Improved Generalized Birthday Attack

Paul Kirchner

July 11, 2011

Abstract

Let r, B and w be positive integers. Let C be a linear code of length Bw and subspace of $\mathbb{F}_2^r$. The k-regular-decoding problem is to find a nonzero codeword consisting of w length-B blocks with Hamming weight k. This problem was mainly studied after 2002. Not being able to solve this problem is critical for cryptography as it gives a fast attack against FSB, SWIFFT and learning parity with noise. In this paper, the classical methods are used in the same algorithm and improved.

Keywords: Generalized Birthday Attack, Linearization, Information-Set Decoding, Wagner, Low memory requirement, SWIFFT, FSB, LPN

1 Introduction

The linear code is considered to be random. Logarithm are base 2 logarithm. We can first remark that the problem can be reduced to 1-regular-decoding problem with $B' = \binom{B}{k}$. The first 1-regular-decoding algorithm better than applying birthday attack was found in 1991[7] for $w = 4$ and independently generalized in 2002[16] for B as large as we want. Finally, Wagner's algorithm was generalized by Minder et al in [13]. This class of attack is named Generalized Birthday Attack and is fast when the largest a such that $B^{\lfloor w/a \rfloor} > 2^{r/(1+log a)}$ is large. It is based on merging lists and removing bits.

The second class of attack is named Linearization. It was firstly introduced in addition of GBA by Wagner[16] and allowed to reduce with a gaussian elimination the problem to $r' = r - l$ and $w' = w - l$. Saarinen[16] used linearization to break some FSB parameters. With $r \leq 2w$ and $k = 1$, the algorithm uses $\Theta((4/3)^(r-w))$ gaussian eliminations. With $r \leq 4w$ and searching for sums of two 1-regular codewords, the algorithm uses $\Theta((4/3)^{r-2w})$. His algorithm was generalized in[1] for every r. This attack is efficient when r/w is low.

The third class of attack is a generalization of information-set decoding. This attack first appeared in [4] and was in fact linearization using information-set decoding words. [5] applied classical information-set decoding methods in order to improve [4].

The new attack combines linearization and generalized birthday attacks and thus, is faster than all these attacks. Furthermore, both algorithms are slightly improved. This leads to practical attacks against the cryptosystems. The algorithm was implemented and allowed to find a collision in FSB-48 in far less ressources than previously in [3].

We will describe the algorithm by different reductions of the parameters of the problem. The goal is to have l differents k-regular non-zero codewords in the subspace $\mathbb{F}_2^r$ with w lists. This approach directly leads to dynamic programming in order to have the fastest algorithm.

2 Generalized birthday attack

2.1 Time efficient solution

Let m be an integer. If $l \leq 2^m$, we reduce to solving two problems with $w' = \lfloor w/2 \rfloor$, $l' = 2^m$, $r' = r - 2m + \lceil \log(l) \rceil$. Then, sort the numbers according to the last m bits. For each pair of numbers, if their $2m - l$ last bits are identical, output the sum. The time and memory complexity are $\Theta(r2^m + rl)$ using counting sort.

Or, we reduce to solving n problems with $w' = \lfloor w/n \rfloor$, $l' = \sqrt[n]{l}$, $r' = r$. For each n tuple of numbers, output the sum. Time complexity is $\Theta(r(l+n))$. Memory complexity is $\Theta(rn\sqrt[n]{l})$.

Wagner proposed to have $m = r/(1+\lfloor \log(w) \rfloor)$. Thus, with $B = m$, $k = 1$ and $l = 1$, the algorithm has a time complexity of $\Theta(rw2^{r/(1+\lfloor \log(w) \rfloor)})$. Using postfix evaluation, we can only keep in memory one list for each level and have a memory complexity of $\Theta(r \log(w) 2^{r/(1+\lfloor \log(w) \rfloor)})$. We propose on level i (level 0 being top of the tree) to have $m'(i) = m + \log w/2 - i + O(1)$. The algorithm works, because $m'(i)$ bits are cleared on level $i > 0$, $2m(0)$ bits are cleared on level 0, and $\sum_{i=1}^{\lfloor \log(w) \rfloor} m'(i) = (\lfloor \log w \rfloor - 1)m + O(\log(w))$ Thus, time complexity is $\Theta(r \log(w)\sqrt{w}2^{r/(1+\lfloor \log(w) \rfloor)})$ and memory complexity is $\Theta(r\sqrt{w}2^{r/(1+\lfloor \log(w) \rfloor)})$. So, this algorithm is $\Theta(\sqrt{w}\log(w))$ faster but uses $\Theta(\sqrt{w}\log(w))$ more memory. Using $w = 2^{\sqrt{2r}}$, we have a time complexity of $\Theta(r^{\frac{3}{2}}2^{\sqrt{2r}})$.

