Security Bounds for the Design of Code-based Cryptosystems

Matthieu Finiasz[1] and Nicolas Sendrier[2]

[1] ENSTA
[2] INRIA, team-project SECRET

Abstract. Code-based cryptography is often viewed as an interesting "Post-Quantum" alternative to the classical number theory cryptography. Unlike many other such alternatives, it has the convenient advantage of having only a few, well identified, attack algorithms. However, improvements to these algorithms have made their effective complexity quite complex to compute. We give here some lower bounds on the work factor of idealized versions of these algorithms, taking into account all possible tweaks which could improve their practical complexity. The aim of this article is to help designers select durably secure parameters.
Keywords : computational syndrome decoding, information set decoding, generalized birthday algorithm.

Introduction

Code-based cryptography has received renewed attention with the recent interest for "Post-Quantum Cryptography" (see for instance [5]). Several new interesting proposals have been published in the last few months [20,18,3]. For those new constructions as well as for previously known code-based cryptosystems, precise parameters selection is always a sensitive issue. Most of the time the most threatening attacks are based on decoding algorithms for generic linear codes. There are two main families of algorithms, Information Set Decoding (ISD), and Generalized Birthday Algorithm (GBA). Each family being suited for some different parameter ranges.

ISD is part of the folklore of algorithmic coding theory and is among the most efficient techniques for decoding errors in an arbitrary linear code. One major step in the development of ISD for the cryptanalysis of the McEliece encryption scheme is Stern's variant [22] which mixes birthday attack with the traditional approach. A first implementation description [10], with several improvements, led to an attack of $2^{64.2}$ binary operations for the original McEliece parameters, that is decoding 50 errors in a code of length 1024 and dimension 524. More recently [6], a new implementation was proposed with several new improvements, with a binary workfactor of $2^{60.5}$. Furthermore, the authors report a real attack (with the original parameters) with a computational effort of about 2^{58} CPU cycles. The above numbers are accurate estimates of the real cost of a decoding attack. They involve several parameters that have to be optimized and furthermore, no close formula exists, making a precise evaluation rather difficult.

GBA was introduced by Wagner in 2002 [25] but was not specifically designed for decoding. Less generic version of this algorithm had already been used in the past for various cryptanalytic applications [9,11]. Its first successful use to cryptanalyse a code-based system is due to Coron and Joux [12]. In particular, this work had a significant impact for selecting the parameters of the FSB hash function [1].

Most previous papers on decoding attacks were written from the point of view of the attacker and were looking for upper bounds on the work factor of some specific implementation. One exception is the asymptotic analysis for ISD that has been recently presented in [8]. Here we propose a designer approach and we aim at providing tools to easily select secure parameters.

For both families, we present new idealized version of the algorithms, which encompass all variants and improvements known in cryptology as well as some new optimizations. This allows us to give easy to compute lower bounds for decoding attacks up to the state of the art.

We successively study three families of algorithms, first the "standard" birthday attack, then two evolutions of this technique, namely Stern's variant of information set decoding and Wagner's generalized birthday algorithm. In each case we propose very generic lower bounds on their complexity. Finally, we illustrate our work with case studies of some of the main code-based cryptosystems.

1 The Decoding Problem in Cryptology

Problem 1 (Computational Syndrome Decoding - CSD) *Given a matrix $H \in \{0,1\}^{r \times n}$, a word $s \in \{0,1\}^r$ and an integer $w > 0$, find a word $e \in \{0,1\}^n$ of Hamming weight $\leq w$ such that $eH^T = s$.*

We will denote $\mathrm{CSD}(H, s, w)$ an instance of that problem. It is equivalent to decoding w errors in a code with parity check matrix H. The decision problem associated with computational syndrome decoding, namely, Syndrome Decoding, is NP-complete [4].

This problem appears in code-based cryptography and for most systems it is the most threatening known attack (sometimes the security can be reduced to CSD alone [23,1]). Throughout the paper we will denote

$$W_{n,w} = \{e \in \{0,1\}^n \mid \mathrm{wt}(e) = w\}$$

the set of all binary words of length n and Hamming weight w. The instances of CSD coming from cryptology usually have solutions. Most of the time, this solution is unique. This is the case for public-key encryption schemes [17,21] or for identification schemes [23,24]. However, if the number w of errors is larger than the Gilbert-Varshamov distance[3] we may have a few, or even a large number, of solutions. Obtaining one of them is enough. This is the case for digital signatures [13] or for hashing [2,1].

[3] The Gilbert-Varshamov distance is the smallest integer d_0 such that ${n \atop d_0} \geq 2^r$.

2 The Birthday Attack for Decoding

We consider an instance $\mathrm{CSD}(H, s, w)$ of the computational syndrome decoding. If the weight w is even, we partition the columns of H in two subsets (a priori of equal size). For instance, let $H = (H_1 \mid H_2)$ and let us consider the sets $\mathcal{L}_1 = \{e_1 H_1^T \mid e_1 \in W_{n/2,w/2}\}$ and $\mathcal{L}_2 = \{s + e_2 H_2^T \mid e_2 \in W_{n/2,w/2}\}$. Any element of $\mathcal{L}_1 \cap \mathcal{L}_2$ provides a pair (e_1, e_2) such that $e_1 H_1 = s + e_2 H_2$ and $e_1 + e_2$ is a solution to $\mathrm{CSD}(H, s, w)$. This collision search has to be repeated $1/\mathrm{Pr}_{n,w}$ times on average where $\mathrm{Pr}_{n,w}$ is the probability that one of the solutions splits evenly between the left and right parts of H. Let $C_{n,r,w}$ denote the total number of columns sums we have to compute. If the solution is unique, we have[4]

$$\mathrm{Pr}_{n,w} = \frac{\binom{n/2}{w/2}^2}{\binom{n}{w}} \text{ and } C_{n,r,w} = \frac{|\mathcal{L}_1| + |\mathcal{L}_2|}{\mathrm{Pr}_{n,w}} = \frac{2\binom{n}{w}}{\binom{n/2}{w/2}} \approx 2\sqrt{\binom{n}{w}}\sqrt[4]{\frac{\pi w}{2}}$$

This number is close to the improvement expected when the birthday paradox can be applied (*i.e.* replacing an enumeration of N elements by an enumeration of $2\sqrt{N}$ elements). In this section, we will show that the factor $\sqrt[4]{\pi w/2}$ can be removed and that the formula often applies when w is odd. We will also provide cost estimations and bounds.

