A Fast Provably Secure Cryptographic Hash Function

Daniel Augot, Matthieu Finiasz, and Nicolas Sendrier

Projet Codes, INRIA Rocquencourt
BP 105, 78153 Le Chesnay - Cedex, France
[Daniel.Augot,Matthieu.Finiasz,Nicolas.Sendrier]@inria.fr

Abstract. We propose a family of fast and provably secure cryptographic hash functions. The security of these functions relies directly on the well-known syndrome decoding problem for linear codes. Attacks on this problem are well identified and their complexity is known. This enables us to study precisely the practical security of the hash functions and propose valid parameters for implementation. Furthermore, the design proposed here is fully scalable, with respect to security, hash size and output rate.

Key Words: hash functions, syndrome decoding, NP-completeness.

1 Introduction

The main cryptographic hash function design in use today iterates a so called compression function according to Merkle's and Damgård's constructions [5, 13]. Classical compression functions are very fast [3, 14, 16] but cannot be proven secure. However, provable security may be achieved with compression functions designed according to public key principles, at the cost of a poor efficiency.

Unlike most other public key cryptosystems, the encryption function of the McEliece cryptosystem [11] (or of Niederreiter's version [15]) is nearly as fast as a symmetric cipher. Using this function with a random matrix instead of the usual Goppa code parity check matrix, we obtain a provably secure one-way function with no trap. The purpose of this paper is to use this function to obtain a fast cryptographic hash function whose security is assessed by a difficult algorithmic problem.

For didactic purposes, we introduce the *Syndrome Based* (SB) compression function, which directly relies on Niederreiter's scheme and the syndrome decoding problem. However, this function can hardly be simultaneously fast and secure for practical parameters. Hence we introduce the *Fast Syndrome Based* (FSB) compression function, derived from the previous one and relying on a similar hard problem. Section 2 is devoted to the description of both functions. In Section 3 we show that, as for McEliece's and Niederreiter's cryptosystems, the security of SB can be reduced to the hardness of syndrome decoding in the average case. Similarly, we prove that the security of FSB is reduced to the average case difficulty of two new NP-complete problems. Finally, in Section 4, we show how the best known decoding techniques can be adapted to the cryptanalysis of our functions. From that we can evaluate the practical security and the scalability of the system, and eventually propose a choice of parameters. Note that, for clarity of the presentation, NP-completeness proofs are postponed in the appendix.

2 The Hash Functions

We will present two different versions of the hash function: the first one is the *Syndrome Based* hash function (SB), the second, a modified version, is called *Fast Syndrome Based* (FSB) and is much faster in practice, but also more secure.

2.1 General Construction

There is one main construction for designing hash functions: it consists in iterating a compression function which takes as input s bits and returns r bits, with $s > r$, so that, using such a chaining, the resulting function can operate on strings of arbitrary length (see Fig. 1). The validity of such a design has been well established [5, 13] and its security is not worse than the security of the compression function. There for we will concentrate on the security of the latter.

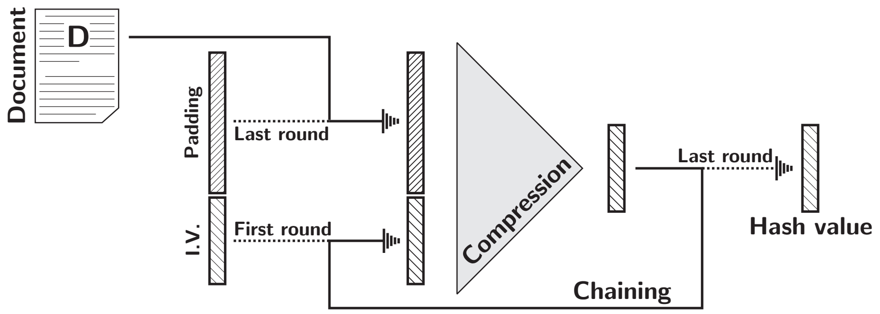


Fig. 1. A standard hash function construction

2.2 Description of the Syndrome Based Compression Function

The core of the compression function is a random binary matrix $\mathcal{H}$ of size $r \times n$. Hashing a document will consist in adding (using binary XORs) w columns of this matrix to finally obtain a hash of length r.

The parameters for the hash function are:

- n the number of columns of matrix $\mathcal{H}$;
- r the number of rows of matrix $\mathcal{H}$ and the size in bits of the function output;
- w the number of columns of $\mathcal{H}$ added at each round.

Once these parameters are chosen, a (truly) random $r \times n$ matrix $\mathcal{H}$ is generated. This matrix is chosen once and for all for the hash function. Using the scheme explained next a function with input size $s = \log_2 \binom{n}{w}$ and output size r is obtained.

As we use a standard chaining method, the input of the compression function will consist of r bits taken from the output of the previous round and $s - r$ bits taken from the file. Of course, w must be such that $s > r$.

The compression is performed in two steps:

Input: s bits of data
1. encode the s bits in a word e of length n and weight w;
2. compute $\mathcal{H}e^T$ to obtain a binary string of length r.
Output: r bits of hash

The first step requires to convert the s input bits into a binary n-tuple containing exactly w ones (and the rest of zeros everywhere else). Then multiply this word by $\mathcal{H}$ (add the corresponding w columns of $\mathcal{H}$) to obtain the r bit hash.

