Improving the efficiency of Generalized Birthday Attacks against certain structured cryptosystems

Robert Niebuhr, Pierre-Louis Cayrel, Johannes Buchmann

Improving the efficiency of Generalized Birthday Attacks against certain structured cryptosystems

Robert Niebuhr[1], Pierre-Louis Cayrel[2], and Johannes Buchmann[1,2]

[1] Technische Universität Darmstadt
Fachbereich Informatik, Kryptographie und Computeralgebra,
10 Hochschulstraße, 64289 Darmstadt, Germany
{rniebuhr,buchmann}@cdc.informatik.tu-darmstadt.de
[2] CASED – Center for Advanced Security Research Darmstadt,
32 Mornewegstrasse, 64293 Darmstadt, Germany
pierre-louis.cayrel@cased.de

Abstract. Code-based cryptographic schemes are promising candidates for post-quantum cryptography since they are fast, require only basic arithmetic, and because their security is well understood. Due to their main drawback of large public key sizes, there have been many proposals on how to reduce the key sizes. Many of these use highly structured matrices which can be stored more efficiently. In this paper, we show how a broad class of such structures can be exploited to increase the time and memory efficiency of a Generalized Birthday Attack (GBA), which is one of the best generic attacks against code-based cryptosystems. For example, this improves the best attack against QD-CFS (with $n = 30924$) and FSB_{512} by a factor of 180 and 1984, respectively. In general, for a parity-check matrix of size $r \times n$, the improvement is a factor of r, which is typically in the order of 2^8 to 2^{12}.

Keywords: Generalized Birthday Attack, quasi-cyclic, quasi-dyadic, codes, post quantum, cryptography.

Introduction

In 1994, P. Shor [15] has shown that quantum computers can break most or all "classical" cryptosystems, e.g. those based on RSA or elliptic curves. Therefore, it is crucial to develop cryptosystems that are resistant to quantum computer attacks. Code-based cryptography is a very promising candidate for post-quantum cryptography since the cryptographic schemes use only basic arithmetic operations and do not require special hardware, specifically no cryptographic coprocessor. The first application was the McEliece encryption scheme [11] which was published in 1978. It is as old as RSA and has resisted cryptanalysis to date (except for a parameter adjustment). Other examples of code-based cryptosystems are the (provably secure) FSB hash function, and QD-CFS signature, which achieves a signature size of only 180 bit. One of the most important generic attacks against code-based cryptosystems is the Generalized Birthday attack (GBA) and its modifications. It has been proposed by Wagner in 2002 [16] and was generalized by Minder and Sinclair in 2009 [12]. Many code-based cryptosystems have the drawback of having large public key sizes. Therefore, there

have been many proposals attempting to reduce these key sizes; most recently, quasi-cyclic (QC) alternant [6] and quasi-dyadic (QD) Goppa [13] codes, and the QD-CFS signature scheme [4]. Many of these proposals have in common that they use codes that can be represented by highly structured matrices, which allows to store them more efficiently.

Related work

The birthday paradox refers to the fact that collisions between two (usually random) lists can be found in $\mathcal{O}(2^{n/2})$, much faster than $\mathcal{O}(2^n)$ for checking every possible combination.
Wagner's generalized this algorithm in 2002 by allowing more than two lists. The GBA takes lists $L_1, \ldots, L_t$ and an element s as input and attempts to find exactly one element $x_i \in L_i$ per list such that $\sum_i x_i = s$. By starting with t instead of 2 lists, the generalized algorithm achieves a complexity of $\mathcal{O}(2^{2\sqrt{n}})$. However, it requires a larger number of input elements and therefore more memory than the classic Birthday algorithm.
Minder and Sinclair's *extended k-tree algorithm* can be seen as a further generalization of Wagner's work. It allows to balance the time and memory efficiency of the algorithm. This enables an attacker to optimize the attack performance in the presence of memory constraints. We will demonstrate our improvement based on Minder and Sinclair's version of the GBA.
In 2009, Bernstein et al. published a successful attack [8] against FSB_{48}, a toy version of FSB, that required just under 8 days. This attack used the two techniques mentioned in Remark 3, but did not exploit the quasi-cyclic structure of the matrix.