2.1 A Decoding Algorithm Using the Birthday Paradox

The algorithm presented in Table 1 generalizes the birthday attack for decoding presented above. For any fixed values of n, r and w this algorithm uses three parameters (to be optimized): an integer ℓ and two sets of constant weight words W_1 and W_2. The idea is to operate as much as possible with partial syndromes of size $\ell < r$ and to make the full comparison on r bits only when we have a partial match. Increasing the size of W_1 (and W_2) will lead to a better trade-off, ideally with a single execution of (MAIN LOOP).

Definition 1. *For any fixed value of n, r and w, we denote* $\mathrm{WF}_{\mathrm{BA}}(n, r, w)$ *the minimal binary work factor (average cost in binary operations) of the algorithm of Table 1 to produce a solution to* CSD*, for any choices of parameters W_1, W_2 and ℓ.*

An Estimation of the Cost. We will use the following assumptions (discussed in appendix):

(B1) For all pairs (e_1, e_2) examined in the algorithm, the sums $e_1 + e_2$ are uniformly and independently distributed in $W_{n,w}$.

[4] We use Stirling's formula to approximate factorials. The approximation we give is valid because $w \ll n$.

Table 1. Birthday decoding algorithm

For any fixed values of n, r and w, the following algorithm uses three parameters: an integer $\ell > 0$, $W_1 \subset W_{n,\lfloor w/2 \rfloor}$ and $W_2 \subset W_{n,\lceil w/2 \rceil}$. We denote by $h_\ell(x)$ the first ℓ bits of any $x \in \{0,1\}^r$.

```
procedure BirthdayDecoding
input: $H_0 \in \{0,1\}^{r \times n}$, $s \in \{0,1\}^r$
  repeat                                                   (MAIN LOOP)
    $P \leftarrow$ random $n \times n$ permutation matrix
    $H \leftarrow H_0 P$
    for all $e \in W_1$
      $i \leftarrow h_\ell(eH^T)$                          (BA 1)
      write$(e, i)$      // store $e$ in some data structure at index $i$
    for all $e_2 \in W_2$
      $i \leftarrow h_\ell(s + e_2 H^T)$                   (BA 2)
      $S \leftarrow$ read$(i)$      // extract the elements stored at index $i$
      for all $e_1 \in S$
        if $e_1 H^T = s + e_2 H^T$                         (BA 3)
          return $(e_1 + e_2)P^T$                          (SUCCESS)
```

(B2) The cost of the execution of the algorithm is approximatively equal to

$$\ell \cdot \sharp(\text{BA } 1) + \ell \cdot \sharp(\text{BA } 2) + K_0 \cdot \sharp(\text{BA } 3), \tag{1}$$

where K_0 is the cost for testing $e_1 H^T = s + e_2 H^T$ given that $h_\ell(e_1 H^T) = h_\ell(s + e_2 H^T)$ and $\sharp(\text{BA } i)$ is the expected number of execution of the instruction (BA i) before we meet the (SUCCESS) condition.

Proposition 1. *Under assumptions* **(B1)** *and* **(B2)**. *We have*[5]

$$\mathrm{WF}_{\mathrm{BA}}(n, r, w) \approx 2L \log(K_0 L) \quad \textit{with } L = \min\left(\sqrt{\binom{n}{w}}, 2^{r/2}\right)$$

and K_0 *is the cost for executing the instruction* (BA 3) *(* i.e. *testing* $eH^T = s$*).*

Remarks.

1. When $\binom{n}{w} > 2^r$, the cost will depend of the number of syndromes 2^r instead of the number of words of weight w. This corresponds to the case where w is larger than the Gilbert-Varshamov distance and we have multiple solutions. We only need one of those solutions and thus the size of the search space is reduced.
2. It is interesting to note the relatively low impact of K_0, the cost of the test in (BA 3). Between an extremely conservative lower bound of $K_0 = 2$, an extremely conservative upper bound of $K_0 = wr$ and a more realistic $K_0 = 2w$ the differences are very small.

[5] Here and after, "log" denotes the base 2 logarithm (and "ln" the Neperian logarithm).

3. In the case where w is odd and $\binom{n}{\lfloor w/2 \rfloor} < L$, the formula of Proposition 1 is only a lower bound. A better estimate would be

$$\mathrm{WF}_{\mathrm{BA}}(n,r,w) \approx 2L' \log\left(K_0 \frac{L^2}{L'}\right) \text{ with } L' = \frac{\binom{n}{\lfloor w/2 \rfloor}^2 + L^2}{2\binom{n}{\lfloor w/2 \rfloor}}. \qquad (2)$$

4. Increasing the size of $|W_1|$ (and $|W_2|$) can be easily and efficiently achieved by "overlapping" H_1 and H_2 (see the introduction of this section). More precisely, we take for W_1 all words of weight $w/2$ using only the n' first coordinates (with $n/2 < n' < n$). Similarly, W_2 will use the n' (or more) last coordinates.

2.2 Lower Bounds

As the attacker can make a clever choice of W_1 and W_2 which may contradict assumption **(B1)**, we do not want to use it for the lower bound. The result remains very close to the estimate of the previous sections except for the multiplicative constant which is $\sqrt{2}$ instead of 2.

Theorem 1. *For any fixed value of n, r and w, we have*

$$\mathrm{WF}_{\mathrm{BA}}(n,r,w) \geq \sqrt{2}L\log(K_0 L) \textit{ with } L = \min\left(\sqrt{\binom{n}{w}}, 2^{r/2}\right).$$

where K_0 is the cost for executing the instruction (BA 3).

3 Information Set Decoding (ISD)

We will consider here Stern's algorithm [22], which is the best known decoder for cryptographic purposes, and some of its implemented variants by Canteaut-Chabaud [10] and Bernstein-Lange-Peters [6]. Our purpose is to present a lower bound which takes all known improvements into account.

