Syndrome Based Collision Resistant Hashing

Matthieu Finiasz

ENSTA

Abstract. Hash functions are a hot topic at the moment in cryptography. Many proposals are going to be made for SHA-3, and among them, some provably collision resistant hash functions might also be proposed. These do not really compete with "standard" designs as they are usually much slower and not well suited for constrained environments. However, they present an interesting alternative when speed is not the main objective. As always when dealing with provable security, hard problems are involved, and the fast syndrome-based cryptographic hash function proposed by Augot, Finiasz and Sendrier at Mycrypt 2005 relies on the problem of Syndrome Decoding, a well known "Post Quantum" problem from coding theory. In this article we review the different variants and attacks against it so as to clearly point out which choices are secure and which are not.

Keywords: hash functions, syndrome decoding, provable security.

1 Introduction

At Mycrypt 2005 Augot, Finiasz and Sendrier proposed a new "provably collision resistant" family of hash functions [1]. This family, called Fast Syndrome Based hash function (or simply FSB), is provably collision resistant in the sense that finding a collision for FSB requires to solve a hard problem of coding theory, namely, the Syndrome Decoding problem. However, even if finding collisions requires to solve an NP-complete problem, some algorithms still exist to solve it and choosing secure parameters for the function turned out to be harder than expected. As a consequence, some attacks were found making some of the originally proposed parameters unsafe. The aim of this article is to review the different FSB variants and the various attacks against them, and to clearly point out which parameters are insecure and which are not.

The FSB construction is based on the Merkle-Damgård design. Therefore, we only describe the compression function which is then iterated in order to obtain the hash function. As a result, if finding collisions for this compression function is hard, then finding collisions for the full hash function will also be hard. The goal of this design is to be able to reduce the problem of finding collisions to the syndrome decoding problem.

The compression function is composed of two sub-functions:

– First a *constant weight encoding* function which takes the s input bits of the compression function and outputs a binary word of length n and Hamming weight w,

J. Buchmann and J. Ding (Eds.): PQCrypto 2008, LNCS 5299, pp. 137–147, 2008.

– Then a *syndrome computation* function which multiplies the previously obtained constant weight word by a $r \times n$ binary matrix $\mathcal{H}$ and outputs the resulting r bits (which in coding theory are usually called a *syndrome*). In practice, as w is usually small, this multiplication is simply the XOR of w columns of $\mathcal{H}$.

Depending on the choice of the constant weight encoding algorithm and of the matrix the FSB hash function can either be faster or safer, depending on what matters the most. Up to now, two different choices have been proposed, and some attacks on specific parameters sets have followed. In this article we first recall these two constructions (see Section 2) and then present all the known attacks against them (see Section 3). In Section 4 we discuss some other issues concerning the FSB construction and eventually propose some up to date candidates in Section 5.

2 Description

As explained, the FSB compression function is composed of two sub-functions : the constant weight encoding function takes s input bits and outputs a word of length n and weight w, the syndrome computation function uses a binary matrix $\mathcal{H}$ of size $r \times n$ and multiplies it by the previous low weight word to output r bits.

2.1 Original Version

The original version of FSB was first presented in [1]. It uses *regular words* for constant weight encoding and matrix $\mathcal{H}$ is a random matrix.

Definition 1. *A* regular word *of weight w and length n is a binary word of length n containing exactly one non-zero bit in each of its w intervals of length $\frac{n}{w}$.*

The reasons for these encoding/matrix choices are quite simple:

– Regular word encoding is the fastest possible constant weight encoding. If $\frac{n}{w}$ is a power of two, then, the constant weight encoding simply consists in reading $\log_2 \frac{n}{w}$ input bits at a time in order to get the index of a column of $\mathcal{H}$.
– A random matrix was chosen for $\mathcal{H}$ for security reasons: it is widely believed among coding theorists that random linear codes have good properties making them hard to decode. As we will see in Section 3, finding collisions for FSB is equivalent to decoding in a code of parity check matrix $\mathcal{H}$, thus, using a random matrix is probably a good choice.

2.2 Quasi-Cyclic Version

The main drawback of the original version is the size of the matrix $\mathcal{H}$. The parameters proposed in [1] all involve using a matrix of at least a few hundred

kilobytes (if not a few megabytes), which is a lot for most constrained environments. For this reason, an improved version was presented in [6], still using regular words for constant weight encoding, but this time using a quasi-cyclic binary matrix $\mathcal{H}$.