Our contributions

In this paper, we show how the structure of the parity check matrix can be used to speed up GBA-like algorithms and reduce the memory requirements. Our efficiency improvement covers a broad range of structures, that includes, but is not limited to, the proposals above. Examples are given in Table 1.

Table 1. Examples showing the improvement of time and space complexity (written in $\log_2$) for various cryptosystems.

					GBA		Improved GBA	
System	n	r	t	q	Time	Space	Time	Space
QD-CFS	30924	180	12	2	76.5	76.6	69.0	69.1
QD-CFS	989724	240	12	2	96.9	96.9	89.0	89.0
FSB_{160}	$5 \cdot 2^{18}$	640	$2 \cdot 80$	2	107.6	106.9	98.3	97.6
FSB_{512}	$31 \cdot 2^{16}$	1984	$2 \cdot 248$	2	344.0	343.5	333.0	332.6
NTRU	$2 \cdot 53$	53	43	37	28.0	20.5	22.3	14.8
NTRU	$2 \cdot 491$	491	337	367	158.0	167.0	149.1	158.1

Due to the structure of the corresponding parity check matrix, the original GBA calculates and stores the same elements many times. We demonstrate how to exploit the structure to get the above improvements.

Organization of the paper

In Section 1 we begin with a review on code-based cryptography. The next section outlines a basic GBA algorithm and shows how it can be used to attack code-based cryptosystems. We present our efficiency improvement in Section 3. We conclude in Section 4.

1 Preliminaries

1.1 Coding theory

In this paper, we consider linear error-correcting codes over a finite field $\mathbb{F}_q$, where q is a prime power. A linear code $\mathcal{C}$ is a k-dimensional subspace of an n-dimensional vector space over $\mathbb{F}_q$ and is called an $[n, k]$ code. The elements of a code are called codewords. The (Hamming) weight of a vector is the number of its non-zero entries, and the (Hamming) distance of two vectors is the weight of their difference. Another common notation is the *co-dimension* r where $r = n-k$.

Definition 1 (Generator and Parity Check Matrix). *Let $\mathcal{C}$ be a linear code over $\mathbb{F}_q$. A generator matrix G of $\mathcal{C}$ is a matrix whose rows form a basis of $\mathcal{C}$:*

$$\mathcal{C} = \{xG : x \in \mathbb{F}_q^k\}.$$

Two generator matrices generate equivalent codes *if one is obtained from the other by a linear transformation. Therefore, we can write any generator matrix G in* systematic form *$G = [I_k|R]$, with $R \in \mathbb{F}_q^{k\times r}$ and I_k is the identity matrix of size k, which allows a more compact representation.*
A parity check matrix H of $\mathcal{C}$ is defined by

$$\mathcal{C} = \{x \in \mathbb{F}_q^n : Hx^T = 0\}$$

and generates the dual space of $\mathcal{C}$. If $\mathcal{C}$ is generated by $G = [I_k|R]$, then a parity check matrix for $\mathcal{C}$ is $H = [-R^T|I_r]$ (sometimes H is transformed so that the identity submatrix is on the left hand side).
For a given parity check matrix H and any vector e, we call s the syndrome *of e with $s^T := He^T$.*

For a matrix H, we will denote its rows by H_i and its columns by $H_{.,i}$.

1.2 The syndrome decoding (SD) problem

Problem 1. Given a matrix H and a vector s, both over $\mathbb{F}_q$, and a non-negative integer t; find a vector $e \in \mathbb{F}_q^n$ of weight t such that $He^T = s^T$.

This problem was proved to be NP-complete in 1978 [7], but only for binary codes. In 1994, A. Barg proved that this result holds for codes over all finite fields ([3, in russian] and [2, Theorem 4.1].
Many cryptosystems are based on the the hardness of the SD problem. Among the best-known are the McEliece [11] public key cryptosystem (PKC) and its dual, the Niederreiter [14] PKC, as well as the Courtois-Finiasz-Sendrier (CFS) [9] signature scheme.
There exist two generic attacks to solve the SD problem: Information Set Decoding (ISD) and Generalized Birthday Attacks (GBA). In this paper, we will present an efficiency improvement for the GBA. Our improved algorithm is not restricted to binary codes, it can be used for codes over $\mathbb{F}_q$, where q is a prime power.