3.1 A New Variant of Stern's Algorithm

Following other works [15,16], J. Stern describes in [22] an algorithm to find a word of weight w in a binary linear code of length n and dimension k (and codimension $r = n - k$). The algorithm uses two additional parameters p and ℓ (both positive integers). We present here a generalized version which acts on the parity check matrix H_0 of the code (instead of the generator matrix). Table 2 describes the algorithm. The partial Gaussian elimination of H_0P consists in

Table 2. Generalized ISD algorithm

For any fixed values of n, r and w, the following algorithm uses four parameters: two integers $p > 0$ and $\ell > 0$ and two sets $W_1 \subset W_{k+\ell,\lfloor p/2 \rfloor}$ and $W_2 \subset W_{k+\ell,\lceil p/2 \rceil}$. We denote by $h_\ell(x)$ the last ℓ bits of any $x \in \{0,1\}^r$.

```
procedure ISDecoding
input: $H_0 \in \{0,1\}^{r \times n}$, $s_0 \in \{0,1\}^r$
  repeat                                                    (MAIN LOOP)
    $P \leftarrow$ random $n \times n$ permutation matrix
    $(H', U) \leftarrow$ PGElim($H_0 P$)        // partial elimination as in (3)
    $s \leftarrow s_0 U^T$
    for all $e \in W_1$
      $i \leftarrow h_\ell(eH'^T)$                                (ISD 1)
      write($e, i$)      // store $e$ in some data structure at index $i$
    for all $e_2 \in W_2$
      $i \leftarrow h_\ell(s + e_2 H'^T)$                           (ISD 2)
      $S \leftarrow$ read($i$)      // extract the elements stored at index $i$
      for all $e_1 \in S$
        if wt  $s + (e_1 + e_2)H'^T$  $= w - p$                    (ISD 3)
          return $(P, e_1 + e_2)$                             (SUCCESS)
```

finding U ($r \times r$ and non-singular) and H (and H') such that[6]

$$UH_0P = H = \begin{array}{c|c|c|} & r-\ell & k+\ell \\ \cline{2-3} & \begin{matrix} 1 & & \\ & \ddots & \\ & & 1 \end{matrix} & \\ \cline{2-2} \ell & 0 & H' \\ \cline{2-3} \end{array} \qquad (3)$$

where U is a non-singular $r \times r$ matrix. Let $s = s_0 U^T$. If e is a solution of CSD(H, s, w) then eP^T is a solution of CSD(H_0, s_0, w). Let (P, e') be the output of the algorithm, *i.e.*, wt$(s + e'H'^T) = w - p$, and let e'' be the first $r - \ell$ bits of $s + e'H'^T$, the word $e = (e'' \mid e')$ is a solution of CSD(H, s, w).

Definition 2. *For any fixed value of n, r and w, we denote* WF$_{\text{ISD}}(n, r, w)$ *the minimal binary work factor (average cost in binary operations) of the algorithm of Table 1 to produce a solution to* CSD*, for any choices of parameters ℓ, p, W_1 and W_2.*

3.2 Estimation of the Cost of the New Variant

To evaluate the cost of the algorithm we will assume that only the instructions (ISD i) are significant. This assumption is stronger than for the birthday attack,

[6] In the very unlikely event that the first $r - \ell$ columns are linearly dependent, we can change P.

because it means that the Gaussian elimination at the beginning of every (main loop) costs nothing. It is a valid assumption as we only want a lower bound. Moreover, most of the improvements introduced in [10,6] are meant to reduce the relative cost of the Gaussian elimination. We claim that within this "free Gaussian elimination" assumption any lower bound on the algorithm of Table 2 will apply on all the variants of [10,6]. Our estimations will use the following assumptions:

(I1) For all pairs (e_1, e_2) examined in the algorithm, the sums $e_1 + e_2$ are uniformly and independently distributed in $W_{k+\ell,p}$.

(I2) The cost of the execution of the algorithm is approximatively equal to

$$\ell \cdot \sharp(\textsc{isd}\ 1) + \ell \cdot \sharp(\textsc{isd}\ 2) + K_{w-p} \cdot \sharp(\textsc{isd}\ 3), \tag{4}$$

where K_{w-p} is the average cost for checking $\mathrm{wt}\left(s + (e_1 + e_2)H'^T\right) = w - p$ and $\sharp(\textsc{isd}\ i)$ is the expected number of executions of the instruction (isd i) before we meet the (success) condition.

Proposition 2. *Under assumptions* **(I1)** *and* **(I2)**. *If* $\binom{n}{w} < 2^r$ *(single solution) or if* $\binom{n}{w} > 2^r$ *(multiple solutions) and* $\binom{r}{w-p}\binom{k}{p} \ll 2^r$, *we have (we recall that* $k = n - r$*)*

$$\mathrm{WF}_{\mathrm{ISD}}(n, r, w) \approx \min_p \frac{2\ell \min\left(\binom{n}{w}, 2^r\right)}{\lambda \binom{r-\ell}{w-p} \sqrt{\binom{k+\ell}{p}}} \text{ with } \ell = \log\left(K_{w-p}\sqrt{\binom{k}{p}}\right)$$

with $\lambda = 1 - \mathrm{e}^{-1} \approx 0.63$. *If* $\binom{n}{w} > 2^r$ *(multiple solutions) and* $\binom{r}{w-p}\binom{k}{p} \geq 2^r$, *we have*

$$\mathrm{WF}_{\mathrm{ISD}}(n, r, w) \approx \min_p \frac{2\ell\, 2^{r/2}}{\sqrt{\binom{r-\ell}{w-p}}} \text{ with } \ell = \log\left(K_{w-p}\frac{2^{r/2}}{\sqrt{\binom{r}{w-p}}}\right).$$

Remarks.

1. For a given set of parameters the expected number of execution of (main loop) is $N = 1/(1 - \exp(-X))$ where $X = \binom{r+\ell}{w-p}\binom{k+\ell}{p} / \min(2^r, \binom{n}{w})$.
2. The second formula applies when $X > 1$, that is when the expected number of execution of (main loop) is (not much more than) one. In that case, as for the birthday attack, the best strategy is to use $W_2 = W_{k+\ell,\lceil p/2\rceil}$ (*i.e.* as large as possible) and W_1 is as small as possible but large enough to have only one execution of (main loop) with probability close to 1.
3. When $X \ll 1$, we have $N = 1/(1 - \exp(-X)) \approx 1/X$ and the first formula applies.
4. When $X < 1$, the first formula still gives a good lower bound. But it is less tight when X gets closer to 1.
5. When p is small and odd the above estimates for $\mathrm{WF}_{\mathrm{ISD}}$ are not always accurate. The adjustment is similar to what we have in (2) (see the remarks following the birthday decoder estimation). In practice, if $\binom{k+\ell}{\lfloor p/2\rfloor} < \sqrt{\binom{k+\ell}{p}}$ it is probably advisable to discard this odd value of p.

6. We use the expression $\ell = \log\left(K_{w-p}L_p(0)\right)$ for the optimal value of ℓ (where $L_p(\ell) = \sqrt{\binom{k+\ell}{p}}$ or $L_p(\ell) = 2^{r/2}/\sqrt{\binom{r-\ell}{w-p}}$ respectively in the first case or in the second case of the Proposition). In fact a better value would be a fixpoint of the mapping $\ell \mapsto L_p(\ell)$. In practice $L_p(0)$ is a very good approximation.

