Cryptanalysis of the Sidelnikov cryptosystem

Lorenz Minder*, Amin Shokrollahi

Laboratoire de mathématiques algorithmiques (LMA), EPFL

Abstract. We present a structural attack against the Sidelnikov cryptosystem [8]. The attack creates a private key from a given public key. Its running time is subexponential and is effective if the parameters of the Reed-Muller code allow for efficient sampling of minimum weight codewords. For example, the length 2048, 3rd-order Reed-Muller code as proposed in [8] takes roughly an hour to break on a stock PC using the presented method.

Keywords: Sidelnikov cryptosystem, McEliece cryptosystem, error-correcting codes, structural attack.

1 Introduction

The McEliece cryptosystem [6] is one of the oldest known public-key cryptosystems. The fact that it has not been broken in more than a quarter of a century and that the best known attacks today are still exponential speaks for itself.

Despite its impressive security record, it plays a rather marginal role in practice, being far less popular than other systems, such as RSA. The principal reason for this is that the McEliece cryptosystem is not as efficient as the alternatives. The main problems are its large public keys, the fact that the message is subject to expansion (the cryptogram is longer than the plaintext message), and its potentially high decryption complexity.

To understand the tradeoffs, we recall how the McEliece cryptosystem works. Let $\mathcal{C}$ be a linear binary Goppa code of block length n, dimension k, and having a decoding algorithm correcting up to t errors. Let G be a $k \times n$ generator matrix for the code. Let P be a random $n \times n$ permutation matrix. Then there is an efficient decoder for the code generated by $G \cdot P$. Let A be a $k \times k$ invertible matrix. The code generated by

$$G_{\text{pub}} := AGP$$

is the same as the code generated by GP. The public key is the pair (G_{pub}, t). To encrypt a message vector $x := (x_1, \ldots, x_k) \in \mathbb{F}_2^k$, we first compute xG_{pub} and then add t errors at random positions. The resulting vector y is the cryptogram.

The decrypting problem is to recover the value of x given y. The receiver can decrypt this message, since he knows a decoding algorithm for $\mathcal{C}$. An attacker has to

* Supported by the Swiss National Fund, grant 200021-103683

either resort to general decoding techniques, and attempt to solve a problem which appears to be intractable, or recover the structure of the code given by G_{pub}. For further details, see [6].

The McEliece cryptosystem can be generalized to codes other than Goppa codes. A priori, any family of linear codes having decoders which allow for efficient correction of a large number of errors with high probability could be used.

The efficiency and security of such a cryptosystem depend on several factors. First, the number t of correctable errors has to be very large to render general linear decoding algorithms inefficient, which is a security requirement. In addition, the McEliece cryptosystem can be modified in a manner so that the expansion factor to which a message is subjected depends on t, see the paper by Niederreiter [7] for such modifications. At the limit, if capacity-achieving codes could be used, t could be made so large that the expansion factor would converge to 1.

Secondly, the difficulty of recovering the structure of a code given by an arbitrary, permuted generator matrix is highly dependent on the code in question. An interesting dichotomy can be observed here: While modern, graph-based codes (like LDPC-, expander-, LT- or turbo-codes) are all unsafe because of the sparse parity checks revealing their structure, classical algebraic codes have proved widely resistant to structural attacks. The most notable exception is given by Sidelnikov and Shestakov [9], showing that generalized Reed-Solomon codes are unsafe.

In 1994, the first author of [9] proposed a variant of the McEliece system, basically replacing the Goppa codes with Reed-Muller codes [8]. The advantage of using Reed-Muller codes is that very efficient decoding algorithms are known for these codes. Thus, using these codes allows simultaneously for faster decryption, smaller key sizes and expansion factors close to 1, if the Niederreiter variant is used.

While all those properties sound very promising, we will show in this paper that Reed-Muller codes are a bad choice, too. More specifically, we present a method to find a private key for a given public key. The most costly step of this procedure is that of finding minimum weight codewords in the code. In the low-rate setting of Reed-Muller codes, this is feasible even for fairly long block lengths. This attack is, to our knowledge, the first known effective attack against this cryptosystem, and it breaks in particular Sidelnikov's original proposed parameters ($m = 11, r = 3$) in less than an hour on a stock PC.

The key observation that makes the attack work, is the fact that minimum weight words in the r-th Reed-Muller code of length 2^m (this code is denoted $\mathcal{R}(r,m)$) are products of r minimum weight words in $\mathcal{R}(1,m)$. The attack uses this fact to reduce the order: First, minimum weight codewords in the given, permuted $\mathcal{R}(r,m)$ are found, and then a statistical test is applied to find factors of those words which lie in the accordingly permuted $\mathcal{R}(r-1,m)$. By iterating this procedure, ultimately the permuted version of $\mathcal{R}(1, m)$ is found, which allows easy identification of a suitable permutation.