For $B \leq 2^m$, this algorithm is the extended k-tree algorithm of Minder and Sinclair [13].

2.2 Memory efficient solution

As previously remarked in many papers, generalized birthday attack takes many memory compared to the time of the attack. This is a problem when this attack is implemented as we cannot afford such a large memory in practice [3].

2.2.1 Clamping through precomputation

This method was discovered in [3]. When $w = 1$ and $B \geq l2^r$, we can take all numbers with their last r bits equal to 0. On average, after having tested $l2^r$ numbers, we have generated l numbers as the numbers are random. Time complexity is $\Theta(l2^r)$ and memory complexity is $\Theta(r)$.

2.2.2 Repeating the attack

This method was discovered in [2] and used in [3]. It consists in changing the lists and repeating the attack. Bernstein remarked that instead to choose only numbers with their last r bits equal to 0, we can choose w constants with a sum equal to 0, and choose for each list numbers with their last bits equal to the constant. So, the problem is reduced to $r' = r - a$ but must be repeated $\Theta(2^a)$ times for an integer a. Thus, when memory is halved, the attack need to be repeated $\Theta(2^{1+\lfloor \log(w) \rfloor})$ times more.

2.2.3 Asymmetric tree

This method was discovered in [9]. They remarked that, when we merge two lists, having one list in memory is enough. Indeed, for each number in the other one, we can search in the first list what numbers start with the same $m + u$ bits in $O(1)$ by using a table. Thus, we reduce the problem to one problem with $r' = r - 2m - u + \log l$, $l' = 2^m$, $w' = \lfloor w/2 \rfloor$ and to one problem with $r'' = r'$, $l'' = 2^{m+u}$, $w'' = w'$. Time complexity is $\Theta(r2^{m+u})$ and memory complexity is $\Theta(r2^{m+u})$.

When we apply this technique on a times, halving the memory required multiply the time complexity by $\Theta(2^a)$. Also, the size of the list on the last level is multiplied by $\Theta(2^a)$ which is a problem when B is not as large as we want. This drawback can be diminished by not splitting the list in 2 parts of nearly the same size. However, as the following algorithms are faster, we will not thoroughly analyzed this algorithm.

2.2.4 Four lists

Let u and m' be integers such that $u \leq 2m' - m \leq m$ and $2m' \geq m$. Reduce the problem to 4 problems with $r' = r - 2m - m' - u + \log(l)$, $l' = 2^{m'}$ and $w' = \lfloor w/4 \rfloor$. We now have 4 lists of size $2^{m'}$. Generate 2^u m'-bits different integers. For each of these integers, add it to each number of the lists 1 and 3. Then, merge lists 1 and 2 into list 5 and clamp $2m' - m$ bits, merge list 3 and 4 into list 6, clamp $2m' - m$ bits. Finally, merge lists 5 and 6 and output all the numbers with $2m + m' + u - \log(l)$ bits equal to zero. Time complexity is $\Theta(r2^{m+u})$, memory complexity is $\Theta(r2^m)$. So, when memory is halved, we need to increase u by two and time is multiplied by two. This is also true when only the four lists algorithm is applied. As long as $u \leq m'$, the product time memory is constant. Interestingly, the devices that maximizes this product are hard disks and graphic cards. This attack can be parallelized : $\Theta(r(2^{m'} + l))$ bits need to be exchanged and the $\Theta(r2^{m+u})$ operations can be executed on 2^p devices with $\Theta(r2^m)$ memory in time $\Theta(r2^{m+u-p})$ with $p \leq u$. There are m' free clamping bits.