Definition 2. *A $r \times n$* quasi-cyclic *matrix is a matrix composed of $\frac{n}{r}$ cyclic blocs. Each $r \times r$ cyclic bloc is such that the i-th line of the bloc is a cyclic shift by $i-1$ positions of the first line of the bloc.*

A quasi-cyclic matrix is thus entirely defined by its first line, and the size of the cyclic blocs.

Quasi-cyclic codes are very interesting as they decrease the size of the description of the hash function a lot (a single line is enough). Moreover, it is proven in [8] that when the size of the cyclic blocs is a prime p (and 2 is a generator of $\mathrm{GF}(p)$), these codes have properties similar to random codes.

Unfortunately, some of the parameters proposed in [6] were awkwardly selected, making them subject to new attacks that we will present in the following section.

3 Known Attacks

The aim of our hash function construction is to be collision resistant. As it uses the Merkle-Damgård design, it is sufficient that our compression function is collision resistant. We thus want to evaluate the complexity of the best collision search algorithms against it. In practice, there are two ways to find a collision in the FSB compression function:

- Either find a collision in the constant weight encoding algorithm: choosing an injective encoding is enough to guarantee that no such collision exists,
- Or find two words of weight w having the same syndrome: that is, find two words c and c' such that $\mathcal{H} \times c = \mathcal{H} \times c'$.

As our compression function needs to compress, this second type of collision always exists, our goal is thus only to make them hard to find!

Problem 1 (Collision search). Given a binary matrix $\mathcal{H}$ and a weight w, find a word c of non-zero weight $\leq 2w$ such that $\mathcal{H} \times c = 0$.

In other words, finding collisions for FSB requires to find a set of $2w$ or less columns of $\mathcal{H}$ which XOR to zero.

3.1 Decoding Attack

The most natural algorithms to solve the collision problem for FSB are decoding algorithms: if one considers $\mathcal{H}$ as the parity check matrix of a binary code, finding a collision consists in looking for a code word of weight $\leq 2w$ (which is roughly equivalent to a decoding problem). Depending on the choice for the matrix $\mathcal{H}$,

this requires to either find a structure in $\mathcal{H}$ making this search easier, or find a low weight code word in a random code (that is, assume that $\mathcal{H}$ contains no specific structure making decoding easier).

For the original version of FSB, a truly random matrix is used for $\mathcal{H}$, therefore, the probability that a structure exists in $\mathcal{H}$ is negligible. For the quasi-cyclic version, an obvious structure exists: the matrix is quasi-cyclic. Nevertheless, if the quasi-cyclic length is well chosen, no specific decoding algorithm is known and decoding can only be done as in a random code. However, as we will see in Section 3.4, a bad choice for the quasi-cyclic length can make the search for low weight code words much easier.

Considering no structure can be found in $\mathcal{H}$, the best decoding algorithm for a random binary code is the Canteaut-Chabaud algorithm [4]. This algorithm is the most advanced of the information set decoding algorithms family and is specifically designed to solve the hardest decoding instances, that is, finding words of weight w close to the Gilbert-Varshamov bound when a single solution exists. Here, the weights we are looking for are much larger (otherwise no compression is possible) which places us in a domain where decoding is somehow easier and where a large number of solutions exist. Giving a closed formula for the complexity of this algorithm is very difficult, especially when many solutions exist, but it is however possible to program an algorithm computing the best possible work factor for a given set of parameters. Additionally, for the domain of parameters we are considering, the Canteaut-Chabaud algorithm is almost always slower than the generalized birthday technique we describe in the next section. When choosing parameters we thus simply checked that the work factor for this attack was above the expected security level, and this was always the case.

3.2 Wagner's Generalized Birthday Technique

Wagner's generalized birthday technique [10] is an extension of the standard birthday collision search technique that uses any power of two number of lists instead of only two. This technique takes advantage of the large number of solutions and looks for specific solutions which can be found more efficiently. It was first applied to FSB in [5].

Standard Birthday Technique. We are looking for $2w$ columns of $\mathcal{H}$ which XOR to 0. These columns are r bits long so we know that if we can build two lists of $2^{\frac{r}{2}}$ elements containing XORs of w columns of $\mathcal{H}$, there is a high probability that these two lists contain an identical element. Building these lists and finding this collision can be done in time/space complexity $O(2^{\frac{r}{2}})$.