The fact that there is only a single Reed-Muller code for a given block length and dimension has been noticed as a cryptographic weakness before. The best known previous attack is using the support splitting algorithm[11], an algorithm to find a permutation between two equivalent codes. While this algorithm is generally very fast, it is ineffective against Reed-Muller codes: Its running time is exponential in the dimension of the hull (i.e., the intersection of the code with its dual), and Reed-Muller codes have a hull as big as the code itself.

This paper is organized as follows. First, we give a short summary of Reed-Muller codes. Second, we present our results on the structure of these codes which form the basis for the attack. Third, we present the attack with a running time analysis.

2 Reed-Muller codes

To recall the notation, we start by presenting the construction of these codes. For further details the reader is referred to [5] or [3].

Reed-Muller codes can be constructed by using Boolean functions. A Boolean function of m variables can be evaluated on 2^m different positions. So to each Boolean function we can associate a binary word of length 2^m. The code $\mathcal{R}(r, m)$ is the set of words obtained by evaluating all the Boolean functions of degree $\leq r$ in this way. We will subsequently call the variables evaluated $v_1, \ldots, v_m$.

We denote by $\mathcal{B}(r, \{v_1, \ldots, v_m\})$ the set of Boolean functions in the variables $v_1, \ldots, v_m$ of degree at most r.

Note that since the base field is $\mathbb{F}_2$, the term v_i^2 can be simplified to v_i, which implies that the degree of any variable in any term of these Boolean functions is at most 1.

The fact that all functions generating words in $\mathcal{B}(r-1, \{v_1, \ldots, v_m\})$ are also in $\mathcal{B}(r, \{v_1, \ldots, v_m\})$ implies the following observation.

Proposition 2.1. *For any m, we have $\mathcal{R}(0, m) \subset \mathcal{R}(1, m) \subset \cdots \subset \mathcal{R}(m, m)$.*

In what follows, we frequently switch back and forth between $\mathcal{B}(r, \{v_1, \ldots, v_m\})$ and $\mathcal{R}(r, m)$. Doing so in the most explicit manner would make the reasonings a lot harder to read, and for this reason we decided to treat codewords and Boolean functions as interchangeable. Note, however, that a codeword has a fixed length, while a function does not. If $x \in \mathcal{R}(r, m)$ is a codeword, its *extension* to $\mathcal{R}(r, m+1)$ is the codeword (x, x), i.e., the codeword obtained by evaluating the function $f \in \mathcal{B}(r, \{v_1, \ldots, v_m\})$ at all the possible values of $(v_1, \ldots, v_m, v_{m+1})$. Similarly, if $x \in \mathcal{R}(r, m)$ is a codeword whose corresponding function does not depend on v_m, then we can *reduce* x to $\mathcal{R}(r, m-1)$, by evaluating the function f corresponding to x on all the possible values of $(v_1, \ldots, v_{m-1})$. Note that in a Reed-Muller code, a position (coordinate) within a codeword can be specified by the value of $(v_1, \ldots, v_m)$.

The block length n, dimension k and minimum distance d of $\mathcal{R}(r, m)$ are

$$n = 2^m, \qquad k = \sum_{i=0}^{r} \binom{m}{i}, \qquad d = 2^{m-r}.$$

The *support* of a codeword $x \in \mathcal{R}(r, m)$, noted by $\operatorname{supp}(x)$, is the set of positions i, for which $x_i \neq 0$.

3 Minimum-weight codewords

We will now present the structural property of Reed-Muller codes which constitutes the theoretical foundation of our cryptanalysis of the Sidelnikov cryptosystem.

The fact that products of r linearly independent first-order codewords are minimum weight in $\mathcal{R}(r, m)$ is well-known. The following proposition states the converse, namely, that minimum weight codewords in Reed-Muller codes can always be written as a (pointwise) product of suitable words in the corresponding first order code. In other words, the only functions giving rise to minimum weight codewords are products of functions in $\mathcal{B}(1, \{v_1, \ldots, v_m\})$.

Proposition 3.1. *Let $f \in \mathcal{R}(r, m)$ be a word of minimum weight. Then there exist $f_1, f_2, \ldots, f_r \in \mathcal{R}(1, m)$, such that*

$$f = f_1 \cdot f_2 \cdots f_r,$$

as functions. The f_i are of minimum weight in $\mathcal{R}(1, m)$.

Proposition 3.1 is proved in [4]. The same paper also gives more precise formulas for the weight distribution, which can be used, in particular, to estimate the number of minimum-weight words:

Proposition 3.2. *There are at least*

$$2^{mr-r(r-1)}.$$

minimum weight codewords in $\mathcal{R}(r, m)$.