2.2.5 Eight lists

Let u and v be integers such that $u \leq 2m' - m$ and $v \leq 2m' + u$. Reduce the problem to 8 problems with $r' = r - 3m - m' - u - v + \log(l)$, $l' = 2^{m'}$ and $w' = \lfloor w/8 \rfloor$. Generate 2^v different $2m' + u$-bits numbers. For each number, add it to lists 1 and 5. Then, apply the four lists algorithm using u on lists 1 to 4 and 5 to 8. Finally, merge the two lists and keep the numbers with $3m + m' + u + v - \log(l)$ zeros. Time complexity is $\Theta(r2^{m+u+v})$ and memory complexity is $\Theta(r2^m)$. There are $3m' + u$ free clamping bits.

2.2.6 Sixteen lists

Let u, v and w be integers such that $u \leq 2m' - m$, $v \leq 6m' + 3u$ and $u \leq w \leq m$. Reduce the problem to 16 problems with $r' = r - 4m - m' - u - v - w + \log(l)$, $l' = 2^{m'}$ and $w' = \lfloor w/16 \rfloor$. Generate 2^v different 3-uple of $2m' + u$-bits numbers. For each 3-uple, add the first number to list 1, the second to list 5, the third to list 9 and the sum of the three to list 14. Then, apply the four lists algorithm using u on the lists grouped by four. Finally, apply the four lists algorithms using w and keep the numbers with $4m + m' + u + v + w - \log(l)$ zeros. Time complexity is $\Theta(r2^{m+w+v})$ and memory complexity is $\Theta(r2^m)$. There are $6m' + 3u + m + w$ free clamping bits.

Generalization of this idea is left to the reader. In general, we can say that for some a, we can reduce the problem to $r' = r - am - m' - u + \log(l)$, $l' = 2^{m'}$ and $w' = \lfloor w/2^a \rfloor$ with $u \leq (2^{a-2} - a + 2)m + (2^{a-2} - 1)(2m - m')$. Time complexity is $O(ra2^{m+u+a/2})$ and memory complexity $\Theta(r2^{m+a/2})$.

2.2.7 Parallel collision search

[3] introduced the idea of using Pollard iteration when the amount of memory is low and k is even. We generalize the idea and give the limit to it. Let 2^{a+1} be the number of lists, u inferior or equal to the number of free clamping bits, $v + \log(l) \leq u$, b the number of bits cleared by the 2^a lists algorithm, $w \leq b$. Reduce the problem to 2^a problems with $r' = r - b - u - v - w$, $l' = 2^{m'}$ and $w' = \lfloor w/2^a \rfloor$. Generate 2^w different b-bits numbers. For each of these numbers, add it to lists 1 and 2^a. We define $f(x)$ with x an integer such that $x \leq 2^u$. x corresponds to the different clamping constants at each merge of the tree. Merge the 2^a first lists, there should be $\Theta(1)$ number starting with b zeros. Then, $f(x)$ is the u next bits of the smallest of these. $g(x)$ is the same but with the 2^a lasts lists. Thus, each x such that $f(x) = g(x)$ gives one collision on $(a-1)m + m' + u$ bits. With the Van Oorschot-Wiener algorithm [14], we can find $l2^v$ collisions in $\Theta(2^{u/2+v/2}\sqrt{l})$ calls to the 2^a lists algorithms. Finally, we need a total of $\Theta(2^{u/2+v/2+w}\sqrt{(l)})$ calls, each of these taking time $O(r2^{b-(a-2)m-2m'})$.

3 Linearization

Let λ_i, σ_i and n be integers such that $0 \leq i < n$ and $k = \sum_{i=0}^{n} \sigma_i$. Let $\lambda = \sum_{i=0}^{n} \lambda_i$. Then, the algorithm generates λ_i vectors from one list. We concatenate these vectors and the vectors from the $w' = w-1$ lists and change the basis such that the first λ rows and columns are the identity matrix and the $r - \lambda$ last rows are zero. Let $f(x, s)$ be the number of codewords of x bits with parameter s such that we can recover from the codeword in the new matrix a codeword in the old matrix. Then, we reduce the problem to $r' = r - \lambda$, $l' = l\frac{2^\lambda}{\Pi_{i=0}^{n} f(\lambda_i, \sigma_i)}$ and $w' = w - 1$. So, the larger the product of $f(x, s)$ is, the faster is the algorithm. The gaussian elimination can be done in polynomial time and will be neglected. Also, the memory required is $\Theta(r)$ and time required is $\Theta(rnkl')$.