Generalized Birthday Technique. With Wagner's generalized technique, one has to build 2^a lists of $2^{\frac{r}{a+1}}$ elements containing XORs of $\frac{w}{2^a}$ columns of $\mathcal{H}$. These lists are then merged pairwise to obtain 2^{a-1} lists of XORs of $\frac{w}{2^{a-1}}$ columns of $\mathcal{H}$. However, in the resulting lists, instead of keeping all the possible elements, only those starting with $\frac{r}{a+1}$ zeros are kept: this way, the size of the lists will not increase. Then, these lists are once again merged pairwise, canceling $\frac{r}{a+1}$

bits again and so on, until only two lists are left and the standard birthday technique can be used. With this technique, collisions can be found in time/space complexity of $O(2^{\frac{r}{a+1}})$, for any value of a such that enough elements are found to populate the 2^a starting lists.

Depending on s and r, the size of the input and output of the compression function, it is easy to evaluate the largest possible value for a, and thus the best possible complexity for this attack. There are s input bits to the function, meaning that 2^s different inputs exist. Thus, 2^s words of weight w can be built. The number L of different words of weight $\frac{w}{2^{a-1}}$ that can be built must thus verify $\binom{L}{2^{a-1}} \leq 2^s$. Additionally, if we want the attack to be possible, the size L of the starting lists must be large enough, meaning that we need $L \geq 2^{\frac{r}{a+1}}$. Thus, any valid parameter a must verify:

$$\binom{2^{\frac{r}{a+1}}}{2^{a-1}} \leq 2^s \iff \frac{r}{a+1} - a + 1 \leq \frac{s}{2^{a-1}} \overset{a \text{ small}}{\iff} \frac{2^{a-1}}{a+1} \leq \frac{s}{r}. \quad (1)$$

For $s = r$, it is interesting to note that $a = 3$ verifies the inequality. If we want the function to compress (that is, $s > r$), $a = 3$ will thus always be possible, and a security higher than $2^{\frac{r}{4}}$ is never possible. This is why a final compression function (see Section 4.2) will always be necessary.

3.3 Linearization

The linearization attack against FSB was presented in [9]. The idea of this attack is that when w becomes large enough, the problem of finding a collision can be linearized in order to reduce it to a linear algebra problem. A collision can then be found in polynomial time!

There are two forms to this attack: first the straight-forward linearization, then a extension making it possible to use this attack in some cases where it could not normally apply.

Simple Linearization. Suppose $w = \frac{r}{2}$. In this case, finding a collision consists in finding r columns of $\mathcal{H}$ XORing to 0. Now, instead of looking for any word of weight r and length n we restrict ourselves to very specific words: each of the r columns of $\mathcal{H}$ will be chosen among one pair of columns, meaning that the i-th column will be either h_i^0 or h_i^1, where all the $h_i^{\{0,1\}}$ are different columns of $\mathcal{H}$. Finding a collision now requires to determine a binary vector B of length r where the i-th bit of B decides which column to choose between h_i^0 and h_i^1. Now comes the linearization: we build a matrix $\mathcal{H}'$ such that the i-th columns of $\mathcal{H}'$ is $h'_i = h_i^1 - h_i^0$. Now, we simply need to find B such that:

$$\mathcal{H}' \times B = \sum_{i=1}^{r} h_i^0.$$

This is a linear system to solve and it is done in polynomial time. Thus, as soon as $2w \geq r$, finding a collision for FSB can be done in polynomial time.

Extension of the Attack. When $w < \frac{r}{2}$, the previous attack can still be applied, but the matrix $\mathcal{H}'$ will no longer be square and the probability that a solution B exists will probably be negligible. To improve this, one can use a larger alphabet: instead of choosing two columns one can choose three columns of $\mathcal{H}$ at a time and code two bits of B with them. However, three columns give three possibilities and two bits of B require four columns (with the fourth column being the XOR of the second and the third). Thus, each solution vector B using extended alphabets will have probability $\frac{1}{4}$ per set of three columns of being invalid. This solution will thus increase the chance that a solution vector B can be found, but will decrease the probability that this solution is realizable in practice. According to [9], if $2w + 2w' = r$ (with $w' \leq w$), the probability that a valid solution is found is:

$$\left(\frac{3}{4}\right)^{2w'} \times 0.28879 \simeq 2^{-0.830w' - 1.792}.$$

This attack is thus usable as soon as $w \geq \frac{r}{4}$, but it will mostly be of interest when w is close to $\frac{r}{2}$.