We will make use of this fact in the analysis of the running time of our algorithm.

4 Cryptanalysis of the Sidelnikov cryptosystem

The Sidelnikov variant of the McEliece cryptosystem [8] uses Reed-Muller codes in combination with powerful decoding algorithms.

Reed-Muller codes are low-rate if any interesting error-correction capability is to be obtained, which makes it easy to apply algorithms such as the Canteaut-Chabaud-algorithm [1] to find low weight words, and also to decode if the number of errors is less than $d/2$ (half the minimum distance). However, there are decoding-algorithms for Reed-Muller codes which decode many more errors (with high probability) than $d/2$, and thus the low weight word finding algorithms cannot be directly used for decoding.

Such algorithms can still be used to find minimum weight words in codes with suitable parameters, though. In this section, we show how to exploit this fact to invert trapdoors from Reed-Muller codes.

4.1 Outline of the attack

We now present an algorithm which, given a permuted, scrambled Reed-Muller code $\mathcal{C}$, constructs a permutation σ such that if the positions of $\mathcal{C}$ are permuted accordingly, the resulting code is a Reed-Muller code.

Let σ be any permutation on $\{1, \ldots, n\}$. For any code $\mathcal{C}$ of length n, we denote by $\mathcal{C}^\sigma$ the code obtained from $\mathcal{C}$ with the positions permuted according to σ, i.e., a word $(x_0, x_1, \ldots, x_n)$ will be a codeword in $\mathcal{C}^\sigma$ if and only if $(x_{\sigma^{-1}(1)}, \ldots, x_{\sigma^{-1}(n)}) \in \mathcal{C}$.

The sketch of the attack is as follows. Let $\mathcal{C} = \mathcal{R}(r, m)^\sigma$ for some unknown σ, given by an arbitrary generator matrix.

1. Find codewords in $\mathcal{C}$ which with very high probability also belong to $\mathcal{R}(r-1, m)^\sigma$. Find enough such vectors to build a basis of $\mathcal{R}(r-1, m)^\sigma$.
2. Iterate the previous step (with decreasing r) until obtaining $\mathcal{R}(1, m)^\sigma$.
3. Determine a permutation τ such that $\mathcal{R}(1, m)^{\tau\circ\sigma} = \mathcal{R}(1, m)$. Then $\mathcal{R}(r, m)^{\tau\circ\sigma} = \mathcal{R}(r, m)$, and this fact can then be used to decode.

The meat of the attack lies in the first step, which is based on the properties of Reed-Muller codes stated in the previous section.

4.2 Finding the subcode $\mathcal{R}(r-1, m)^\sigma \subseteq \mathcal{R}(r, m)^\sigma$

The basic idea of this step is to find a codeword for which we know that it is a product of other codewords, and then to split off a factor lying in the $\mathcal{R}(r-1, m)^\sigma$ subcode.

By proposition 3.1, a minimum weight codeword is actually a product of several codewords of $\mathcal{R}(1,m)^\sigma$. Hence, we do the following: We find a minimum weight codeword x and split off a factor of this word.

To this end, we shorten the code on $\mathrm{supp}(x)$, and use the structure of the shortened code to find a factor of x which lies in $\mathcal{R}(r-1,m)^\sigma$.

Finding enough words in $\mathcal{R}(r-1,m)^\sigma$ will result in a basis of $\mathcal{R}(r-1,m)^\sigma$.

Finding factors of minimum weight words. We drop the permutation σ in this section, since our ideas do not depend on σ.

Let $x \in \mathcal{R}(r,m)$ be a minimum weight codeword. Using proposition 3.1, and changing the basis, we can assume that $x = v_1 v_2 \cdots v_r$. Let $\mathcal{C}_{\mathrm{supp}(x)}$ be the code $\mathcal{R}(r,m)$ shortened on the support of x. (In other words, $\mathcal{C}_{\mathrm{supp}(x)}$ is the subcode of $\mathcal{R}(r,m)$ containing only the words which are zero on $\mathrm{supp}(x)$, and with these positions punctured afterwards.)

