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. We show that the system with the originally proposed parameters can be broken on a moderate cluster in about a week. We have implemented our attack and are carrying it out now.

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 our 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. We achieve considerably smaller public key sizes 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 [8] 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 attempts to reduce the key size by using other codes, most notably codes over

* Permanent ID of this document: 7868533f20f51f8d769be2aa464647c9. Date of this document: 2008.07.22. This work has been supported in part by the National Science Foundation under grant ITR–0716498.

larger fields instead of the subfield codes, have led to several breaks of variants leaving essentially only the original system as the strongest candidate.

The best known attacks on the system are based on information set decoding as implemented by Canteaut and Chabaud [2] and analyzed in greater detail by Canteaut and Sendrier in [3].

In this paper we reconsider attacks on the McEliece cryptosystem and present improvements to Stern's attack [12] (an attack predating the Canteaut-Chabaud one) 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 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 and Niederreiter cryptosystems, selected from a much wider range of cryptosystem parameters than have been analyzed before. 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 list-decoding algorithm for binary Goppa codes recently introduced by Bernstein; this 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 [8] 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 $n = 2^m$ 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.

Notes on linear codes. We fix some notation: We refer to a linear code C of length n and dimension k as an $[n, k]$ code. The minimum distance d of C is the smallest Hamming weight of any non-zero element in C.

The generator matrix of an $[n, k]$ code C is the matrix corresponding to the injective linear map from $\mathbf{F}_2^k$ to $C \subset \mathbf{F}_2^n$. The parity check matrix of C is an $(n-k) \times k$-matrix H satisfying $GH^T = 0$. Thus, $H\mathbf{c}^T = 0$ for $\mathbf{c} \in \mathbf{F}_2^n$ if and only if $\mathbf{c} \in C$. For $\mathbf{y} \in \mathbf{F}_2^n$ we call $H\mathbf{y}^T$ the syndrome of $\mathbf{y}$.

Each generator matrix can, after a permutation of columns, be written in systematic form $G = (I_k|Q)$ where I_k denotes the identity matrix of dimension k and Q some matrix of dimension $k \times (n-k)$. The parity check matrix H of C is then given by $H = (-Q^T|I_{n-k})$.

The classical decoding problem is to find the closest codeword $\mathbf{x} \in C$ to a given $\mathbf{y} \in \mathbf{F}_2^n$.

Classical Goppa codes. Fix a finite field $\mathbf{F}_{2^m}$ and consider a set of n distinct elements $\alpha_1, \ldots, \alpha_n$ in $\mathbf{F}_{2^m}$. Fix an integer t, $2 \leq t \leq (2^m - 1)/m$, and an irreducible degree-t polynomial g. Note that we do not require n equal to 2^m.

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

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

The dimension k of the code Γ is at least $n - tm$ and the minimum distance is at least $2t + 1$. Patterson in [10] gave an efficient algorithm to correct t errors

Given the irreducible Goppa polynomial g we can construct the parity check matrix H of the code Γ by taking a closer look at equation (1):

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

We write elements of $\mathbf{F}_{2^m}$ as binary strings in $\{0,1\}^m$. Considering H over $\mathbf{F}_2$ yields a $tm \times n$ binary matrix. We simultaneously consider the McEliece cryptosystem and a variant published by Niederreiter in [9]. We note that the original Niederreiter scheme uses Generalized Reed-Solomon codes and was broken by Sidelnikov and Shestakov in 1992 (see [11]). However, both systems have equivalent security when using a classical Goppa code with same parameters n, k, t as shown in [7].

Public keys. Fix a Goppa code Γ with parameters n, k, t. Let G denote the generator matrix of Γ and H the parity check matrix of Γ, respectively. Choose an $n \times n$ permutation matrix P and a non-singular matrix S of size $k \times k$.

For the McEliece public key set $\hat{G} = SGP$ and publish the pair $(\hat{G}, t)$.

For the Niederreiter scheme choose a non-singular $(n-k) \times (n-k)$-matrix S' instead of S. Set $\hat{H} = S'HP$ and publish the pair $(\hat{H}, t)$.

McEliece encryption of a message $\mathbf{m}$ of length k: Compute $\mathbf{m}\hat{G}$ and add some error vector $\mathbf{e}$ of weight t and length n. Send $\mathbf{y} = \mathbf{m}\hat{G} + \mathbf{e}$.
Decryption: Compute $\mathbf{y}' = \mathbf{y}P^{-1} = \mathbf{m}'G + \mathbf{e}'$ where $\mathbf{m}' = \mathbf{m}S$ and $\mathbf{e}'$ is the permuted error vector $\mathbf{e}$. Use Patterson's algorithm to find $\mathbf{m}'$ and thereby $\mathbf{m} = \mathbf{m}'S^{-1}$.