3.4 Quasi-Cyclic Divisibility

This attack was presented in [7] and it exploits the divisibility of the cycle length in the quasi-cyclic version of FSB. Suppose $\mathcal{H}$ is built of $\frac{n}{r}$ cyclic blocs of size $r \times r$ and there exists a divisor p or r such that $r = p \times r'$. Then, Wagner's generalized birthday technique can be applied on the length r' instead of r. Instead of looking for any word of weight w and length n, we focus on words having a "cyclic" syndrome: each time the i-th column of a bloc of $\mathcal{H}$ is chosen, the $p - 1$ columns at position $i + r'$, $i + 2r'$,... cyclicly to $i - r'$, are also chosen. This way, a bloc-wise cyclic shift by r' positions of the input word keeps it unchanged. This means that the syndrome of a word selected this way also remains unchanged when cyclicly shifted by r' positions. Thus, if the r' top positions of the syndrome are null, the whole syndrome is also null.

Focusing on the r' top rows of $\mathcal{H}$ and selecting only the previously described words, we now need to apply Wagner's technique to the following problem: find $\frac{w}{p}$ columns of a $r' \times \frac{n}{p}$ binary matrix XORing to 0. The largest possible value for the parameter a of Wagner's algorithm is smaller than before, but the attack applies to r' bits only and the final complexity drops significantly. When selecting parameters for FSB, it is important that such an attack cannot be applied.

4 Other Issues

4.1 IV Weakness

As pointed out in [7] another weakness of the original FSB compression function lies in the way the input to the compression function is handled. In particular,

the chaining bits (or IV) and the message bits are simply concatenated, and no mixing whatsoever is applied. When using regular words, this means that the output of the compression function is simply the XOR of two independent hashes: one resulting from the IV bits, the other one from the message bits. If one can find a collision on the message part of the compression function (this will be somehow harder than a normal collision as less input bits are available), then this collision is IV independent. This has no influence on the collision resistance of the function, but it is a problem when using the hash function as a MAC or as a PRF for example: the resistance to some attacks falls from the cost of an inversion (or second preimage) to the cost of building a message only collision (which will probably be just above the cost for building a standard collision).

In order to avoid such problems, the best thing would be to mix the input bits through a diffusion function. However, such a mixing is quite costly and would severely reduce the throughput of the hash function. The best solution is thus probably to integrate this diffusion in the constant weight encoding function. As stated in Section 5.1, a simple interleaving of the message bits with the IV bits is enough to avoid this problem.

4.2 Final Compression Function

Another issue with the FSB compression function is the ratio between the security against collision and the output size of the function. A hash function is expected to have a security of $2^{\frac{r}{2}}$ against collisions if it outputs r-bits hashes. With FSB, we have seen in Section 3.2 that if our compression function is to compress, an attack in $2^{\frac{r}{4}}$ will always be possible. The solution is thus to use a final compression function: use FSB to hash the message into a large hash, and then use a final compression function g to reduce the size of the hash from r to r' bits, where r' is twice the bit security of FSB against collisions. However, finding a suitable function g is not straight-forward:

- If g is collision resistant, then using FSB hash and then g will lead to a collision resistant hash function. However, even if g is not collision resistant, building a collisions on the complete hash from a collision on g will not be easy: it requires to invert FSB. So requiring collision resistance for g is clearly too much.
- If g is a linear function, then g can be applied to all the columns of $\mathcal{H}$ and finding a collision on the whole function will only require to find a collision on a matrix $\mathcal{H}'$ of size $r' \times n$. Thus the security against collisions will be less than $2^{\frac{r'}{4}}$.

Apart from this, it is hard to state anything relevant about the final compression function and we believe that most non-linear compression function could do the trick. However, as far as provable security is concerned, choosing a provably collision resistant function g is probably the only choice at the moment.