3.3 Gain Compared with Stern's Algorithm

Stern's algorithm corresponds to a complete Gaussian elimination and to a particular choice of W_1 and W_2 in the algorithm of Table 2. A full Gaussian elimination is applied to the permuted matrix H_0P and we get U and H' such that:

$$UH_0P = H = \begin{array}{|c|cc|l} \multicolumn{1}{c}{r} & \multicolumn{2}{c}{k} & \\ \cline{1-3} 1 & \multicolumn{2}{c|}{} & \\ & \multicolumn{2}{c|}{H'} & \\ \ddots & \multicolumn{2}{c|}{} & \\ \cline{2-3} 1 & \multicolumn{1}{c|}{H_1} & \multicolumn{1}{c|}{H_2} & \ell \\ \cline{1-3} \end{array} \tag{5}$$

The ℓ-bit collision search is performed on k columns, moreover p is always even and W_1 and W_2 will use $p/2$ columns of H_1 and H_2. The variants presented in [10,6] consist in reducing the cost of the Gaussian elimination, or, for the same H', to use different "slices" $(H_1 \mid H_2)$ of ℓ rows. All other improvements lead to an operation count which is close to what we have in (4). The following formula, obtained with the techniques of the previous section, gives a tight lower bound all those variants.

$$\mathrm{WF}_{\mathrm{Stern}}(n,r,w) \approx \min_p \frac{2\ell\binom{n}{w}}{\binom{r-\ell}{w-p}\binom{k/2}{p/2}} \text{ with } \ell = \log\left(K_{w-p}\binom{k/2}{p/2}\right).$$

The gain of the new version of ISD is $\approx \lambda\sqrt[4]{\pi p/2}$ which is rather small in practice and correspond to the improvement of the "birthday paradox" part of the algorithm.

4 Generalized Birthday Algorithm (GBA)

4.1 General Principle

The generalized birthday technique is particularly efficient for solving Syndrome Decoding-like problems with a large number of solutions. Suppose one has to solve the following problem:

Problem 2 *Given a function $f : \mathbb{N} \mapsto \{0,1\}^r$ and an integer a, find a set of 2^a indexes x_i such that:*

$$\bigoplus_{i=0}^{2^a-1} f(x_i) = 0.$$

In this problem, f will typically return the x_i-th column of a binary matrix H. Note that, here, f is defined upon an infinite set, meaning that there are an infinity of solutions. To solve this problem, the Generalized Birthday Algorithm (GBA) does the following:

- build 2^a lists $L_0, \ldots, L_{2^a-1}$, each containing $2^{\frac{r}{a+1}}$ different vectors $f(x_i)$
- pairwise merge lists L_{2j} and L_{2j+1} to obtain 2^{a-1} lists L'_j of XORs of 2 vectors $f(x_i)$. Only keep XORs of 2 vectors starting with $\frac{r}{a+1}$ zeros. On average, the lists L'_j will contain $2^{\frac{r}{a+1}}$ elements.
- pairwise merge the new lists L'_{2j} and L'_{2j+1} to obtain 2^{a-2} lists L''_j of XORs of 4 vectors $f(x_i)$. Only keep XORs of 4 vectors starting with $2\frac{r}{a+1}$ zeros. On average, the lists L''_j will still contain $2^{\frac{r}{a+1}}$ elements.
- continue these merges until only 2 lists remain. These 2 lists will be composed of $2^{\frac{r}{a+1}}$ XORs of 2^{a-1} vectors $f(x_i)$ starting with $(a-1)\frac{r}{a+1}$ zeros.
- as only $2\frac{r}{a+1}$ bits of the previous vectors are non-zero, a simple application of the standard birthday technique is enough to obtain 1 solution (on average).

As all the lists manipulated in this algorithm are of the same size, the complexity of the algorithm is easy to compute: $2^a - 1$ merge operations have to be performed, each of them requiring to sort a list of size $2^{\frac{r}{a+1}}$. The complexity is thus $O(2^a \frac{r}{a} 2^{\frac{r}{a+1}})$. For simplicity we will only consider a lower bound of the effective complexity of the algorithm: if we denote by L the size of the largest list in the algorithm, the complexity is lower-bounded by $O(L \log L)$. this gives a complexity of $O(\frac{r}{a+1} 2^{\frac{r}{a+1}})$

Minimal Memory Requirements. The minimal memory requirements for this algorithm are not as easy to compute. If all the lists are chosen to be of the same size (as in the description of the algorithm we give), then it is possible to compute the solution by storing at most a lists at a time in memory. This gives us a memory complexity of $O(a 2^{\frac{r}{a+1}})$. However, the starting lists can also be chosen of different sizes so as to store only smaller lists.

In practice, for each merge operation, only one of the two lists has to be stored in memory, the second one can always be computed on the fly. As a consequence, looking at the tree of all merge operations (see Fig. 1), half the lists of the tree can be computed on the fly (the lists in dashed line circles). Let $L = 2^{\frac{r}{a+1}}$ and suppose one wants to use the Generalized Birthday Algorithm storing only lists of size $\frac{L}{\lambda}$ for a given λ. Then, in order to get, on average, a single solution in the end, the lists computed on the fly should be larger. For instance, in the example of Fig. 1 one should have:

- $|L''_1| = \lambda L$, $|L'_3| = \lambda^2 L$, and $|L_7| = \lambda^3 L$,
- $|L'_1| = L$ and $|L_3| = \lambda L$,
- $|L_1| = L$ and $|L_5| = L$.

In the general case this gives us a time/memory tradeoff when using GBA: one can divide the memory complexity by λ at the cost of an increase in time complexity by a factor λ^a. However, many other combinations are also possible depending on the particular problem one has to deal with.

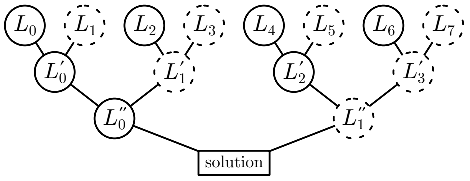


Fig. 1. Merge operations in the Generalized Birthday Algorithm. All lists in dashed line circles can be computed on the fly.

4.2 GBA Under Constraints

In the previous section, we presented a version of GBA where the number of vectors available was unbounded and where the number of vectors to XOR was a power of 2. In practice, when using GBA to solve instances of the CSD problem only n different r-bit vectors are available and w can be any number. We thus consider an idealized version of GBA so as to bound the complexity of "real world" GBA. The bounds we give are not always very tight. See for instance [7] for the analysis of a running implementation of GBA under realistic constraints.