2 Generalized Birthday algorithms

Generalized Birthday algorithms are used for some of the most efficient attacks against code-based cryptosystems. They have been proposed by Wagner in 2002 [16], and they are named after the famous birthday paradox which allows to quickly find common entries in lists. In 2009, Minder and Sinclair proposed a further generalization of this algorithm [12] which allows to trade-off time and space efficiency. In this section we will review the basic principles of Generalized Birthday Attacks (GBA).
GBA attempts to solve the SD problem stated above, i.e. for given parity check matrix H, syndrome s and integer t, to find a vector e of weight t such that $He^T = s^T$. In other words, find a set of columns of H whose weighted sum equals the given syndrome. For easier demonstration of the algorithm, let $t = 2^a$ for some integer a. If this is not the case, the algorithm still works with a slight loss of efficiency. Let H be the parity check matrix of a $[n, k]$ code, i.e. H has size $r \times n$.
First, the algorithms sets up t lists $L_1^1, \ldots, L_t^1$, each of which consists of every column of H (in the q-ary case, also all non-zero multiples of these columns). Each list therefore consists of $(q-1)n$ vectors in $\mathbb{F}_q^r$.
In the next step, pairs of lists are merged into new ones: For $i \in [1, t/2]$, the lists L_{2i-1}^1 and L_{2i}^1 are merged to form a new list L_i^2, with $L_i^2 = L_{2i-1}^1 \oplus L_{2i}^1$. This means that the new list is created by considering all pairs of one element from L_{2i-1}^1 and one from L_{2i}^1, and adding their sum to L_i^2. If an entry occurs more than once in a new list, it is removed to minimize the size of the lists. The maximum size of L_i^2 is hence $((q-1)n)^2$.
Following this, the lists L_i^2 are merged to form L_i^3, and this merging step is repeated until exactly one list remains. This list contains the solution to the problem stated above. Often, a slight modification is introduced: The vector s is subtracted from each entry in the last list L_t^1, and the algorithm attempts to find a combination of entries that sum up to zero. In fact, the final list contains *all* possible solutions. The number of solutions can be quite large for certain applications, e.g. for attacks on signature schemes. This fact can be used to

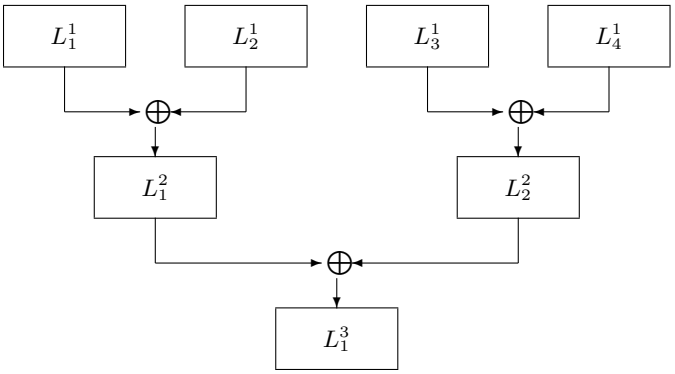

Fig. 1. Illustration of the GBA for $a = 2$.

further improve the attack, as shown by Wagner (and in a more general way, by Minder and Sinclair):
After each merge step, the lists are being reduced in size. This is done by forcing a certain, increasing number of bits to zero. Each forced bit reduces the expected list size by a factor of q, and the number of bits that are forced to zero in each step can be optimized with respect to the parameters $[n, r]$ and t. The reduction of the list size is an improvement in space – less data needs to be stored – as well as in time, since there are less elements that need to be manipulated.
The time complexity of the GBA is in $\mathcal{O}(2^{a+u})$, where 2^u is the maximum expected list length and $t = 2^a$. See [12] for a detailed proof.
This fact is the reason why GBA are usually most efficient when there are a large number of solutions. For example, an unpublished attack by D. Bleichenbacher against the CFS signature scheme [9] was very efficient and required an increase in the CFS parameters to remain secure (the attack is described in [10]).