Niederreiter encryption of a message $\mathbf{m}$ of length n and weight t: Compute and send $\mathbf{y} = \hat{H}\mathbf{m}^T$.
Decryption: Compute $\mathbf{y}' = S'^{-1}\mathbf{y} = HP\mathbf{m}^T \in \mathbf{F}_{2^m}^{n-k}$ which is a syndrome of some element with respect to H. Apply Patterson's algorithm to find $\mathbf{m}' = P\mathbf{m}^T$ and thereby $\mathbf{m}$.

McEliece's system does not resist chosen ciphertext attacks (CCA2 security). For instance, encryption of the same message 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. An overview can be found in [4]. 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.

Further advantages of these techniques reveal themselves when we look at the key sizes.

Size of public keys. McEliece suggests in his original paper to choose an $[1024, 524]$ classical binary Goppa code Γ with irreducible polynomial g of degree $t = 50$. We consider the same parameters for the Niederreiter system.

Storing the public key: If we take actions to secure the McEliece encryption towards chosen ciphertext attacks, we can publish G in its systematic form. In this case it is sufficient to store the $k \times (n-k)$-matrix Q as described above. Similarly for Niederreiter's system it suffices to store the non-trivial part of the parity check matrix.

Storing the private key: the Goppa polynomial g, the matrices P, S and G (S' and H for the Niederreiter system) need to be stored.

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 [8, Section III]. Subsequent variants were introduced by Leon in [6], by Lee and Brickell in [5], by Stern in [12], by van Tilburg in [13], by Canteaut and Chabanne in [1], by Canteaut and Chabaud in [2], and by Canteaut and Sendrier in [3].

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. Leon and Stern actually state attacks on a different problem, namely the problem of finding a low-weight codeword. However, as mentioned by Canteaut and Chabaud in [2, 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, i.e., a syndrome. By linear algebra the attacker finds a word that, when multiplied by the parity-check matrix, produces the specified syndrome. 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, a non-redundant $(n-k) \times n$ 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, the set Z of ℓ columns corresponds to ℓ rows. For every size-p subset A of X, Stern computes the sum 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 [2] and [3]. 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 "order $(1/2)(n-k)^3 + k(n-k)^2$" 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; 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 $(n-k)^2/n$ saves time correspondingly.

Beware, however, that extremely small values of c require more iterations before the algorithm finds the desired weight-w word. Changing only a few of the selected $n-k$ columns will very often preserve the number of errors in those columns; if the number of errors in the previously selected columns was different from $w-2p$ then the number of errors in the newly selected columns is also likely to be different from $w-2p$.

The extreme case $c = 1$ has appeared before: it was used by Canteaut et al. in [1, Algorithm 2], [2, Section II.B], and [3, 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 [3, 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. Canteaut and Chabaud say that Gaussian elimination is the "most expensive step" in previous attacks, justifying the switch to $c = 1$; our experience, however, is that $c = 1$ is suboptimal when the algorithm as a whole is optimized.

Faster pivoting. Kronrod's algorithm (the "Four Russians Algorithm") combines several pivots, reducing the number of additions by typically a factor of three. Full details will appear in the next version of this paper.

Multiple choices of 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.

The probability of a weight-w word having exactly $2p$ errors in a uniform random selection of k columns is $\binom{w}{2p}\binom{n-w}{k-2p}/\binom{n}{k}$. The conditional probability of the $2p$ errors splitting as p, p between X, Y is $\binom{2p}{p}/2^{2p}$. 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 $\binom{n-k-(w-2p)}{\ell}/\binom{n-k}{\ell}$.

We generalize Stern's algorithm to allow m disjoint sets Z with the same X, Y; 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; more precisely, the conditional probability of $w - 2p$ errors avoiding $Z_1, Z_2, \ldots, Z_m$ 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.

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× 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 ℓ-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 $2lp\binom{k/2}{p}$ 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 $A \cup B$.