Write $\bar{v} = (v_{r+1}, \ldots, v_m)$, and let f be a codeword in $\mathcal{C}_{\mathrm{supp}(x)}$. Then we can write f as

$$f(v_1, \ldots, v_r, \bar{v}) = \sum_{I \in 2^{\{1,\ldots,r\}}} f_I(\bar{v}) \cdot \prod_{i \in I} v_i,$$

where for each $I \subseteq \{1, \ldots, r\}$, we have $f_I \in \mathcal{B}(r-|I|, \{v_{r+1}, \ldots, v_m\})$. The condition that f be 0 on $\{v_1 = v_2 = \cdots = v_r = 1\}$ implies

$$0 = \sum_{I \in 2^{\{1,\ldots,r\}}} f_I(\bar{v}), \tag{1}$$

and shows in particular that $f_\emptyset(\bar{v}) \in \mathcal{B}(r-1, \{v_{r+1}, \ldots, v_m\})$. Therefore, if we take any codeword in the shortened code, fix a value for $(v_1, \ldots, v_r)$, and look at the positions determined by this value, we get a codeword in $\mathcal{R}(r-1, m-r)$. In other words, the shortened code is a concatenated code[1] with the inner codewords being on the disjoint sets of positions determined by the value of $(v_1, \ldots, v_r)$.

We shorten on $\mathrm{supp}(x)$, i.e., the set $\{v_1 = \cdots = v_r = 1\}$, so there are actually $2^r - 1$ such sets, and each is of length 2^{m-r}. We apply the algorithm of the next section (Algorithm 1) to find the sets and then construct a word y of length 2^m that has ones exactly on the points $\{v_1 = \cdots = v_r = 1\} \cup S$, where S is one of the determined sets, say $\{v_1 = v_2 = \cdots = v_\ell = 0, v_{\ell+1} = v_{\ell+2} = \cdots = v_r = 1\}$. The set $\mathrm{supp}(x) \cup S$ can also be written as $\{v_1 = v_2 = \cdots = v_\ell, v_{\ell+1} = \cdots = v_r = 1\}$, and hence we can write

$$y = (1 + v_1 + v_2)(1 + v_2 + v_3) \cdots (1 + v_{\ell-1} + v_\ell) v_{\ell+1} \cdots v_r,$$

[1] A *concatenated code* is a subspace of the Cartesian product of several nontrivial codes.

which shows that $y \in \mathcal{B}(r-1, \{v_1, \dots, v_r\})$. Note that one can write $x = v_i y$ for any $1 \le i \le \ell$, showing that y is indeed a factor of x.

Finding inner words in the shortened code. To solve the problem of distinguishing the sets with different values of $(v_1, \dots, v_r)$, we use the fact that the code is a concatenated code, with an inner codeword on each of these sets.

The problem of recovering concatenated codes has previously been studied by Sendrier; the algorithm presented in [10] could possibly be applied in our case, if one showed that the code in question verifies the assumptions of this algorithm, namely that the most lightweight parity checks all have their support within one inner word.

Another possibility is to use a similar method which acts on the code itself, rather than on its dual, and works well in our setting. The method is based on a statistical analysis, and we start by describing the relevant random experiment. Let $\mathcal{C}$ be a concatenated code, i.e.,

$$\mathcal{C} \subseteq \underbrace{\mathcal{C}_i \times \cdots \times \mathcal{C}_i}_{n \text{ times } \mathcal{C}_i},$$

where $\mathcal{C}_i$ is a non-trivial code of length n_i and relative minimum distance δ, called the *inner code*.

For our analysis, we need the following assumption: If $Y \in \mathcal{C}$ is sampled randomly in the low weight words of $\mathcal{C}$, we assume that the events $\{Y_i = 1\}$ and $\{Y_j = 1\}$ are independent if the positions i and j do not belong to the same inner block. (Note that this is almost universally true for linear codes with Y sampled from all the words, and not just the low weight ones.)

Now we randomly sample words of relative weight $< \delta$ from $\mathcal{C}$. Call these samples $X_0, X_1, \dots$, and denote by $(X_\ell)_k$ the k-th position of X_ℓ. For two indexes $1 \le i < j \le n_i \cdot n$, we define the random variable

$$I_{ij,k} := \begin{cases} 1 & \text{if } (X_k)_i = 1 \text{ and } (X_k)_j = 1, \\ 0 & \text{otherwise.} \end{cases}$$

The punch line will be that the behaviour of $I_{ij,k}$ depends on whether i and j lie within the same inner code or not.

We first assume that i and j are not in the same inner block; then $(X_k)_i$ and $(X_k)_j$ will be independent random variables, and we get

$$\begin{aligned} E[I_{ij,k}] &= \text{Prob}((X_k)_i = 1 \wedge (X_k)_j = 1) \\ &= \text{Prob}((X_k)_i = 1)\text{Prob}((X_k)_j = 1) \\ &\approx \delta^2, \end{aligned}$$

assuming the relative weight of X_k is very likely close to δ.