If w is not a power of 2, some of the starting lists L_j should contain vectors $f(x_i)$ and others XORs of 2 or more vectors $f(x_i)$. We consider that the starting lists all contain XORs of $\frac{w}{2^a}$ vectors $f(x_i)$, even if this is not an integer. This will give the most time efficient algorithm, but will of course not be usable in practice.

The length of the matrix n limits the size of the starting lists. For GBA to find one solution on average, one needs lists L_j of size $2^{\frac{r}{a+1}}$. As the starting lists contain XORs of $\frac{w}{2^a}$ vectors, we need $\binom{n}{\frac{w}{2^a}} \geq 2^{\frac{r}{a+1}}$. However, this constraint on a is not sufficient: if all the starting lists contain the same vectors, all XORs will be found many times and the probability of success will drop. To avoid this, we need lists containing different vectors and this can be done by isolating the first level of merges.

- first we select 2^{a-1} distinct vectors s_j of a bits such that $\bigoplus s_j = 0$.
- then we pairwise merge lists L_{2j} and L_{2j+1} to obtain lists L_j' containing elements having their a first bits equal to s_j.

After this first round, we have 2^{a-1} lists of XORs of $\frac{2w}{2^a}$ vectors such that, if we XOR the a first bits of one element from each list we obtain 0. Also, all the lists contain only distinct elements, which means we are back in the general case of GBA, except we now have 2^{a-1} lists of vectors of length $r-a$. These lists all have a maximum size $L = \frac{1}{2^a}\binom{n}{\frac{2w}{2^a}}$ and can be obtained from starting lists L_j of size $\sqrt{\binom{n}{\frac{2w}{2^a}}}$ (see Sect. 2). We get the following constraint on a:

$$\frac{1}{2^a}\binom{n}{\frac{2w}{2^a}} \geq 2^{\frac{r-a}{a}}. \qquad (6)$$

In practice, after the first level of merges we are not exactly in the general case of GBA: if, for example, $s_0 \oplus s_1 = s_2 \oplus s_3$, after the second merges, lists L''_0 and L''_1 would contain exactly the same elements. This can be avoided by using another set of target values s'_j such that $\bigoplus s'_j = 0$ for the second level of merges (as for the first level) and so on for the subsequent levels of merges (except the last two levels).

Using Non-Integer Values for *a*. Equation (6) determines the largest possible value of a that can be used with GBA. For given n and r, if w varies, the complexity of the algorithm will thus have a stair-like shape (see Fig. 2(a)). The left-most point of each step corresponds to the case where Equation (6) is an equality. However, when it is not an equality, it is possible to gain a little: instead of choosing values s_j of a bits one can use slightly larger values and thus start the second level of merge with shorter vectors. This gives a broken-line complexity curve (see Fig. 2(b)). This is somehow similar to what Minder and Sinclair denote by "extended k-tree algorithm" [19]. In practice, this is almost equivalent to using non-integer values for a (see Fig. 2(c)). We will thus assume that in GBA, a is a real number, chosen such that Equation (6) is an equality.

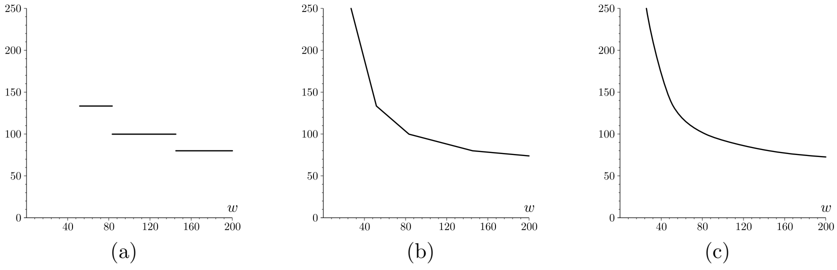


Fig. 2. Logarithm of the complexity of the Generalized Birthday Algorithm for given n and r when w varies. (a) with no optimization, (b) when the lists are initialized with shortened vectors, and (c) when a is not an integer.

Proposition 3. *We can lower bound the binary work factor* $\mathrm{WF}_{\mathrm{GBA}}(n,r,w)$ *of GBA applied to solving an instance of CSD with parameters* (n,r,w) *by:*

$$\mathrm{WF}_{\mathrm{GBA}}(n,r,w) \geq \frac{r-a}{a} 2^{\frac{r-a}{a}}, \textit{ with } a \textit{ such that } \frac{1}{2^a}\binom{n}{\frac{2w}{2^a}} = 2^{\frac{r-a}{a}}.$$

Note that this gives us a bound on the minimal time complexity of GBA but does not give any bound on the memory complexity of the algorithm. Also, this bound is computed using an idealized version of the algorithm: one should not expect to achieve such a complexity in practice, except in some cases where a is an integer and w a power of 2.

5 Case Studies

Now that we have given some bounds on the complexities of the best algorithms to solve CSD problems, we propose to study what happens when using them to attack existing constructions.

Note that in this section, as in the whole paper, we only consider the resistance to decoding attacks. Code-based cryptosystems may also be vulnerable to structural attacks. However, no efficient structural attack is known for binary Goppa codes (McEliece encryption and CFS signature) or for prime order random quasi-cyclic codes (FSB hash function).

5.1 Attacking the McEliece Cryptosystem

In the McEliece [17] and Niederreiter [21] cryptosystems the security relies on two different problems: recovering the private key from the public key and decrypting an encrypted message. Decrypting consists in finding an error pattern e of weight w, such that $e \times H^T = c$ where H is a binary matrix derived from the public key and c is a syndrome derived from the encrypted message one wants to decrypt. Here, we suppose that the structural attack consisting in recovering the private key is infeasible and can assume that H is a random binary matrix. Decryption thus consists in solving an instance of the CSD problem where one knows that one and only one solution exists.

Having a single solution rules out any attempt to use GBA, or at least, any attempt to use GBA would consist in using the classical birthday attack. For this reasons the best attacks against the McEliece and Niederreiter cryptosystems are all based on ISD. Table 3 gives the work factors we obtain using our bound from Sect. 3. For the classical McEliece parameters $(10, 50)$ this bound can be compared to the work factors computed by non-idealized algorithms. Canteaut and Chabaud [10] obtained a work factor of $2^{64.2}$ and Bernstein, Lange and Peters [6] a work factor of $2^{60.5}$. As one can see, the gap between our bound and their complexities is very small indicating two things:

- our bound on ISD is tight when evaluating the practical security of some McEliece parameters,
- the best ISD-based algorithms are sufficiently advanced to make our assumption that Gaussian elimination is free almost realistic. Almost no margin is left for these techniques to improve and better attacks will need to introduce new methods.