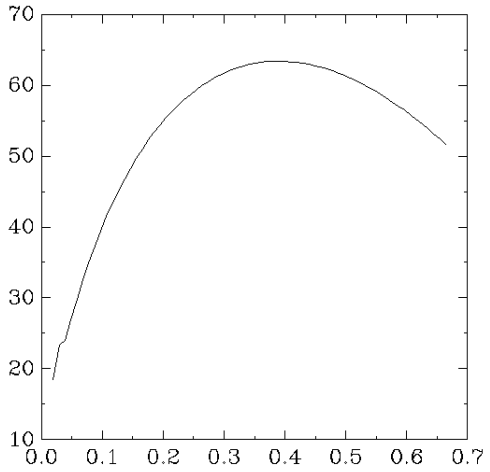
We point out that, as above, many of these additions overlap. We further point out that it is rarely necessary to compute *all* of the rows of the result: After computing $2(w - 2p + 1)$ rows one already has, on average, $w - 2p + 1$ errors; in general, as soon as the number of errors exceeds $w - 2p$,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

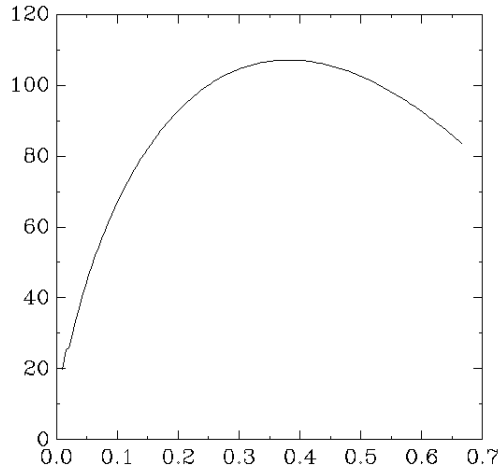
5 Attack optimization and comparison

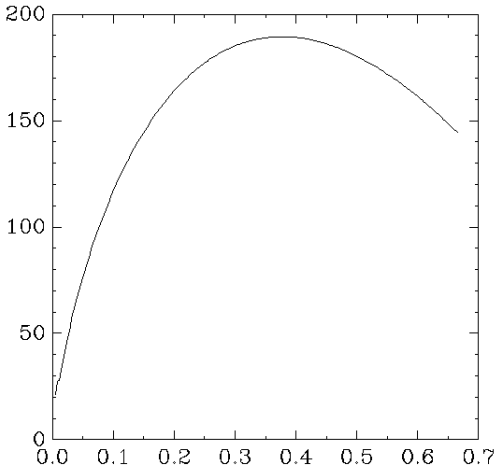
Canteaut, Chabaud, and Sendrier announced ten years ago that the original parameters for McEliece's cryptosystem were not acceptably secure: specifically, an attacker can decode 50 errors in a $[1024, 524]$ code over $\mathbf{F}_2$ in $2^{64.1}$ bit operations.

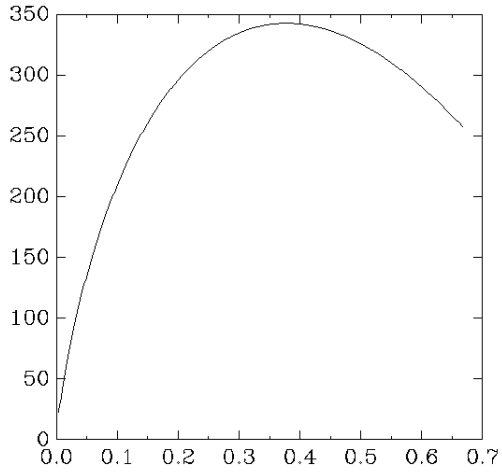
Choosing parameters $p = 3$, $m = 2$, and $\ell = 27$ in our new attack shows that the same computation can be done in only $2^{61.7}$ bit operations. Note that we chose p and ℓ considerably larger than Canteaut et al.

For each $n \in 16\mathbf{Z}$ with $128 \leq n \leq 8192$, for each $t \in \mathbf{Z}$ between 2 and $2n/(3\lceil \lg n \rceil)$, we have computed parameters for our new attack that minimize bit operations for decoding t errors in an $[n, n - t\lceil \lg n \rceil]$ code. The following graphs are for $n = 1024$, $n = 2048$, $n = 4096$, and $n = 8192$ respectively; the horizontal axis is $t\lceil \lg n \rceil / n$, and the vertical axis is lg(bit operations).









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.

