to state $u + d$ with probability

$$\sum_i \binom{w-u}{i}\binom{n-k-w+u}{c-i}\binom{u}{d+i}\binom{k-u}{c-d-i} \Big/ \binom{n-k}{c}\binom{k}{c}.$$

For $c = 1$ this matrix matches the Canteaut-et-al. matrix.

We have implemented the type-1 Markov analysis and the type-3 Markov analysis. To save time we use floating-point computations with a few hundred bits of precision rather than exact rational computations. We use the MPFI library (on top of the MPFR library on top of GMP) to compute intervals around each floating-point number, guaranteeing that rounding errors do not affect our final results.

As a check we have also performed millions of type-1, type-2, and type-3 simulations and millions of real experiments decoding small numbers of errors. The simulation results are consistent with the experimental results. The type-1 and type-3 simulation results are consistent with the predictions from our Markov-chain software. Type 1 is slightly slower than type 3, and type 2 is intermediate. Our graphs below use type 3. Our current attack software uses type 2 but we intend to change it to type 3.

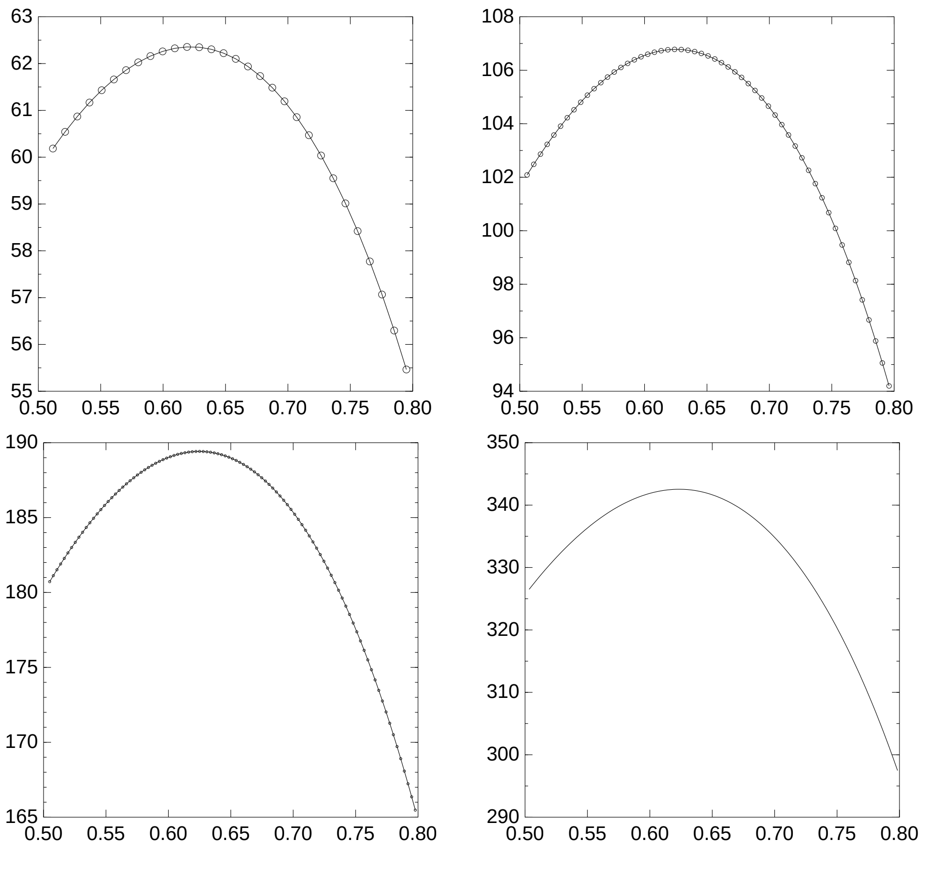

Fig. 1. Attack cost for $n = 1024$, $n = 2048$, $n = 4096$, $n = 8192$. Horizontal axis is the code rate $(n - t\lceil\lg n\rceil)/n$. Vertical axis is lg(bit operations).

Results. For each (n, t) in a wide range, we have explored parameters for our new attack and set new records for the number of bit operations needed to decode t errors in an $[n, n - t \lceil \lg n \rceil]$ code. Figure 1 shows our new records. Note that the optimal attack parameters (p, m, ℓ, c, r) depend on n, and depend on t for fixed n.