3 Efficiency improvement

Since the size of the public key is one of the drawbacks of many code-based cryptosystems, there have been many proposals on how to reduce the key size, most recently quasi-cyclic (QC) codes [6], quasi-dyadic (QD) codes [13], QD-CFS signatures [4], and the Fast Syndrome-Based (FSB) hash function [1]. In many cases, the key size reduction is achieved by restricting the cryptosystem to a class of highly structured codes which can be stored efficiently. In this chapter we will show how this structure can be exploited to improve Generalized Birthday Attacks.

3.1 Preliminaries

Our analysis is applicable in all cases where the underlying code has a parity check matrix of a certain structure:

The structure of the parity check matrix H is such that each row is a permutation of the first row. This is true, for example, for the cryptosystems based on QC and QD codes in [6,13,4].
This means that we have permutations Θ_i, $i \in [2, r]$, such that the i-th row H_i of H can be computed by

$$H_i = \Theta_i(H_{i-1}) \quad \text{for } i \in [2, r].$$

We abuse notation and write, for integers a and b, $\Theta_i(a) = b$, to say that Θ_i permutates the a-th entry to position b.

Definition 2. *To efficiently address the list entries we note that for every entry x in the lists there are unique $c_1 < c_2 < \ldots < c_k$ and $h_1, h_2, \ldots, h_k$, such that x was created by the weighted sum of columns of H:*

$$x = \sum_{i=0}^{k} c_i H_{\cdot,h_k}.$$

We define $\text{index}(x) := (c_1, \ldots, c_k; h_1, \ldots, h_k)$ *and write*

$$\Theta_i((c_1, \ldots, c_k; h_1, \ldots, h_k)) = (c_1, \ldots, c_k; \Theta_i(h_1), \ldots, \Theta_i(h_k)). \tag{1}$$

Since an index uniquely defines an element, we can associate the two and write for short
$y := \text{index}(x)$. *For any vector x, let us denote its i-th entry by $x[i]$. Then*

$$x[i] = \Theta_i^{-1}(\text{index}(x))[i-1]. \tag{2}$$

3.2 Improved algorithm

We will describe our improved algorithm as a modification of the one by Minder and Sinclair [12]. We start with $t = 2^a$ lists L_i^1. In our case, each lists contains n entries, the columns of H. However, due to the structure of H, we only need to store the first bit of each column. We let l_i denote the number of bits that is forced to zero in the i-th merging step.
For the moment we assume $s = 0$, i.e. we want to find t columns of H whose weighted sum is the zero vector. We will later relax this assumption.
In the first merging step, we calculate the first bit of the entries in L_i^2 by adding pairs of entries of L_{2i-1}^1 and L_{2i}^1. If $l_1 = 0$ we continue with the second merging step; again, only calculating the first bit of L_i^3. If $l_1 > 0$ we need to calculate the next $l_1 - 1$ bits of each list entry to decide which ones to eliminate:
We start by eliminating all vectors whose first bit is non-zero. If $l_1 > 1$ we continue with the second row. To calculate the second bit of each remaining vector, we apply equation (2). If the list entry $\Theta_i^{-1}(\text{index}(x))$ cannot be found, we know that we have already eliminated it, i.e. $\Theta_i^{-1}(\text{index}(x))[1] \neq 0$. Since we force the second row to zero as well, we eliminate x.
After each row that has been calculated, we do not have to store the previous

row anymore, since it is completely forced to zero. Hence, in each merging step j, we only have to store *one* row of the current lists L_i^j, plus one row of the lists currently being computed.
After completing the computation of the first l_1 rows, we continue with the next merging step, this time calculating the l_1-th bit of the new vectors.
The following merge and elimination steps work as described above, where the j-th step forces bits $[\sum_i^{j-1} l_i + 1, \sum_i^j l_i]$ to zero and continues of the next step in row $\sum_i^j l_i$. The pseudo code of our algorithm is shown in Algorithm 2. The operand $\oplus_i$ refers to the operation of creating the new list from sums of all pairs of the two parent lists, only computing and storing the i-th bit, as shown in Algorithm 1 (here, $||$ refers to the concatenation operation):

Algorithm 1 Pseudo code for $C := A \oplus_f B$ over $\mathbb{F}_q$

```
1: for all a ∈ A, b ∈ B do
2:     C ← C||(a[f] + b[f]  mod q)
3: end for
```