3.1 Saarinen method

Saarinen proposed[15] to take $n = k$ and $\sigma_i = 1$. We partitioned the B vectors in n blocks, the ith one being of size $\lambda_i + 1$. The ith vector generated of a block is the sum of the first vector and the $i + 1$th vector of the block. Also, we add the sum of the first vector of each block to all the vectors of the last list.

So, for each block, if the Hamming weight is 1 and the one is present in the jth column, we can recover the codeword in the old matrix by adding the jth vector of the block. If the Hamming weight is 0 in the ith block, we can recover the codeword by adding the ith vector. Thus, $f(x, s) = \binom{x}{1} + \binom{x}{0} = x + 1$.

3.2 Augot method

Augot proposed in [4] to take $n = 1$ and $\sigma_0 = k$. The ith vector generated is exactly the ith vector of the list.

To recover, if the Hamming weight of the codeword is k, then for each position where there is a 1 in the codeword, we add the corresponding vector. Thus, $f(x, s) = \binom{x}{k}$.

3.3 Improved method

We partitioned the B vectors in n blocks, the ith block being of size $\lambda_i + 1$. The ith vector generated of a block is the sum of the first vector and the $i+1$th vector of the block. If σ_i is odd, we add the first vector of the block to all vector of the last list.

To recover, for each position where there is a 1 in the codeword, we add the corresponding vector. If the Hamming weight is σ_i, then the method is successful. If the Hamming weight is $\sigma_i - 1$ in the ith block, we can recover the codeword by adding the first vector of the ith block. Thus, $f(x,s) = \binom{x}{s} + \binom{x}{s-1}$. For $k = 2$, we use $n = 1$ and $\sigma_0 = 2$. So $f(x,s) = \binom{x}{2} + \binom{x}{1} = x(x+1)/2$. Also, we can check that this method is better than the two others.

4 Applications

4.1 Fast Syndrome-Based Hash

FSB was first introduced in [4]. The compression function used is a multiplication of a matrix which we will consider random by a 1-regular vector.

Finding a collision is equivalent to finding a codeword which is the sum of two differents 1-regular vectors using the matrix given as the parity check matrix of the code. A sum of two 1-regular vectors means that each block has either a Hamming weight of 2 or a Hamming weight of 0. So, we can use the previous algorithm for searching a 2-regular codeword. As a block of Hamming weight 0 is allowed, $f(x,s)$ may be higher. For the improved method, $f(x,s) = x(x+1)/2 + 1$. If we use only linearization and $2w \leq r \leq 3w$, the algorithm selects $r - 2w$ lists with $\lambda = 3$ and $3w - r$ lists with $\lambda = 2$. The number of iteration is thus $(8/7)^{r-2w}$. When $r = 4w$, the algorithm selects w lists with $\lambda = 4$ and needs $(16/11)^w$ iterations or approximately $2^{0.14r}$ instead of $2^{0.21r}$ for the Saarinen method.

Finding first pre-image can be done by adding the given hash to all vectors from one list and then searching for 1-regular codeword. Finding second pre-image can be done by searching a first pre-image. In order not to find the same message, we remove only one column. If we use linearization, then generally $\lambda < B$, so we can remove one column is this list and finding a second pre-image is exactly as difficult as finding a first pre-image.

We have managed to find a collision in the compression function of FSB 48 ($r = 192$, $w = 24$, $B = 2^{14}$) using $m = 25$ in 6 hours. The program used 3 GB of RAM, about the same quantity of memory on hard disk and one core at 2 GHz. [3] used only Wagner's algorithm and their attack took 8 days, on 32 cores at 2.4 GHz, 5376 GB of memory and $m = 37$.

RFSB[1] ($r = 509$, $w = 112$, $B = 2^8$) can be attacked with a cost of 2^{79} using $m = 45$.