6 A successful attack on the original McEliece parameters

We have implemented, and are carrying out, an attack against the cryptosystem parameters originally proposed by McEliece. Our attack software extracts a plaintext from a ciphertext by decoding 50 errors in a $[1024, 524]$ code over $\mathbf{F}_2$.

If we were running our attack software on a single computer with a 2.4GHz Intel Core 2 Quad Q6600 CPU then we would need, on average, approximately 1400 days (2^{58} CPU cycles) to complete the attack. We are actually running our attack software on more machines. Running the software on 200 such computers — a moderate-size cluster costing under \$200000 — would reduce the average time to one week. Note that no communication is needed between the computers.

These attack speeds are much faster than the best speeds reported in the previous literature. Specifically, Canteaut, Chabaud, and Sendrier in [4] and [5] report implementation results for a 433MHz DEC Alpha CPU and conclude that one such computer would need approximately 7400000 days (2^{68} CPU cycles): “decrypting one message out of 10,000 requires 2 months and 14 days with 10 such computers.”

Of course, the dramatic reduction from 7400000 days to 1400 days can be partially explained by hardware improvements — the Intel Core 2 Quad runs at $5.54\times$ the clock speed of the Alpha 21164, has four parallel cores (compared to one), and can perform three arithmetic instructions per cycle in each core (compared to two). But these hardware improvements alone would only reduce 7400000 days to 220000 days.

The remaining speedup factor of 150, allowing us to carry out the first successful attack on the original McEliece parameters, comes from our improvements of the attack itself. This section discusses the software performance of our attack in detail. Beware that optimizing CPU cycles is different from, and more difficult than, optimizing the simplified notion of “bit operations” considered in Section 4.

We gratefully acknowledge contributions of CPU time from several sources. At the time of this writing we are carrying out about $3.26 \cdot 10^9$ attack iterations each day:

- about $1.25 \cdot 10^9$ iterations/day from 38 cores of the Coding and Cryptography Computer Cluster (C4) at Technische Universiteit Eindhoven (TU/e);
- about $0.99 \cdot 10^9$ iterations/day from 32 cores in the Department of Electrical Engineering at National Taiwan University;
- about $0.50 \cdot 10^9$ iterations/day from 22 cores in the Courbes, Algèbre, Calculs, Arithmétique des Ordinateurs (CACAO) cluster at Laboratoire Lorrain de Recherche en Informatique et ses Applications (LORIA);

- about $0.26 \cdot 10^9$ iterations/day from 16 cores of the System Architecture and Networking Distributed and Parallel Integrated Terminal (sandpit) at TU/e;
- about $0.13 \cdot 10^9$ iterations/day from 8 cores of the Argo cluster at the Academic Computing and Communications Center at the University of Illinois at Chicago (UIC);
- about $0.13 \cdot 10^9$ iterations/day from 6 cores at the Center for Research and Instruction in Technologies for Electronic Security (RITES) at UIC; and
- about $0.13 \cdot 10^9$ iterations/day from 4 cores owned by D. J. Bernstein and Tanja Lange.

We plan to publish our attack software to allow public verification of our speed results and to allow easy reuse of the same techniques in other decoding problems.

Number of iterations. Recall that the Canteaut-et-al. attack uses $9.85 \cdot 10^{11}$ iterations on average, with (in our notation) $p = 2$, $\ell = 18$, $m = 1$, and $c = 1$.

To avoid excessive time spent handling collisions in the main loop, we increased ℓ from 18 to 20. This increased the number of iterations to $11.14 \cdot 10^{11}$.

We then increased m from 1 to 5: for each selection of column sets X, Y we try five sets Z_1, Z_2, Z_3, Z_4, Z_5. We further increased c from 1 to 32: each iteration replaces 32 columns from the previous iteration. These choices increased various parts of the per-iteration time by factors of 5 and (almost) 32 respectively; but the choices also combined to reduce the number of iterations by a factor of more than 6, down to $1.85 \cdot 10^{11}$.

Further adjustment of the parameters will clearly produce additional improvements, but having reached feasibility we decided to proceed with our attack.

Time for each iteration. Our attack software carries out an attack iteration in 6.38 million CPU cycles on one core of a busy Core 2 Quad. "Busy" means that the other three cores of the Core 2 Quad are also working on the attack; the cycle counts drop slightly, presumably reflecting reduced L2-cache contention, if only one core of the Core 2 Quad is active.