Remark 1. The GBA algorithm described above assumes $s = 0$. If $s \neq 0$, we can modify it similarly as Wagner proposed in [16]: The rightmost list L_t^1 does not get columns $(H_{\cdot,i})$ of H, but $(H_{\cdot,i} - s)$. This allows us to search for list elements that sum to zero, and the corresponding columns of H will sum to s.
For the merging step of each rightmost list, we have to modify equation (2), since different rows use different bits of s as an offset:

$$\text{For given } s : x[i] = \Theta_i^{-1}(\text{index}(x))[i-1] + s[i-1] - s[i]. \tag{3}$$

Remark 2. In practice, this modification can be implemented efficiently by using look-up tables for the Θ_i, and by using a suitable data structure for the lists L_i^j.

3.3 Analysis of time and memory efficiency

Our modification to the GBA, using the structure of H, reduces the number of computations and memory access, as well as the total memory required. Table 2 shows the effect of our improvement. The number of computations and memory access refers to a single merge step of two lists. We will use the resulting list size ls as the complexity of the operation (as described in [12], for example). Let r be the length of the list entries in bits, and q the field size. The last table item shows the total memory required to store the lists. Let $\text{ls}(x)$ denote the size of list x.
The second term of the required memory for the improved GBA refers the memory that is needed to temporarily store the new row of a list while it being computed, until the previous row can be deleted from memory.

Algorithm 2 Pseudo code of improved GBA algorithm

INPUT: A $r \times n$ parity check matrix H, integer $t = 2^a$, integers $l_1, \ldots, l_a$ with $\sum_i l_i \leq r$, vector s, a prime power q.

OUTPUT: Index(es) index(x) such that the corresponding weighted sum equals s, or "No solution found".

SETUP: Add the first entry of all n columns of H to each list $L_1^1, \ldots, L_t^1$. Subtract s from all elements of L_t^1. Let $f_1 = f_2 = 0$.

for $step \leftarrow 1$ **to** a **do**
  $f_1 \leftarrow f_2$
  $f_2 \leftarrow f_2 + l_{step}$
  for $j \leftarrow 1$ **to** 2^{a-step} **do**
    $L_j^{step+1} = L_{2j-1}^{step} \oplus_{f_1} L_{2j}^{step}$
    for $row \leftarrow f_1 + 1$ **to** f_2 **do**
      ▷ Compute row-th bit of all vectors x in L_j^{step+1} using equations (3)
        (for $j = 2^{a-step}$ and $s \neq 0$) and (2) (for all other lists)
      ▷ If not possible, delete current vector x
      ▷ After each row has been computed, delete the previous row
    end for
  end for
end for
if $|L_1^a| = 0$ **then**
  Return "No solution found"
end if
return index(L_1^a)

Table 2. Effect of the efficiency improvement.

	GBA	Improved GBA
Computations	$ls \cdot r$	ls
Memory access r/w	$ls \cdot r$	ls
Memory for a list	$\sum_{i,j} \mathrm{ls}(L_i^j) \cdot r \cdot \log_2(q)$	$\sum_{i,j} \mathrm{ls}(L_i^j) \cdot \log_2(q)$ $+ \max_{i,j} \lvert\mathrm{ls}(L_i^j)\rvert \cdot \log_2(q)$

Proposition 1. *Our modified GBA algorithm decreases the number of computations and memory access by a factor of r; the total memory required is reduced by a factor close to r, since the maximum list size is small compared to the sum of all list sizes.*

Remark 3. There are (at least) two further possible optimization techniques:

1. Depending on the setup of the initial lists L_i^1, some of the following lists L_i^j can be identical and do not need to be computed or stored twice (e.g. if $L_1^1 = L_3^1$ and $L_2^1 = L_4^1$, then $L_1^2 = L_2^2$)
2. Instead of computing all lists on one level of the tree before continuing to the next level, we can compute the lists in a "depth-first" way:
   - Set up L_1^1 and L_2^1
   - Compute L_1^2 and delete L_1^1 and L_2^1
   - Set up L_3^1 and L_4^1
   - Compute L_2^2 and delete L_3^1 and L_4^1
   - Compute L_1^3 and delete L_1^2 and L_2^2
   - ...