FSB-384[8] ($r = 1472$, $w = 184$, $B = 2^{13}$) can be attacked with a bit complexity of about 2^{369} using $m = 60$ instead of 2^{622} given in [3].

4.2 SWIFFT

SWIFFT was introduced in [12]. The compression function used is a multiplication of a matrix which we will consider as random on $\mathbb{F}_q^r$ by a vector in $0, 1^n$ with $q = 257$, $r = 64$ and $n = 1024$. So, to find a collision, it is sufficient to find two vectors in $0, 1^r$ with an identical product with the matrix. Linearization can be used : if we select k columns, we have $l' = l(q/3)^k$, $r' = r - k$ and $B' = B - k$.

We choose $m = 98$, $k = 14$. We built 16 lists, each using 63 vectors. By merging using Wagner's algorithm and Minder's trick[13], we can remove a non-integer number of coordinates. We merge until we have one list of size 2^{92}which contains the messages which we will take. Finally, duplicate the list, merge it with itself and for the vectors which have their last 14 coordinates in $-1, 0, 1$, we can recover a collision. Thus, we can find collisions with 2^{109} bit operations which is faster than the 2^{120} bit operations of the algorithm given in [12].

4.3 Parity learning with noise

Parity learning with noise has numerous applications in cryptography. The private key is a random vector s in $\mathbb{F}_2^r$. We are given access to an oracle which gives $v, s \cdot v$ with probability $\epsilon < \frac{1}{2}$ with a random v. Let $d = 1 - 2\epsilon$.

4.3.1 Leviel algorithm

This algorithm was discovered in [11] and was an improvement over the BKW algorithm [6].

Let a and m be integers such that $r \leq am$. We build a list of size $\Theta(a2^m + md^{-2^a})$ by using the oracle $\Theta(a2^m + md^{-2^a})$ times. We merge $a-1$ times the list with itself : we grouped the values with the same m bits and for each group, we take one value, add it to the others and remove it. We now have reduced the problem to $r' = r - (a-1)m$, $d' = d^{2^a/2}$ and we can access the oracle $\Theta(m/d'^2)$ times. Use a fast Walsh transform to recover the r' last bits with high probability. Then, we can reduce the problem to $r'' = r - r'$ which is much easier.

4.3.2 Improved algorithm

Ask $\Theta(r)$ vectors, and find r linearly independent vectors. Let M_i be the ith vector, and M be the concatenation of these vectors in an inversible matrix. Let $c, e \in \mathbb{F}_2^r$ such that $Ms = c + e$ and e_i unknown and equal to 1 with probability ϵ and c_i the number returned by the oracle.

Now, we can generate an oracle which has the same properties that the former, but with $s' = e$. An algorithm which can solve the new problem can solve the original problem with a polynomial overhead.

Proof : Ask the former oracle a vector v and $C+E = s \cdot v$ with E unknown and equal to 1 with probability ϵ. Return $v' = M^{-1}v$ and $C + c \cdot M^{-1}v$. We have $Mv' = (MM^{-1})v = v$ and $\sum_{i=0}^{r} v'_i M_i = Mv'$ by definition. So,

$$s \cdot v = s \cdot \sum_{i=0}^{r} v'_i M_i = \sum_{i=0}^{r} v'_i (s \cdot M_i) = \sum_{i=0}^{r} v'_i (c_i + e_i) = v' \cdot (c + e) = v' \cdot c + v' \cdot e$$

And finally, $v' \cdot e = s \cdot v + v' \cdot c$ and $v' \cdot e + E = C + v' \cdot c$. Now, find $s' = e$ using some algorithm. $Ms = c + e$ so $s = M^{-1}(c + e)$.

If we use a naive algorithm, asking the oracle uses $\Theta(r^2)$ bit operations. However, if we want q questions, the problem can be solved in $\Theta(rq)$ bit operations, plus the operations needed to multiply a $r \times r$ matrix by a $r \times q$ matrix. Also, we need to compute M^{-1} which can be done in $O(r^3)$ bit operations. Finally, as the reduction to this new problem is not very expensive in bit operations, the secret s can be choosen with s_i equal to 1 with probability ϵ instead of $\frac{1}{2}$.