5.2 Attacking the CFS Signature Scheme

The attack we present here is due to Daniel Bleichenbacher, but was never published. We present what he explained through private communication including a few additional details.

The CFS signature scheme [13] is based on the Niederreiter cryptosystem: signing a document requires to hash it into a syndrome and then try to decode

Table 3. Work factors for the ISD lower-bound we computed for some typical McEliece/Niederreiter parameters. The code has length $n = 2^m$ and codimension $r = mw$ and corrects w errors.

(m, w)	optimal p	optimal ℓ	binary work factor
$(10, 50)$	4	22	$2^{59.9}$
$(11, 32)$	6	33	$2^{86.8}$
$(12, 41)$	10	54	$2^{128.5}$

this syndrome. However, for a Goppa code correcting w errors, only a fraction $\frac{1}{w!}$ of the syndromes are decodable. Thus, a counter is appended to the message and the signer tries successive counter values until one hash is decodable. The signature consists of both the error pattern of weight w corresponding to the syndrome and the value of the counter giving this syndrome.

Attacking this construction consists in forging a valid signature for a chosen message. One must find a matching counter and error pattern for a given document. This looks a lot like a standard CSD problem instance. However, here there is one major difference with the case of McEliece or Niederreiter: instead of having one instance to solve, one now needs to solve one instance among many instances. One chooses a document and hashes it with many different counters to obtain many syndromes: each syndrome corresponds to a different instance. It has no importance which instance is solved, each of them can give a valid "forged" signature.

For ISD algorithms, having multiple instances available is of little help, however, for GBA, this gives us one additional list. Even though Goppa code parameters are used and an instance has less than a solution on average, this additional list makes the application of GBA with $a = 2$ possible. This will always be an "unbalanced" GBA working as follows:

- first, build 3 lists L_0, L_1, and L_2 of XORs of respectively w_0, w_1 and w_2 columns of H (with $w = w_0 + w_1 + w_2$). These lists can have a size up to $\binom{n}{w_i}$ but smaller sizes can be used,
- merge the two lists L_0 and L_1 into a list L_0' of XORs of $w_0 + w_1$ columns of H, keeping only those starting with λ zeros (we will determine the optimal choice for λ later). L_0' contains $\frac{1}{2^\lambda}\binom{n}{w_0+w_1}$ elements on average.
- All the following computations are done on the fly and additional lists do not have to be stored. Repeat the following steps:
  - choose a counter and compute the corresponding document hash (an element of the virtual list L_3),
  - XOR this hash with all elements of L_2 matching on the first λ bits (to obtain elements of the virtual list L_1'),
  - look up each of these XORs in L_0': any complete match gives a valid signature.

The number L of hashes one will have to compute on average is such that:

$$\frac{1}{2^\lambda}\binom{n}{w_0 + w_1} \times \frac{L}{2^\lambda}\binom{n}{w_2} = 2^{r-\lambda} \quad \Leftrightarrow \quad L = \frac{2^{r+\lambda}}{\binom{n}{w_0+w_1}\binom{n}{w_2}}.$$

The memory requirements for this algorithm correspond to the size of the largest list stored. In practice, the first level lists L_i can be chosen so that L'_0 is always the largest, and the memory complexity is $\frac{1}{2^\lambda}\binom{n}{w_0+w_1}$. The time complexity corresponds to the size of the largest list manipulated: $\max(\frac{1}{2^\lambda}\binom{n}{w_0+w_1}, L, \frac{L}{2^\lambda}\binom{n}{w_2})$. The optimal choice is always to choose $w_0 = \lceil \frac{w}{3} \rceil$, $w_2 = \lfloor \frac{w}{3} \rfloor$, and $w_1 = w - w_0 - w_2$. Then, two different cases can occur: either L'_1 is the largest list, or one of L'_0 and L_3 is. If L'_1 is the largest, we choose λ so as to have a smaller list L'_0 and so a smaller memory complexity. Otherwise, we choose λ so that L'_0 and L_3 are of the same size to optimize the time complexity. Let $\mathcal{T}$ be the size of the largest list we manipulate and $\mathcal{M}$ the size of the largest list we store. The algorithm has time complexity $O(\mathcal{T}\log\mathcal{T})$ and memory complexity $O(\mathcal{M}\log\mathcal{M})$ with:

$$\begin{cases} \text{if } \frac{2^r}{\binom{n}{w-\lfloor w/3 \rfloor}} \geq \sqrt{\frac{2^r}{\binom{n}{\lfloor w/3 \rfloor}}} \text{ then } \mathcal{T} = \frac{2^r}{\binom{n}{w-\lfloor w/3 \rfloor}} \text{ and } \mathcal{M} = \frac{\binom{n}{w-\lfloor w/3 \rfloor}}{\binom{n}{\lfloor w/3 \rfloor}}, \\ \text{else} \qquad\qquad\qquad\qquad\qquad\quad \mathcal{T} = \mathcal{M} = \sqrt{\frac{2^r}{\binom{n}{\lfloor w/3 \rfloor}}}. \end{cases}$$

This algorithm is realistic in the sense that only integer values are used, meaning that effective attacks should have time/memory complexities close to those we present in Table 4. Of course, for a real attack, other time/memory tradeoffs might be more advantageous, resulting in other choices for λ and the w_i.

Table 4. Time/memory complexities of Bleichenbacher's attack against the CFS signature scheme. The parameters are Goppa code parameters so $r = mw$ and $n = 2^m$.

	$w = 8$	$w = 9$	$w = 10$	$w = 11$	$w = 12$
$m = 15$	$2^{51.0}/2^{51.0}$	$2^{60.2}/2^{43.3}$	$2^{63.1}/2^{55.9}$	$2^{67.2}/2^{67.2}$	$2^{81.5}/2^{54.9}$
$m = 16$	$2^{54.1}/2^{54.1}$	$2^{63.3}/2^{46.5}$	$2^{66.2}/2^{60.0}$	$2^{71.3}/2^{71.3}$	$2^{85.6}/2^{59.0}$
$m = 17$	$2^{57.2}/2^{57.2}$	$2^{66.4}/2^{49.6}$	$2^{69.3}/2^{64.2}$	$2^{75.4}/2^{75.4}$	$2^{89.7}/2^{63.1}$
$m = 18$	$2^{60.3}/2^{60.3}$	$2^{69.5}/2^{52.7}$	$2^{72.4}/2^{68.2}$	$2^{79.5}/2^{79.5}$	$2^{93.7}/2^{67.2}$
$m = 19$	$2^{63.3}/2^{63.3}$	$2^{72.5}/2^{55.7}$	$2^{75.4}/2^{72.3}$	$2^{83.6}/2^{83.6}$	$2^{97.8}/2^{71.3}$
$m = 20$	$2^{66.4}/2^{66.4}$	$2^{75.6}/2^{58.8}$	$2^{78.5}/2^{76.4}$	$2^{87.6}/2^{87.6}$	$2^{101.9}/2^{75.4}$
$m = 21$	$2^{69.5}/2^{69.5}$	$2^{78.7}/2^{61.9}$	$2^{81.5}/2^{80.5}$	$2^{91.7}/2^{91.7}$	$2^{105.9}/2^{79.5}$
$m = 22$	$2^{72.6}/2^{72.6}$	$2^{81.7}/2^{65.0}$	$2^{84.6}/2^{84.6}$	$2^{95.8}/2^{95.8}$	$2^{110.0}/2^{83.6}$