The situation is different if i and j are in the same inner block. Let ϵ_k denote the fraction of zero inner codewords of X_k, and let $T_{k,i}$ be the indicator variable being one whenever the inner block of X_k containing the position i (and also position j) is nonzero. Then we get the following estimate for the case where i and j are in the same inner block:

$$\begin{aligned} E[I_{ij,k} \mid \epsilon_k] &= \operatorname{Prob}((X_k)_i = 1 \wedge (X_k)_j = 1 \mid T_{k,i} = 1, \epsilon_k) \cdot \operatorname{Prob}(T_{k,i} = 1 \mid \epsilon_k) \\ &\approx \left(\frac{\delta}{1-\epsilon_k}\right)^2 \epsilon_k \\ &= \delta^2 \cdot \frac{\epsilon_k}{(1-\epsilon_k)^2}. \end{aligned}$$

Since the relative weight of X_k is less than δ, this means that the average relative weight of the inner blocks is less than δ. Knowing that the relative distance of the inner code is δ, we get the combinatorial guarantee that at least one of the inner code blocks contains the zero codeword. We therefore know that $\epsilon_k \geq n^{-1}$. (In reality, we expect a constant fraction of them to be zero.)

Now, if for each pair of indices (i, j), we compute

$$S_{ij} := \sum_{k=1}^{N} I_{ij,k},$$

then, if N is large enough, those random variables can be used to determine the inner codewords: Just declare (i, j) as belonging to the same set whenever S_{ij} is large enough.

After the sampling, the values S_{ij} can then be used to recover the sets, using a greedy algorithm, for example. Algorithm 1 illustrates this approach.

Note that the behaviour of ϵ_k has an impact on the complexity of the algorithm. The bound $\epsilon_k \geq n^{-1}$ guarantees that only a polynomial number of low weight codewords has to be sampled, but larger values cause much faster convergence. (In practice, choosing the number of observations linear in the number of sets works well over a wide parameter range, although this is significantly less than what we can prove to be sufficient.)

We close this section by noting that according to our definition, Reed-Muller codes themselves are concatenated codes, so one could think of applying this method directly, rather than first finding minimum weight words and shortening. This does not work, since the minimum distance is in this case just large enough to prevent any codewords from lying in the space we want to sample from.

Algorithm 1 Decompose inner sets of $\mathcal{C}$

$\mathcal{C}$ is a concatenated code of block length $N = n \cdot n_i$. The inner code $\mathcal{C}_i$ has distance d_i and length n_i. M is the number of samples deemed sufficient.

```
Let S_ij ← 0, 1 ≤ i, j ≤ N.
for i = 1, ..., M do
   Sample a word (x_1, ..., x_N) ∈ C of weight < N(d_i/n_i).
   for each (i, j) with x_i = x_j = 1 and i ≠ j do
      Increment S_ij.
   end for
end for
for e = 1, ..., n do
   Let i be such that S_ij is maximal for some j, i.e., i ← arg max_{1≤i≤N} max_{1≤j≤N} S_ij
   T_e ← {i}
   while |T_e| < n_i do
      Let 1 ≤ i ≤ N be a vertex such that Σ_{j∈T_e} S_ij is maximal.
      T_e ← T_e ∪ {i}
      Let S_ji ← −∞ and S_ij ← −∞ for all 1 ≤ j ≤ N.
   end while
end for
```

4.3 The case $r = 1$

Consider the matrix A formed by the rows corresponding to the codewords v_m, $v_{m-1}, \ldots, v_1$ of the (unpermuted) $\mathcal{R}(1, m)$. By construction, the i-th column of this matrix is just the number $i - 1$, if we read the vector as a binary number. Any possible binary vector of length m appears exactly once among the columns of this matrix, and if we add the all-one row, we get a generator matrix for a first-order Reed-Muller code.

Now, let $f_1, f_2, \ldots, f_m, f_{m+1}$ be a random basis of $\mathcal{R}(1, m)^\sigma$. If the all-one codeword is not linearly dependent on $f_1, \ldots, f_m$, then in the matrix A^σ formed by the rows $f_1, \ldots, f_m$, each column-vector is distinct. Thus, we can just reorder the columns by moving the zero-vector to the first position, etc., and thus obtain the matrix A. The same permutation applied to the positions of $\mathcal{R}(1, m)^\sigma$ will then yield $\mathcal{R}(1, m)$.

This suggests a simple method to find a suitable permutation: Pick any basis $f_1, \ldots, f_{m+1}$ of $\mathcal{R}(1, m)^\sigma$, check if the columns of the corresponding matrix A^σ are distinct, repeat if not, identify the corresponding permutation otherwise.

What is the success-probability of such an iteration? Since the f_i are linearly independent, the following estimate of this probability holds:

$$\frac{(2^{m+1} - 2)(2^{m+1} - 2^2) \cdots (2^{m+1} - 2^m)}{(2^{m+1} - 1)(2^{m+1} - 2) \cdots (2^{m+1} - 2^{m-1})} = \frac{2^m}{2^{m+1} - 1} > \frac{1}{2}.$$

In other words, we need merely two trials on the average.