Let $m' = r - (a-1)m$. We use the classical BKW algorithm to generate L independent numbers of m' bit with $d' = d^{2^a/2}$ and their believed dot product with s'. We define $S(n, m, p) = \sum i = 0^m \binom{n}{i} p^i (1-p)^{n-i}$. The algorithm consists in checking for every m'-bit number with a Hamming weight less than W $l = \lfloor \log(L) \rfloor$. We know that the secret is of this form with probability $S(m', W, \epsilon)$ and we will thus repeat the attack $\Theta(1/S(m', W, \epsilon))$ by changing the vectors used in the change of oracle. We can remark that the algorithm do not use any property of the last r' bits of the secret. Therefore, we can ask in the first oracle only m' vectors, and add r' vectors of the canonical basis. We can set the believed dot product for them as equal to 0, which is true with probability $\frac{1}{2}$. Thus, if we need q queries, we must multiply a $m' \times r$ matrix by a $r \times q$ matrix. This can be done in $O(m'rq/\log(q))$ bit operations.

Let N be the number of considered possible secrets. We have $N = 2^{m'} S(m', W, \frac{1}{2})$.

Let $A_1 = \begin{pmatrix} 1 \end{pmatrix}$ and $A_n = \begin{pmatrix} A_n & A_n \\ A_n & -A_n \end{pmatrix}$. Then, the Walsh transform of the vector v is given by $A_{m'}v$. The fast Walsh transform consists in computing the fast Walsh transform of the two halves of v, and then use the structure of $A_{m'}$ to recover $A_{m'}v$.

Our problem is to compute the Walsh transform of a sparse vector on a sparse subset of points. We can also use the fast Walsh transform. In order to recover the value of the Walsh transform, we need to recursively compute the Walsh transforms on the same positions with the most significant bit removed. We can remark that, if the most significant bit of a position is one, then the same position with an opposite most significant bit is also in the set. Thus, on level $i < l$, the algorithm takes time $\Theta(2^{m'-i} S(m'-i, W, \frac{1}{2}))$ to conquer. When we want to compute the Walsh transform of a vector with $\Theta(1)$ non-zero values, we can stop the recursion. This occurs on level l as the non-zero values are distributed uniformly. For each possible secret, we need to know with probability $\Theta(S(m', W, \epsilon)/N)$ if it is the secret. Using Chernoff bound, we can bound L to $\Theta(r/d^{2^a})$.

So, the total number of operations is

$$\Theta(\sum_{i=0}^{l} 2^i 2^{m'-i} S(m'-i, W, \frac{1}{2})) = \Theta(2^{m'} \sum_{i=0}^{l} S(m'-i, W, \frac{1}{2}))$$

The BKW algorithm needs $\Theta(a(L+a2^m))$ operations and $\Theta(L+a2^m)$ queries.

As an application, [11] proposed using $r = 768$ and $\epsilon = 0.05$ for 80-bit security. Their algorithm uses 2^{90} bytes of memory. Using $a = 3$, $m = 8$, $m' = 752$, $W = 1$ and $l = 8$, we can solve the problem in 2^{69} operations. As our algorithm consists in about 2^{50} independent iterations, it can easily be massively parallelized and the memory requirement is much lower : about 2^{16} bytes. Finally, the algorithm needs 2^{60} queries.

When ϵ is too high, we cannot choose m' much higher than m and the benefit of this algorithm is destroyed by the cost of the matrix multiplication.

5 Conclusion

By combining two algorithms efficients on different parameters, and by improving each one, we were able to create a new algorithm much faster that the previous one. We have proven that this increased efficiency can be used in practice for the cryptanalysis of the hash function FSB and SWIFFT, and the LPN authentication method.

In many problems, lattice reduction and generalized birthday attack are two differents approaches. Both algorithms can be used for solving integer subset-sums, shortest vector problem or closest vector problem which are criticals to many cryptosystems, including SWIFFT and LPN. [10] has developed an algorithm against NTRU combining lattice reduction and meet-in-the-middle attack, which is close to GBA with two lists.

References

[1] Bernstein, Lange, Peters, and Schwabe. Really fast syndrome-based hashing. 2011. `http://cr.yp.to/codes/rfsb-20110214.pdf`.