5.3 Attacking the FSB Hash Function

FSB [1] is a candidate for the SHA-3 hash competition. The compression function of this hash function consists in converting the input into a low weight word and then multiplying it by a binary matrix H. This is exactly a syndrome computation and inverting this compression function requires to solve an instance of the CSD problem. Similarly, finding a collision on the compression function requires to find two low weight words having the same syndrome, that is, a word of twice the Hamming weight with a null syndrome. In both cases, the security of the compression function (and thus of the whole hash function) can be reduced to the hardness of solving some instances of the CSD problem. For inversion (or second preimage), the instances are of the form $\mathrm{CSD}(H, w, s)$ and, for collision, of the form $\mathrm{CSD}(H, 2w, 0)$.

Compared to the other code-based cryptosystems we presented, here, the number of solutions to these instances is always very large: we are studying a compression function, so there are a lot of collisions, and each syndrome has a lot of inverses. For this reason, both ISD and GBA based attacks can be used. Which of the two approaches is the most efficient depends on the parameters. However, for the parameters proposed in [1], ISD is always the best choice for collision search and GBA the best choice for inversion (or second preimage). Table 5 contains the attack complexities given by our bounds for the proposed FSB parameters. As you can see, the complexities obtained with GBA for inversion are lower than the standard security claim. Unfortunately this does not give an attack on FSB for many reasons: the version of GBA we consider is idealized and using non-integer values of a is not practical, but most importantly, the input of the compression of FSB is not any word of weight w, but only regular words, meaning that the starting lists for GBA will be much smaller in practice, yielding a smaller a and higher complexities.

Table 5. Complexities of the ISD and GBA bounds we propose for the official FSB parameters.

				inversion		collision	
	n	r	w	ISD	GBA	ISD	GBA
FSB_{160}	5×2^{18}	640	80	$2^{211.1}$	$2^{156.6}$	$2^{100.3}$	$2^{118.7}$
FSB_{224}	7×2^{18}	896	112	$2^{292.0}$	$2^{216.0}$	$2^{135.3}$	$2^{163.4}$
FSB_{256}	2^{21}	1 024	128	$2^{330.8}$	$2^{245.6}$	$2^{153.0}$	$2^{185.7}$
FSB_{384}	23×2^{16}	1 472	184	$2^{476.7}$	$2^{360.2}$	$2^{215.5}$	$2^{268.8}$
FSB_{512}	31×2^{16}	1 984	248	$2^{687.8}$	$2^{482.1}$	$2^{285.6}$	$2^{359.3}$

Conclusion

In this article we have reviewed the two main families of algorithms for solving instances of the CSD problem. For each of these we have discussed possible tweaks and described idealized versions of the algorithms covering those tweaks. The work factors we computed for these idealized versions are lower bounds on the effective work factor of existing real algorithms, but also on the future improvements that could be implemented. Solving CSD more efficiently than these bounds would require to introduce new techniques, never applied to code-based cryptosystems.

For these reasons, the bounds we give can be seen as a tool one can use to select parameters for code-based cryptosystems. We hope they can help other designers choose durable parameters with more ease.

References

1. D. Augot, M. Finiasz, Ph. Gaborit, S. Manuel, and N. Sendrier. SHA-3 proposal: FSB. Submission to the SHA-3 NIST competition, 2008.
2. D. Augot, M. Finiasz, and N. Sendrier. A family of fast syndrome based cryptographic hash function. In E. Dawson and S. Vaudenay, editors, *Progress in Cryptology - Mycrypt 2005*, number 3715 in LNCS, pages 64–83. Springer-Verlag, 2005.
3. T. Berger, P.-L. Cayrel, P. Gaborit, and A. Otmani. Reducing key length of the mceliece cryptosystem. In B. Preneel, editor, *AFRICACRYPT 2009*, LNCS. Springer-Verlag, 2009. to appear.
4. E. R. Berlekamp, R. J. McEliece, and H. C. van Tilborg. On the inherent intractability of certain coding problems. *IEEE Transactions on Information Theory*, 24(3), May 1978.
5. D. Bernstein, J. Buchmann, and J. Ding, editors. *Post-Quantum Cryptography.* Springer, 2008.
6. D. Bernstein, T. Lange, and C. Peters. Attacking and defending the McEliece cryptosystem. In J. Buchmann and J. Ding, editors, *Post-Quantum Cryptography*, number 5299 in LNCS, pages 31–46. Springer-Verlag, 2008.
7. D. J. Bernstein, T. Lange, C. Peters, R. Niederhagen, and P. Schwabe. Implementing wagner's generalized birthday attack against the sha-3 candidate fsb. Cryptology ePrint Archive, Report 2009/292, 2009. http://eprint.iacr.org/.
8. D. J. Bernstein, T. Lange, C. Peters, and H. van Tilborg. Explicit bounds for generic decoding algorithms for code-based cryptography. In *Pre-proceedings of WCC 2009*, pages 168–180, 2009.
9. P. Camion and J. Patarin. The knapsack hash function proposed at crypto'89 can be broken. In D. W. Davies, editor, *Advances in Cryptology - EUROCRYPT'91*, number 547 in LNCS, pages 39–53. Springer-Verlag, 1991.
10. A. Canteaut and F. 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, January 1998.
11. P. Chose, A. Joux, and M. Mitton. Fast correlation attacks: An algorithmic point of view. In L. R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 209–221. Springer, 2002.
12. J.-S. Coron and A. Joux. Cryptanalysis of a provably secure cryptographic hash function. Cryptology ePrint Archive, 2004. http://eprint.iacr.org/2004/013/.
13. N. Courtois, M. Finiasz, and N. Sendrier. How to achieve a McEliece-based digital signature scheme. In C. Boyd, editor, *Asiacrypt 2001*, number 2248 in LNCS, pages 157–174. Springer-Verlag, 2001.
14. M. Finiasz and N. Sendrier. Security bounds for the design of code-based cryptosystems. Cryptology ePrint Archive, Report 2009/414, 2009. http://eprint.iacr.org/.
15. P. J. Lee and E. F. Brickell. An observation on the security of McEliece's public-key cryptosystem. In C. G. Günther, editor, *Advances in Cryptology - EUROCRYPT'88*, number 330 in LNCS, pages 275–280. Springer-Verlag, 1988.