About 6.20 of these 6.38 million CPU cycles are accounted for by the following major components:

- 0.68 million CPU cycles to select new column sets X and Y and to perform Gaussian elimination. We use 32 new columns in each iteration, as mentioned above. Each new column is handled by an independent pivot, modifying a few hundred thousand bits of the matrix; we use standard techniques to combine 64 bit modifications into a small number of CPU instructions, reducing the cost of the pivot to about 20000 CPU cycles. Further improvements are clearly possible with further tuning.
- 0.35 million CPU cycles to precompute $\pi(L)$ for each single column L. There are $m = 5$ choices of π, and $k = 525$ columns L for each π. We handle each $\pi(L)$ computation in a naive way, costing more than 100 CPU cycles; this could be improved but is not a large part of the overall computation.

- 0.36 million CPU cycles to clear hash tables. There are two hash tables, each with $2^\ell = 2^{20}$ bits, and clearing both tables costs about 0.07 million CPU cycles; this is repeated $m = 5$ times, accounting for the 0.36 million CPU cycles.
- 1.13 million CPU cycles to mark, for each size-p set A, the bit at position $\pi(A)$ in the first hash table. We use $p = 2$, so there are $262 \cdot 261/2 = 34191$ choices of A, and $m = 5$ choices of π, for a total of 0.17 million marks, each costing about 6.6 CPU cycles. Probably the 6.6 could be reduced with further CPU tuning.
- 1.30 million CPU cycles to check, for each set B, whether the bit at position $\pi(B)$ is set in the first hash table, and if so to mark the bit at position $\pi(B)$ in the second hash table while appending B to a list of colliding B's.
- 1.35 million CPU cycles to check, for each set A, whether the bit at position $\pi(A)$ is set in the second hash table, and if so to append A to a list of colliding A's.
- 0.49 million CPU cycles to sort the list of colliding sets A by $\pi(A)$ and to sort the list of colliding sets B by $\pi(B)$. We use a straightforward radix sort.
- 0.54 million CPU cycles to skim through each collision $\pi(A) = \pi(B)$, checking the weight of the sum of the columns in $A \cup B$. There are on average about $5 \cdot 34453 \cdot 34191/2^{20} \approx 5617$ collisions. Without early aborts this step would cost 1.10 million CPU cycles.

For comparison, Canteaut et al. use 260 million cycles on an Alpha 21164 for each of their iterations ("1000 iterations of the optimized algorithm are performed in 10 minutes ... at 433 MHz").

7 Defending the McEliece cryptosystem

This section proposes new parameters for the McEliece cryptosystem.

Increasing n. The most obvious way to defend McEliece's cryptosystem is to increase n, the length of the code used in the cryptosystem. We comment that allowing values of n between powers of 2 allows considerably better optimization of (e.g.) the McEliece/Niederreiter public-key size. See below for examples. Aside from a mild growth in decoding time, there is no obstacle to the key generator using a Goppa code defined via a field $\mathbf{F}_{2^d}$ of size *much* larger than n.

Using list decoding to increase w. The very recent paper [2] has introduced a list-decoding algorithm for classical irreducible binary Goppa codes, exactly the codes used in McEliece's cryptosystem. This algorithm allows the receiver to efficiently decode approximately $n - \sqrt{n(n - 2t - 2)} \geq t + 1$ errors instead of t errors. The sender, knowing this, can introduce correspondingly more errors; the attacker is then faced with a more difficult problem of decoding the additional errors.

List decoding can, and occasionally does, return more than one codeword within the specified distance. In CCA2-secure variants of McEliece's system there is no difficulty in identifying which codeword is a valid message. Our attack can, in exactly the same way, easily discard codewords that do not correspond to valid messages.

Analysis and optimization of parameters. We now propose concrete parameters $[n, k]$ for various security levels in CCA2-secure variants of the McEliece cryptosystem. Recall that public keys in these variants are systematic generator matrices occupying $k(n - k)$ bits.

For (just barely!) 80-bit security against our attack we propose $[1632, 1269]$ Goppa codes (degree $t = 33$), with 34 errors added by the sender. The public-key size here is $1269(1632 - 1269) = 460647$ bits.

Without list decoding, and with the traditional restriction $n = 2^d$, the best possibility is $[2048, 1751]$ Goppa codes ($t = 27$). The public key here is considerably larger, namely 520047 bits.