4.4 Running time analysis

In the analysis, we will take the quantity $n = 2^m$ (the block length) as the input length, and we will assume r to be small with respect to m which leads to a low-rate setting. This assumption is based on the fact that Reed-Muller codes behave very poorly when r is large, and are therefore practically useless in these instances. For this reason, we will assume $r/m \to 0$ and $r < m/2$. In practice, r is usually a small constant. See [2] for tradeoffs between r, m and decoding thresholds.

The only computationally hard operation of the attack is the one of finding low weight words in a code, everything else is polynomial time. Thus, in order to determine the running time up to a polynomial factor, it is sufficient to verify that only a polynomial number of low weight words is needed, and then to restrict attention to the low weight word finding algorithm.

Checking that only a polynomial number of low weight words has to be found is straightforward: In order to find a single vector in $\mathcal{R}(r-1,m)^\sigma$, a minimum weight word in the original code has to be found, and then the statistical test has to be performed to recover the concatenated structure of the shortened code. Since the bias in the statistical test is at least $(2^r-1)^{-2}$ per observation, we have to collect $O(2^{2r}) = O(2^m) = O(n)$ vectors to get good estimates.

Thus, finding a single vector of $\mathcal{R}(r-1,m)^\sigma$ needs the sampling of a polynomial number of low weight words. But then, since $O(k) = O(n)$, so does clearly the sampling for a complete basis of $\mathcal{R}(r-1,m)^\sigma$. And given that $r \ll n$, the reduction to $\mathcal{R}(1,m)^\sigma$ requires still only a polynomial number of samples.

Because of this, we conclude that, in the exponent, only the complexity of the low weight word finding algorithm matters asymptotically, and we restrict our attention to this algorithm. In practice, those polynomial factors do of course matter to some extent, but notice that the degree of the polynomial is not very large.

Finding very low weight codewords. The problem of finding very low weight words is generally intractable for linear codes. For example, if the rate is kept fixed and the relative weight of the sought word is fixed and small enough, then even the best known algorithms are exponential in the block length.

However, finding low weight words is much easier if the rate is low, and if it is not actually fixed but converges to 0 with the block length.

Good methods for finding low weight words are based on the following (information set decoding) algorithm: Take a random $k \times n$ generator matrix G of the code, pick a random set I of k columns of G, and diagonalize the matrix G on the set I. Check the rows of G to see if any of them is low weight. If not, try again with another random set I.

The condition for a specific word in G of weight w to pop up as a row in such a diagonalized matrix is that exactly one of its bits is inside the information set and the other ones are outside. To simplify, we instead compute the probability that none of its bits are in the information set, a probability which is a bit smaller. We can approximate this by noting that if k is small compared to n, the probability that none of the positions of I match with the support of the word of weight w is roughly

$$\left(1 - \frac{w}{n}\right)^k . \tag{2}$$

This probability becomes large if k is very small with respect to n (i.e., the rate is very low), or if w is very small.

Note that (2) estimates the probability of finding a *single* word of the given weight given a random set I. If many words of the desired weight exist, the probability has to be multiplied with the number of such words. The above estimate decreases with w, but if such an algorithm is applied to find any word of weight $\leq w_0$, then the larger w_0 is, the easier the task becomes. The reason for this apparent contradiction is simply that the number of acceptable words increases dramatically with w_0.

Finite-length analysis. The goal of this section is to specialize (2) to the case of Reed-Muller codes, and to devise a crude bound which allows to estimate the feasibility of the low weight word finding problems (and thus the attack) for different values of r and m.

We first study the hardness of the minimum weight word finding procedure for Reed-Muller codes. In this case, we have $w = 2^{m-r}$ and $k = \sum_{i=0}^{r} \binom{m}{i} \leq \frac{m-r+1}{m-2r+1} \cdot \frac{m^r}{r!}$. If we plug this into (2), we get the hit probability of at least

$$\exp\left\{ \frac{m-r+1}{m-2r+1} \cdot \frac{m^r}{r!} \cdot \ln\left(1 - 2^{-r}\right) \right\} \tag{3}$$

for a single codeword per information set. By Proposition 3.2, there are at least $2^{mr-r(r-1)}$ such words, and so the cost for finding any one of them can be estimated to be at most

$$2^{-\frac{m-r+1}{m-2r+1} \cdot \frac{m^r}{r!} \cdot \log_2\left(1-2^{-r}\right) - mr + r(r-1)} \tag{4}$$

diagonalizations of the generator matrix. This rough estimate predicts, for example, that finding a minimum weight word in $\mathcal{R}(3, 11)$ would cost roughly 2^{37} diagonalizations, and thus finding such words is feasible in that case.