16. J. S. Leon. A probabilistic algorithm for computing minimum weights of large error-correcting codes. *IEEE Transactions on Information Theory*, 34(5):1354–1359, September 1988.
17. R. J. McEliece. A public-key cryptosystem based on algebraic coding theory. *DSN Prog. Rep.,* Jet Prop. Lab., California Inst. Technol., Pasadena, CA, pages 114–116, January 1978.
18. C. Aguilar Melchor, P.-L. Cayrel, and P. Gaborit. A new efficient threshold ring signature scheme based on coding theory. In J. Buchmann and J. Ding, editors, *PQCrypto*, number 5299 in LNCS, pages 1–16. Springer-Verlag, 2008.
19. L. Minder and A. Sinclair. The extended k-tree algorithm. In C. Mathieu, editor, *Proceedings of SODA 2009*, pages 586–595. SIAM, 2009.
20. R. Misoczki and P. S. L. M. Barreto. Compact McEliece keys from Goppa codes. Cryptology ePrint Archive, Report 2009/187, 2009. http://eprint.iacr.org/.
21. H. Niederreiter. Knapsack-type crytosystems and algebraic coding theory. *Prob. Contr. Inform. Theory*, 15(2):157–166, 1986.
22. J. Stern. A method for finding codewords of small weight. In G. Cohen and J. Wolfmann, editors, *Coding theory and applications*, number 388 in LNCS, pages 106–113. Springer-Verlag, 1989.
23. J. Stern. A new identification scheme based on syndrome decoding. In D. R. Stinson, editor, *Advances in Cryptology - CRYPTO'93*, number 773 in LNCS, pages 13–21. Springer-Verlag, 1993.
24. P. Véron. A fast identification scheme. In *IEEE Conference, ISIT'95*, page 359, Whistler, BC, Canada, September 1995.
25. D. Wagner. A generalized birthday problem. In M. Yung, editor, *CRYPTO'02*, number 2442 in LNCS, pages 288–303. Springer-Verlag, 2002.

A Comments on the assumptions

We have assumed the following in Sect. 2:

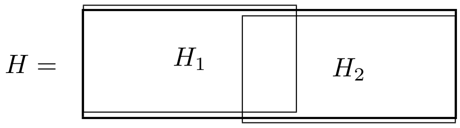
(B1) For all pairs (e_1, e_2) examined in the algorithm, the sums $e_1 + e_2$ are uniformly and independently distributed in $W_{n,w}$.

(B2) The cost of the execution of the algorithm is approximatively equal to

$$\ell \cdot \sharp(\text{BA } 1) + \ell \cdot \sharp(\text{BA } 2) + K_0 \cdot \sharp(\text{BA } 3),$$

where K_0 is the cost for testing $e_1 H^T = s + e_2 H^T$ given that $h_\ell(e_1 H^T) = h_\ell(s + e_2 H^T)$ and $\sharp(\text{BA } i)$ is the expected number of execution of the instruction (BA i) before we meet the (SUCCESS) condition.

The first assumption has to do with the way the attacker chooses the sets W_1 and W_2. In the version presented at the beginning of Sect. 2, they use different sets of columns and thus all pairs (e_1, e_2) lead to different words $e = e_1 + e_2$. When W_1 and W_2 increase, there is some waste, that is some words $e = e_1 + e_2$ are obtained several times. A clever choice of W_1 and W_2 may decrease this waste, but this seems exceedingly difficult. The "overlapping" approach



is easy to implement and behaves (almost) as if W_1 and W_2 where random (it is even sometimes slightly better). The second assumption counts only ℓ binary operations to perform the sum of $w/2$ columns of ℓ bits. This can be achieved by a proper scheduling of the loops and by keeping partial sums. This was described and implemented in [6]. We also neglect the cost of control and memory handling instructions. This is certainly optimistic but on modern processors most of those costs can be hidden in practice. The present work is meant to give security levels rather than a cryptanalysis costs. So we want our estimates to be implementation independent as much as possible.

Similar comments apply to the assumptions **(I1)** and **(I2)** of Sect. 3.

B A Sketch of the Proof of Proposition 2

We provide here some clues for the proof of Proposition 2. More details on this proof and on the proofs of the other results of this paper can be found in the extended version [14].

Proof. (of Proposition 2 - Sketch) In one execution of (MAIN LOOP) we examine $\lambda(z)\binom{k+\ell}{p}$ distinct value of $e_1 + e_2$, where $z = |W_1||W_2|/\binom{k+\ell}{p}$ and $\lambda(z) = 1 - \exp(-z)$. The probability for one particular element of $W_{k+\ell,p}$ to lead to a solution is

$$P = \frac{\binom{r-\ell}{w-p}}{\min\left(\binom{n}{w}, 2^r\right)}.$$

Thus the probability for one execution of (MAIN LOOP) to lead to (SUCCESS) is

$$P_p(\ell) = 1 - (1-P)^{\lambda(z)\binom{k+\ell}{p}} \approx 1 - \exp\left(-\frac{\lambda(z)}{N_p(\ell)}\right) \text{ where } N_p(\ell) = \frac{\min\left(\binom{n}{w}, 2^r\right)}{\binom{r-\ell}{w-p}\binom{k+\ell}{p}}$$

When $N_p(\ell)$ is large (much larger than 1), we have $P_p(\ell) \approx \lambda(z)/N_p(\ell)$ and a good estimate for the cost is

$$\frac{N_p(\ell)}{\lambda(z)}\left(\ell|W_1| + \ell|W_2| + K_{w-p}\frac{\lambda(z)\binom{k+\ell}{p}}{2^\ell}\right).$$

Choosing $|W_1|$, $|W_2|$, ℓ and z which minimize this formula leads to the first formula of the statement.

Else we have $N_p(\ell) < 1$ and the expected number of execution of (MAIN LOOP) is not much higher than one (obviously it cannot be less). In that case we are in a situation very similar to a birthday attack in which the list size is $L = \sqrt{1/P} = 2^{r/2}/\sqrt{\binom{r-\ell}{w-p}}$. This gives a cost of $2L\log(K_{w-p}L)$ which has to be minimized in ℓ, leading to the second formula of the statement.