These attack speeds are much faster than the best speeds reported in the previous literature. Specifically, Canteaut, Chabaud, and Sendrier in [2] and [3] 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 can perform three arithmetic instructions per cycle (compared to two for the Alpha 21164), runs at 5.54× the clock speed, and has four parallel cores. But these hardware improvements alone would only reduce 7400000 days to 220000 days.

The remaining speedup factor of 150, making the attack feasible for anyone who owns a few computers, 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 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. Our attack software uses, on average, about $1.8 \cdot 10^{11}$ iterations. For comparision, Canteaut et al. report that their attack uses, on average, $9.85 \cdot 10^{11}$ iterations.

There are two major reasons for the reduced iteration count. First, we choose $m = 5$: for each selection of column sets X, Y we try five sets Z_1, Z_2, Z_3, Z_4, Z_5. Second, we choose $c = 32$: each iteration replaces 32 columns from the previous iteration. Compared to the extreme choices $m = 1$ and $c = 1$ used in the previous literature, our choices $m = 5$ and $c = 32$ increase various parts of the per-iteration time by factors of 5 and (almost) 32 respectively; but the choices also combine to reduce the number of iterations by a factor of about 6.23, down to $1.58 \cdot 10^{11}$.

To avoid excessive time spent handling collisions in the main loop, we increased ℓ from 18 to 20, increasing the number of iterations somewhat.

To double-check our predictions of average iteration counts we have carried out 100000 experiments decoding 10 errors, 100000 experiments decoding 11 errors, etc. The results—for example, 100000 experiments using 15 errors used 119.86 iterations on average—are consistent with our predictions.

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 Kronrod's algorithm and with further CPU tuning.
- 0.35 million CPU cycles to precompute $\pi(C)$ for each single column C. There are $m = 5$ choices of π, and $k = 525$ columns C for each π. We handle each $\pi(C)$ 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 comparision, 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 n does not have to be a power of 2, and 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 cost in decoding time, there is no obstacle to the receiver using a field size *much* larger than n.

Using list decoding to increase w. Bernstein has very recently 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 the valid codeword. Our attack can easily discard invalid codewords in exactly the same way.

Analysis and optimization of parameters. For (just barely!) 80-bit security against our best attacks we propose degree-33 length-1616 Goppa codes, with 34 errors added by the sender. The public key size here is 454839 bits.

Without list decoding, and with the traditional restriction $n = 2^m$, the best possibility is degree-27 length-2048 Goppa codes. The public key here is considerably larger, namely 520047 bits.

For 128-bit security we propose degree-57 length-2928 Goppa codes, with 58 errors added by the sender. The public key size here is 1534896 bits.

For 256-bit security we propose degree-118 length-6544 Goppa codes, with 120 errors added by the sender. The public key size here is 7685340 bits.

References

1. 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.
2. 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.
3. Anne Canteaut and Nicolas Sendrier. Cryptanalysis of the original McEliece cryptosystem. In Kazuo Ohta and Dingyi Pei, editors, *ASIACRYPT*, volume 1514 of *Lecture Notes in Computer Science*, pages 187–199. Springer, 1998.
4. Daniela Engelbert, Raphael Overbeck, and Arthur Schmidt. A summary of McEliece-type cryptosystems and their security. Cryptology ePrint Archive: Report 2006/162, 2006.
5. Pil Joong Lee and Ernest F. Brickell. An observation on the security of McEliece's public-key cryptosystem. In *EUROCRYPT*, pages 275–280, 1988.
6. Jeffrey S. Leon. A probabilistic algorithm for computing minimum weights of large error-correcting codes. *IEEE Transactions on Information Theory*, 34(5):1354–, 1988.
7. 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–, 1994.
8. Robert J. McEliece. A public-key cryptosystem based on algebraic coding theory. Technical report, CA, 1978.
9. Harald Niederreiter. Knapsack-type cryptosystems and algebraic coding theory. *Problems Control Inform. Theory/Problemy Upravlen. Teor. Inform.*, 15(2):159–166, 1986.
10. Nicholas J. Patterson. The algebraic decoding of Goppa codes. *IEEE Transactions on Information Theory*, 21:203–207, 1975.
11. V. M. Sidelnikov and S. O. Shestakov. On insecurity of cryptosystems based on generalized Reed-Solomon codes. *Discrete Math. Appl.*, 2:439–444, 1992.
12. 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, 1988.
13. Johan van Tilburg. On the McEliece public-key cryptosystem. In Shafi Goldwasser, editor, *CRYPTO*, volume 403 of *Lecture Notes in Computer Science*, pages 119–131. Springer, 1988.