As expected and easily seen by comparing to real running times, the bound (4) is somewhat pessimistic, i.e., it overestimates the running time. For example, finding a minimum weight word in $\mathcal{R}(3, 11)$ needs only about 2^{17} diagonalizations in practice. More precise estimates are of course possible, but result in uglier formulas.

The other low weight finding instance operates on the shortened code. In practice the sampling turns out to be much easier, because of the lower rate and the weakened condition on the weight. A conservative estimate is easy to find. For example, one can show that there are at least $2^{mr-r(r-1)-(m+r^2-4r+2)}$ minimum weight words in the shortened code, and then apply (3) to get a bound similar to (4). The obtained bound is even weaker than (4), though: It does not take into account the lower rate of the code, nor the fact that words do not have to be strictly minimum weight in this case.

Asymptotic analysis. Asymptotically, the running time for the algorithm is

$$O(\mathrm{poly}(n)) \cdot e^{O(\mathrm{poly}(\log(n)))} \tag{5}$$

for any fixed value of r.

To see this, we start again with (3). Using the assumption that $r/m \to 0$, and writing the expression in terms of the block length $n = 2^m$ instead of m, we get that this probability behaves like

$$\exp\left\{-\log_2(n)^r C_r(1 + o(1))\right\},$$

where C_r is a constant depending only on r. This time, we assume there is just a single minimum weight codeword, and thus a conservative estimate for the number of trials to find a minimum weight word is

$$C_{\mathrm{lw}} := \exp\left\{\log_2(n)^r C_r(1 + o(1))\right\}. \tag{6}$$

We take C_{lw} as the cost for both of the low weight sampling instances, deferring justification of this to A.1. Using the fact that only a polynomial (in n) number of samplings is needed, we conclude that (5) is is indeed a bound for the running time of the algorithm.

For large r, the numbers get very large. That is not an artifact: If the code is not sufficiently low-rate, then finding minimum weight becomes a very hard problem, rendering the attack infeasible.

4.5 Experimental running time

To check the real-life behaviour, we ran our algorithm for different parameters on a 2.4GHz PC. Our implementation uses rather simple low weight word finding algorithms, and not elaborate ones like, e.g., the ones described in [1]. The average running-times for ten runs were:[2]

	$r=2$	$r=3$	$r=4$
$m=5$ $(n=32)$	<0.01s		
$m=6$ $(n=64)$	<0.01s		
$m=7$ $(n=128)$	0.02s	5.261s	
$m=8$ $(n=256)$	0.081s	2.059s	
$m=9$ $(n=512)$	0.448s	3.462s	176.914s
$m=10$ $(n=1024)$	2.46s	26.6s	82197.4s
$m=11$ $(n=2048)$	18.34s	1192.71s	no try

As predicted by the analysis, the performance degrades quickly with larger r. This does indeed exhibit a limit of our attack, but note that since the performance of Reed-Muller codes degrades with large r, choosing such values would very likely open the doors to other attacks.

The $(r=3, m=7)$-case is an anomaly of our implementation; we have decided to leave the high numbers for consistency reasons.

Acknowledgement.

We would like to thank Gérard Maze, Arjen Lenstra and Martijn Stam for helpful discussions and reviewing early draft versions of this paper; their help and support has been invaluable in the process of writing up this paper.

References

1. A. Canteaut, F. Chabaut, *A new algorithm for finding minimum-weight words in a linear code: application to primitive narrow-sense BCH-codes of length 511*, 1998, IEEE Transactions on Information Theory, 44(1):367-378
2. I. Dumer, K. Shabunov, *Soft-decision decoding of Reed-Muller codes: a simplified algorithm*, 2006, IEEE Transactions on Information Theory 52(3): 954-963
3. W. Cary Huffman, V. Pless, *Fundamentals of Error-Correcting Codes*, 2003, Cambridge University Press
4. T. Kasami, N. Tokura, *On the Weight Structure of Reed-Muller Codes*, 1970, IEEE Transactions on Information Theory, 16(6): 752-759
5. F. J. MacWilliams, N. J. A. Sloane, *The Theory of Error-Correcting Codes*, 1978, North-Holland
6. R. J. McEliece, *A public key cryptosystem based on algebraic coding theory*, DSN progress report, 42-44:114-116, 1978
7. H. Niederreiter, *Knapsack-Type Cryptosystems and Algebraic Coding Theory*, Problems of Control and Information Theory, 15(2):159–166, 1986.

[2] We only look at $m > 2r$; since in the other case, the attack can be carried out more efficiently on the dual code.