   This allows to reduce the required memory since only approx. one list per level of the tree needs to be stored. See [8], for example, where this technique was used in an attack on the FSB hash function.

4 Conclusion and outlook

In this paper we have shown how the time and memory efficiency of Generalized Birthday Attacks can be improved by a factor of r when applied to a structured matrix of size $r \times n$. Our improvement can be applied to a wide range of possible structures including quasi-cyclic and quasi-dyadic matrices. Even if only part of the matrix is structured, as for the truncated quasi-cyclic matrices in the FSB hash function, we can apply the improvement, but the effect is smaller.
The efficiency improvement is quite large, often in the range of 2^9 to 2^{12}. While this does not completely break the attacked cryptosystem, it has to be taken into account when proposing secure parameters.
As further work we propose to analyze if and how other attacks, for example information set decoding, can exploit such structures. Furthermore, it would be interesting to transfer these results to other areas of cryptography, for example lattices-based schemes.
Also, it would be interesting to analyze if the structure of the matrix can be exploited in case of a "Grover-improved GBA". In 1996, Bennett et al. [5] showed that a brute-force search on a quantum computer cannot be faster than $\mathcal{O}(2^{n/2})$ operations compared to $\mathcal{O}(2^n)$ in the classical case. This can be offset by a twicefold increase in the key size on large enough quantum computer exist. If structures as analyzed in our paper can also be exploited by a Grover-GBA, it would require an additional key size increase.

References

1. D. Augot, M. Finiasz, and N. Sendrier. A family of fast syndrome based cryptographic hash functions. In *Mycrypt*, volume 3715 of *LNCS*, pages 64–83. Springer, 2005.
2. A. Barg. Complexity issues in coding theory. *Electronic Colloquium on Computational Complexity (ECCC)*, 4(46), 1997.
3. S. Barg. Some new NP-complete coding problems. *Probl. Peredachi Inf.*, 30:23–28, 1994. (in Russian).
4. P. S. L. M. Barreto, P.-L. Cayrel, R. Misoczki, and R. Niebuhr. Quasi-dyadic CFS signatures. In *Inscrypt 2010*, LNCS. Springer, Oct 2010.
5. C. H. Bennett, E. Bernstein, G. Brassard, and U. V. Vazirani. Strengths and weaknesses of quantum computing. *SIAM J. Comput.*, 26(5):1510–1523, 1997.
6. T. P. Berger, P.-L. Cayrel, P. Gaborit, and A. Otmani. Reducing key length of the McEliece cryptosystem. In *AFRICACRYPT*, volume 5580 of *LNCS*, pages 77–97. Springer, 2008.
7. E. Berlekamp, R. McEliece, and H. van Tilborg. On the inherent intractability of certain coding problems. *IEEE Trans. Inform. Theory*, 24(3):384–386, 1978.
8. D. J. Bernstein, T. Lange, R. Niederhagen, C. Peters, and P. Schwabe. Fsbday. In B. K. Roy and N. Sendrier, editors, *INDOCRYPT*, volume 5922 of *LNCS*, pages 18–38. Springer, 2009.
9. 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.
10. M. Finiasz and N. Sendrier. Security bounds for the design of code-based cryptosystems. In *ASIACRYPT*, pages 88–105, 2009.
11. R.J. McEliece. A public-key cryptosystem based on algebraic coding theory. *DNS Progress Report*, pages 114–116, 1978.
12. L. Minder and A. Sinclair. The extended k-tree algorithm. In C. Mathieu, editor, *SODA*, pages 586–595. SIAM, 2009.
13. R. Misoczki and P. S. L. M. Barreto. Compact McEliece keys from Goppa codes. In *SAC 2009*, volume 5867 of *LNCS*. Springer, 2009.
14. H. Niederreiter. Knapsack-type cryptosystems and algebraic coding theory. *Problems of Control and Information Theory*, 15(2):159–166, 1986.
15. P. W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. pages 124–134. IEEE Press, 1994.
16. D. Wagner. A generalized birthday problem. In M. Yung, editor, *CRYPTO*, volume 2442 of *LNCS*, pages 288–303. Springer, 2002.