[2] D. J. Bernstein. Better price-performance ratios for generalized birthday attacks, 2007. `http://cr.yp.to/rumba20/genbday-20070719.pdf`.

[3] D. J. Bernstein, T. Lange, R. Niederhagen, C. Peters, and P. Schwabe. Implementing wagner's generalized birthday attack against the sha-3 round-1 candidate fsb. Cryptology ePrint Archive, Report 2009/292, 2009. `http://eprint.iacr.org/2009/292`.

[4] D. J. Bernstein, T. Lange, C. Peters, and P. Schwabe. A fast provably secure cryptographic hash function. Cryptology ePrint Archive, Report 2003/230, 2003. `http://eprint.iacr.org/2003/230`.

[5] D. J. Bernstein, T. Lange, C. Peters, and P. Schwabe. Faster 2-regular information-set decoding. 2011. `http://eprint.iacr.org/2011/120`.

[6] A. Blum, A. Kalai, and H. Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, STOC '00, pages 435–440, New York, NY, USA, 2000. ACM.

[7] P. Camion and J. Patarin. The knapsack hash function proposed at crypto'89 can be broken. In *Proceedings of the 10th annual international conference on Theory and application of cryptographic techniques*, EUROCRYPT'91, pages 39–53, Berlin, Heidelberg, 1991. Springer-Verlag. `http://hal.inria.fr/inria-00075097/en/`.

[8] M. Finiasz, P. Gaborit, N. Sendrier, and S. Manuel. SHA-3 proposal: FSB, Oct. 2008. Proposal of a hash function for the NIST SHA-3 competition `http://www-rocq.inria.fr/secret/CBCrypto/fsbdoc.pdf`.

[9] M. Finiasz and N. Sendrier. Security bounds for the design of code-based cryptosystems. In *Proceedings of the 15th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, ASIACRYPT '09, pages 88–105, Berlin, Heidelberg, 2009. Springer-Verlag.

[10] N. Howgrave-Graham. A hybrid lattice-reduction and meet-in-the-middle attack against ntru. In *Proceedings of the 27th annual international cryptology conference on Advances in cryptology*, CRYPTO'07, pages 150–169, Berlin, Heidelberg, 2007. Springer-Verlag.

[11] E. Levieil and P.-A. Fouque. An improved lpn algorithm. In R. De Prisco and M. Yung, editors, *Security and Cryptography for Networks*, volume 4116 of *Lecture Notes in Computer Science*, pages 348–359. Springer Berlin / Heidelberg, 2006. `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.137.3510&rep=rep1&type=pdf`.

[12] V. Lyubashevsky, D. Micciancio, C. Peikert, and A. Rosen. Swifft: A modest proposal for fft hashing. In K. Nyberg, editor, *Fast Software Encryption*, volume 5086 of *Lecture Notes in Computer Science*, pages 54–72. Springer Berlin / Heidelberg, 2008. `http://www.cc.gatech.edu/~cpeikert/pubs/swifft.pdf`.

[13] L. Minder and A. Sinclair. The extended k-tree algorithm. In *Proceedings of the twentieth Annual ACM-SIAM Symposium*

on Discrete Algorithms, SODA '09, pages 586–595, Philadelphia, PA, USA, 2009. Society for Industrial and Applied Mathematics. `http://www.cs.berkeley.edu/~sinclair/ktree.pdf`.

[14] M. J. W. Paul C. van Oorschot. Parallel collision search with cryptanalytic applications. *Journal of Cryptology*, 12:1–28, 1999. 10.1007/PL00003816 `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.59.9389&rep=rep1&type=pdf`.

[15] M.-J. Saarinen. Linearization attacks against syndrome based hashes. In K. Srinathan, C. Rangan, and M. Yung, editors, *Progress in Cryptology INDOCRYPT 2007*, volume 4859 of *Lecture Notes in Computer Science*, pages 1–9. Springer Berlin / Heidelberg, 2007. `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.74.8468&rep=rep1&type=pdf`.

[16] D. Wagner. A generalized birthday problem. 2442:288–304, 2002. `http://www.cs.berkeley.edu/~daw/papers/genbday.html`.