8. V. M. Sidelnikov, *A public-key cryptosystem based on binary Reed-Muller codes*, Discrete Mathematics and Applications, 4 No. 3, 1994
9. V. M. Sidelnikov, S. O. Shestakov, *On insecurity of cryptosystems based on generalized Reed-Solomon codes*, Discrete Mathematics and Applications, 2, No. 4:439–444, 1992
10. N. Sendrier, *On the Structure of a randomly permuted concatenated code*, EUROCODE 94, October 1994.
11. N. Sendrier, *Finding the permutation between equivalent codes: the support splitting algorithm*, IEEE Transactions on Information Theory, 46(4):1193-1203, 2000

A APPENDIX

This appendix contains detailed proofs that have been omitted in the paper, as well as some other comments we do not consider vital for the understanding of the paper.

A.1 The low weight word problem in the shortened code

In the running-time analysis, we based our running-time estimates on estimates on the difficulty of the low weight word finding problem.

It should be noted that two different low weight word finding problems have to be solved; the minimum weight word finding problem in the Reed-Muller code, and the low weight word finding problem in the shortened Reed-Muller code.

We assumed that the low weight word finding problem in the shortened code is easier than the minimum weight word finding algorithm in the original code. The reason for this is that first the weight restriction is relieved, and second, the shortened code has lower rate, as we will now show.

The shortened code has lower rate. The correctness of our running time analysis depends on the fact that the shortened code has lower rate. This is not an obvious fact, since, even though the dimension clearly has to decrease, the length does so too. We prove the assertion in this section.

We write $P_{r,m}$ the number of linearly independent parity checks that $\mathcal{R}(r,m)$ has, i.e., $P_{r,m} = \dim(\mathcal{R}(r,m)^{\perp})$.

We can use (1) to deduce the number its number of linearly independent parity checks in the shortened code, and get that there are

$$\sum_{i=1}^{r} \binom{r}{i} P_{r-i,m-r}$$

of them. We can deduce a similar formula for $P_{r,m}$ itself using an induction on the equality $P_{r,m} = P_{r,m-1} + P_{r-1,m-1}$. We then get that for any $\ell \leq r$, we have

$$P_{r,m} = \sum_{i=0}^{\ell} \binom{\ell}{i} P_{r-i,m-\ell}.$$

Proposition A.1. *The shortened code (constructed in the section on the attack) has lower rate than the original code.*

Proof. We have to show that

$$\frac{\sum_{i=1}^{r} \binom{r}{i} P_{r-i,m-r}}{2^m - 2^{m-r}} > \frac{\sum_{i=0}^{r} \binom{r}{i} P_{r-i,m-r}}{2^m}.$$

Rearranging the terms, we get that this is equivalent to showing that

$$\frac{1}{1-2^{-r}} > \frac{\sum_{i=0}^{r} \binom{r}{i} P_{r-i,m-r}}{\sum_{i=1}^{r} \binom{r}{i} P_{r-i,m-r}},$$

or yet,

$$2^{-r} > \frac{P_{r,m-r}}{\sum_{i=0}^{r} \binom{r}{i} P_{r-i,m-r}}.$$

Let μ be the weighted average of the $P_{r-i,m-r}$, i.e.,

$$\mu = 2^{-r} \sum_{i=0}^{r} \binom{r}{i} P_{r-i,m-r}.$$

Then we see that we have to show

$$\mu > P_{r,m-r}.$$

Now this last equation is true because

$$P_{r,m-r} < P_{r-1,m-r} < \cdots < P_{0,m-r},$$

as implied by proposition 2.1. □

A.2 A brief note on the generalized Sidelnikov system

The paper [8] also proposes to use more than a single generator. The proposition is to juxtapose several differently scrambled generators, and then intermingle the separate blocks with a right-hand permutation matrix. If u is some small integer, R is a generator matrix of some $\mathcal{R}(r,m)$-matrix of dimension $k \times n$, $E_1, \ldots, E_u$ are random invertible matrices, and Γ is a $un \times un$ random permutation matrix, then the public key is of the form

$$|E_1 R, E_2 R, \ldots, E_u R|\Gamma,$$

which is the generator matrix of some $[un, k]$-code.

In fact, there is no added security using this when compared to the case $u = 1$. To see this, note that on the positions corresponding to $E_i R$, all the parity checks for $\mathcal{R}(r,m)$ are valid parity checks. So to recover the independent code blocks, it is enough to sample low weight parity checks and to mark the bits in their support as belonging to the same inner block. Doing this for not to many codewords should be enough to recover the block decomposition.

In general, it is hard to find low weight words, but not if the sought words are very low weight. Since $\mathcal{R}(r,m)^{\perp} = \mathcal{R}(m-r-1,m)$, the lowest-weight in the dual code is 2^{r+1}, which is indeed very low weight for the values of r of interest in practice.

So, in summary, breaking the general Sidelnikov system is roughly equivalent to recovering a single Reed-Muller code.