5 Possible Candidates

5.1 Constant Weight Encoding

There are many ways to perform constant weight encoding, spanning from the one to one encoding where all words of weight w are equiprobable, to the regular word encoding. The first one is the most bit efficient (the compression function will have the largest possible input for some given parameters n and w), the second one is the fastest. When dealing with hash functions, speed is usually a very important factor and fast constant weight encoding would be a natural choice, however, concerning security, all results on the hardness of syndrome decoding consider random words of weight w, not regular words (or words with any other structure). Luckily, when looking for collisions, a collision for any given constant weight encoding is also a collision for the one to one encoding: any pair of words of weight w (even with a strong structure) can be coded with the one to one equiprobable encoding. Thus, finding collisions for FSB using regular words can not be easier than finding collisions for FSB using a more bit efficient encoding.

However, no proof can be given that finding collisions for regular words is indeed harder than with the one to one equiprobable encoding. Thus, when choosing parameters for FSB, we will consider the security of FSB with one to one constant weight encoding, even if a faster encoding is used in practice.

The conclusion of this is that using regular word encoding is certainly the best choice for efficiency. However, as seen in Section 4.1, using such an encoding causes IV weakness issues. In order to avoid these issues it is necessary that every index of a non-zero bit of the constant weight word depends from the value of both the IV and the message. This way, no IV independent collision can be built. Interleaving the bits coming from the IV (or chaining value) with those of the message is thus a solution. Depending on the parameters chosen for the function, different interleavings will be possible.

5.2 Matrix Choice

The choice of the matrix $\mathcal{H}$ is also very important for the efficiency of FSB. Of course, for optimal security, nothing can be better than a truly random matrix, but in this case the description of the hash function will be very large and will not be suitable for memory constraint devices and for most hardware implementations. Thus, using matrices that can be described with a single line is important if FSB is ever to be used in practice.

The results of Gaborit and Zémor [8] tend to prove that well chosen quasi-cyclic codes have good properties making them suitable candidates. However, this requires that r is a prime number, which will certainly make implementation less efficient than using powers of two. Our idea is thus to use a *truncated quasi-cyclic* matrix instead of a standard quasi-cyclic matrix.

Definition 3. *A $r \times n$ matrix $\mathcal{H}$ is a* truncated quasi-cyclic *matrix if it can be divided in $\frac{n}{r}$ blocs of size $r \times r$ and each bloc is the top left sub-bloc of a $p \times p$ cyclic bloc.*

With this definition, any matrix $\mathcal{H}$ built of blocks which are Toeplitz matrices will be a truncated quasi-cyclic matrix with $p > 2r$, but in order to be as close as possible to standard quasi-cyclic matrices, we will always choose r very close to p. Then, the description of the $r \times n$ matrix $\mathcal{H}$ can be reduced to a "first line" of $\frac{n}{r} \times p$ bits and the values of p and r.

As explained in [8], in order for p to be a suitable choice it must be prime, and 2 must be a generator of GF(p). Hence, it is easy to check the best p for a given r: one simply needs to test the primes greater than r one by one until 2 is a generator. For example, for $r = 512$ we get $p = 523$, for $r = 768$ we get $p = 773$ and for $r = 1024$ we get $p = 1061$.

5.3 Choosing Parameters

When choosing parameters we want Wagner's attack to be the most efficient so that we precisely control the security of FSB. As seen in Section 3.2, this security only depends on the output size r and the input size s of the compression function. As stated in Section 5.1 we want to measure the security of the construction when using a one to one constant weight encoding, which means, s is not the effective number of bits that the compression function will read, but $s = \log_2 \binom{n}{w}$. Once r and s are chosen to obtain the desired security level, one simply needs to select a convenient value for w (and deduce n), such that linearization attacks are impossible.

Concerning previously proposed parameters, all those proposed in [6] use a quasi-cyclic matrix of cyclicity a power of 2 and are thus all subject to the attack of Section 3.4: none of these parameters should be used. Three sets of parameters were proposed in the original paper [1]:

- Short hash has its security reduced to $2^{72.2}$ as the security gained from using regular words is no longer taken into account.
- Fast hash is subject to the extended linearization attack and has its security reduced to $2^{59.9}$.
- The Intermediate proposal however still has a security above 2^{80} and can still be considered safe.