For 128-bit security we propose $[2960, 2288]$ Goppa codes ($t = 56$), with 57 errors added by the sender. The public-key size here is 1537536 bits.

For 256-bit security we propose $[6624, 5129]$ Goppa codes ($t = 115$), with 117 errors added by the sender. The public-key size here is 7667855 bits.

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.

References

1. Gregory V. Bard. Accelerating cryptanalysis with the Method of Four Russians. Cryptology ePrint Archive: Report 2006/251, 2006. URL: `http://eprint.iacr.org/2006/251`.
2. Daniel J. Bernstein. List decoding for binary Goppa codes, 2008. URL: `http://cr.yp.to/papers.html#goppalist`.
3. Anne Canteaut and Hervé Chabanne. A further improvement of the work factor in an attempt at breaking McEliece's cryptosystem. In P. Charpin, editor, *EUROCODE 94*, 1994. URL: `http://www.inria.fr/rrrt/rr-2227.html`.
4. 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 Transactions on Information Theory*, 44(1):367–378, 1998.
5. Anne Canteaut and Nicolas Sendrier. Cryptanalysis of the original McEliece cryptosystem. In Kazuo Ohta and Dingyi Pei, editors, *Advances in cryptology—ASIACRYPT'98*, volume 1514 of *Lecture Notes in Computer Science*, pages 187–199. Springer, Berlin, 1998.
6. Daniela Engelbert, Raphael Overbeck, and Arthur Schmidt. A summary of McEliece-type cryptosystems and their security. Cryptology ePrint Archive: Report 2006/162, 2006. URL: `http://eprint.iacr.org/2006/162`.
7. 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, volume 330 of *Lecture Notes in Computer Science*, pages 275–280. Springer, Berlin, 1988.

8. Jeffrey S. Leon. A probabilistic algorithm for computing minimum weights of large error-correcting codes. *IEEE Transactions on Information Theory*, 34(5):1354–1359, 1988.
9. Yuan Xing Li, Robert H. Deng, and Xin Mei Wang. On the equivalence of McEliece's and Niederreiter's public-key cryptosystems. *IEEE Transactions on Information Theory*, 40(1):271–273, 1994.
10. Robert J. McEliece. A public-key cryptosystem based on algebraic coding theory, 1978. Jet Propulsion Laboratory DSN Progress Report 42–44. URL: `http://ipnpr.jpl.nasa.gov/progress_report2/42-44/44N.PDF`.
11. Harald Niederreiter. Knapsack-type cryptosystems and algebraic coding theory. *Problems of Control and Information Theory. Problemy Upravlenija i Teorii Informacii*, 15(2):159–166, 1986.
12. Raphael Overbeck and Nicolas Sendrier. Code-based cryptography. In Daniel J. Bernstein, Johannes Buchmann, and Erik Dahmen, editors, *Introduction to post-quantum cryptography.* Springer, Berlin, to appear.
13. Nicholas J. Patterson. The algebraic decoding of Goppa codes. *IEEE Transactions on Information Theory*, IT-21:203–207, 1975.
14. Nicholas Pippenger. The minimum number of edges in graphs with prescribed paths. *Mathematical Systems Theory*, 12:325–346, 1979. URL: `http://cr.yp.to/bib/entries.html#1979/pippenger`.
15. Nicolas Sendrier. On the security of the McEliece public-key cryptosystem. In Mario Blaum, Patrick G. Farrell, and Henk C. A. van Tilborg, editors, *Information, coding and mathematics*, volume 687 of *Kluwer International Series in Engineering and Computer Science*, pages 141–163. Kluwer, 2002.
16. Vladimir M. Sidelnikov and Sergey O. Shestakov. On insecurity of cryptosystems based on generalized Reed-Solomon codes. *Discrete Mathematics and Applications*, 2:439–444, 1992.
17. Jacques Stern. A method for finding codewords of small weight. In Gérard D. Cohen and Jacques Wolfmann, editors, *Coding theory and applications*, volume 388 of *Lecture Notes in Computer Science*, pages 106–113. Springer, New York, 1989.
18. Johan van Tilburg. On the McEliece public-key cryptosystem. In Shafi Goldwasser, editor, *Advances in cryptology—CRYPTO '88*, volume 403 of *Lecture Notes in Computer Science*, pages 119–131, Berlin, 1990. Springer.