Parameters for 80-bit Security. Choosing $r = 512$ and a security of 2^{80} against collisions we get from Equation (1) that $s \leq 1688$. Now, to avoid linearization attacks we need $w \leq \frac{r}{4} = 128$. If we choose $w = 128$, we get for $n = 2^{18}$ a value $s = 1587$ which is suitable. Our first proposition is thus to use:

$$r = 512, n = 2^{18}, w = 128,$$

with regular word encoding, and a truncated quasi-cyclic matrix with $p = 523$. For the IV interleaving, each of the w positions are coded by 11 input bits, 4 of which are taken from the IV and the rest from the message. With these parameters FSB reads input blocs of 896 bits and outputs 512 bits. These bits can then be compressed to 160 bits using a suitable final compression function. The matrix $\mathcal{H}$ is described by 267 776 bits ($\sim$ 32.7kB).

Parameters for 128-bit Security. For 128-bit security we need r larger than 512. We can use $r = 768$ and obtain $s \leq 2048$. If we pick $w = 192$ and $n = 3 \times 2^{15}$ we get $s = 1999$ and linearization attacks are impossible. Our proposition is to use:

$$r = 768, n = 3 \times 2^{15}, w = 192,$$

with regular word encoding, and a truncated quasi-cyclic matrix with $p = 773$. Each position is coded using 9 input bits, so the IV interleaving will take 4 bits from the IV and 5 bits from the message each time. FSB thus reads input blocs of 960 bits and output 768 bits which, at the end, need to be compressed to 256 bits. The matrix $\mathcal{H}$ is described by 98 944 bits ($\sim$ 12kB).

The same parameters, using a shorter $n = 3 \times 2^{14}$ will probably be more efficient as each position will be coded with 8bits, 4 from the IV and 4 from the message, even if only 768 bit blocs are read instead of 960 bit blocs. Moreover, it will have a shorter description ($\sim$ 6kB) and the security against collisions will be a little higher (about 2^{133}).

6 Conclusion

Taking into account all the different attacks against FSB, it is still possible to select parameters that offer both a high level of security (relying on well identified problems) and a satisfying efficiency. Also, apart from the choice of the final compression function, the other choices that had to be made for FSB seem clear: use regular word encoding (with IV interleaving) and a truncated quasi-cyclic matrix. For the final compression function, using a provably secure pseudo-random generator could be a good choice: use the output of FSB as an IV and generate the desired number of bits of output. One could then use the generators of Blum-Blum-Shub [3], or preferably for post-quantum security QUAD [2].

References

1. Augot, D., Finiasz, M., Sendrier, N.: A family of fast syndrome based cryptographic hash functions. In: Dawson, E., Vaudenay, S. (eds.) Mycrypt 2005. LNCS, vol. 3715, pp. 64–83. Springer, Heidelberg (2005)
2. Berbain, C., Gilbert, H., Patarin, J.: QUAD: a practical stream cipher with provable security. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 109–128. Springer, Heidelberg (2006)
3. Blum, L., Blum, M., Shub, M.: Comparison of two pseudo-random number generators. In: Chaum, D., Rivest, R.L., Sherman, A. (eds.) Crypto 1982, pp. 61–78. Plenum (1983)
4. Canteaut, A., Chabaud, F.: A new algorithm for finding minimum-weight words in a linear code: Application to McEliece's cryptosystem and to narrow-sense BCH codes of length 511. IEEE Transactions on Information Theory 44(1), 367–378 (1998)

5. Coron, J.-S., Joux, A.: Cryptanalysis of a provably secure cryptographic hash function. IACR eprint archive (2004), http://eprint.iacr.org/2004/013
6. Finiasz, M., Gaborit, P., Sendrier, N.: Improved fast syndrome based cryptographic hash functions. In: Rijmen, V. (ed.) ECRYPT Workshop on Hash Functions (2007)
7. Fouque, P.-A., Leurent, G.: Cryptanalysis of a hash function based on quasi-cyclic codes. In: Malkin, T. (ed.) CT-RSA 2008. LNCS, vol. 4964, pp. 19–35. Springer, Heidelberg (2008)
8. Gaborit, P., Zémor., G.: Asymptotic improvement of the Gilbert-Varshamov bound for linear codes. In: IEEE Conference, ISIT 2006, pp. 287–291 (2006)
9. Saarinen, M.-J.O.: Linearization attacks against syndrome based hashes. In: Srinathan, K., Rangan, C.P., Yung, M. (eds.) INDOCRYPT 2007. LNCS, vol. 4859, pp. 1–9. Springer, Heidelberg (2007)
10. Wagner, D.: A generalized birthday problem. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 288–304. Springer, Heidelberg (2002)