This function is expected to be very fast as only a few operations are required: input encoding and a few XORs. In practice, the second step will be very fast, but the first step is much slower. Indeed, the best algorithm for embedding data in a constant weight word [6, 7] makes extensive use of large integer arithmetics and is by far the most expensive part.

2.3 Description of the Fast Syndrome Based Compression Function

Definition 1. *A word of length n and weight w is called* regular *if it has exactly one non-zero position in each of the w intervals* $\left[\!\left[(i-1)\frac{n}{w}; i\frac{n}{w}\right]\!\right]_{i=1..w}$.

To improve the speed, we embed less data in each constant weight word by using a faster, no longer one-to-one, *constant weight encoder*. Instead of using any word of weight w we embed the input bits in a *regular* word of weight w. Hence we will have $s = w \log_2(n/w)$.

The matrix $\mathcal{H}$ is split into w sub-blocks $\mathcal{H}_i$ of size $r \times \frac{n}{w}$ and the algorithm is the following:

Input: s bits of data
1. split the s input bits in w parts $s_1, \ldots, s_w$ of $\log_2(n/w)$ bits;
2. convert each s_i to an integer between 1 and $\frac{n}{w}$;
3. choose the corresponding column in each $\mathcal{H}_i$;
4. add the w chosen columns to obtain a binary string of length r.

Output: r bits of hash

Using this encoder the cost of the first step becomes negligible as it only consists in reading the input bits a fixed number at a time. This compression function is hence very fast and its speed is directly linked to the number of XORs required for a round.

2.4 Related Work

In Merkle's construction [13], the compression function is an encryption function (modular squaring, knapsack...), used as a one-way function. In the case of the SB hash function the compression function is very similar to the encoding function of Niederreiter's version of the McEliece cryptosystem [11, 15]. The only difference is that, instead of using the parity check matrix of a permuted Goppa code, SB uses a random matrix $\mathcal{H}$. Doing this, the trap in the one-way function of the cryptosystem was removed.

From a security point of view this can only strengthen the system as all attacks on the trap no longer hold. Breaking Niederreiter's cryptosystem can be reduced to the two problems of inverting the one-way function or recovering the trap. In the case of SB, only the inversion problem remains.

3 Theoretical Security

As stated in [12], a cryptographic hash function has to be pre-image resistant, second pre-image resistant and collision resistant. As the second pre-image resistance is strictly weaker than collision resistance, we will only check that both hash functions are collision free and resistant to inversion. In the SB hashing scheme this can be reduced to solving an instance of *Syndrome Decoding*, which is NP-complete [2]. In the FSB version, these two kinds of attack can be reduced to two very close new problems. We will first describe them and show (in appendix) that they are also NP-complete.

We will then show that finding a collision or an inversion is at least as hard as solving one of these two new problems. This is what we call provable security.

3.1 Two New NP-complete Problems

In this section we will recall the problems of syndrome decoding and null syndrome decoding and then describe two closely related new problems.

Syndrome Decoding (SD)
Input: a binary matrix $\mathcal{H}$ of dimension $r \times n$ and a bit string $\mathcal{S}$ of length r.
Property: there exists a set of w' columns of $\mathcal{H}$ adding to $\mathcal{S}$ (with $0 < w' \leq w$).

Null Syndrome Decoding (NSD)
Input: a binary matrix $\mathcal{H}$ of dimension $r \times n$.
Property: there exists a set of w' columns of $\mathcal{H}$ adding to 0 (with $0 < w' \leq w$).

These two problems are NP-complete [2], which means that at least some instances of the problem are difficult. However, it is a common belief that they should be *difficult in average* (for well chosen parameter ranges), which means that random instances are difficult. For cryptographic purposes this is much more interesting than simple NP-completeness.

For the following two problems the same comment can be made. They are NP-complete (see appendix A) and we believe that they are hard in average.

Regular Syndrome Decoding (RSD)
Input: w matrices $\mathcal{H}_i$ of dimension $r \times n$ and a bit string $\mathcal{S}$ of length r.
Property: there exists a set of w columns, one in each $\mathcal{H}_i$, adding to $\mathcal{S}$.

2-Regular Null Syndrome Decoding (2-RNSD)
Input: w matrices $\mathcal{H}_i$ of dimension $r \times n$.
Property: there exists a set of $2w'$ columns (with $0 < w' \leq w$), 0 or 2 in each $\mathcal{H}_i$, adding to 0.

3.2 Security Reduction

In this section we will show how finding collisions or inverting any of the two proposed hash function is as hard as solving an instance of one of the NP-complete problems described in the previous section.

We will prove the security of the compression function, which is enough when using a standard construction (see [12] p. 333).

Security of the Syndrome Based Hash Function. Finding an *inversion* for this compression function consists in finding an input (of length s) which will hash to a given bit string $\mathcal{S}$. Now suppose an algorithm $\mathcal{A}$ is able to compute inversions for this function, and an instance $(\mathcal{H}, w, \mathcal{S})$ of the SD problem has to be solved. Then, using $\mathcal{A}$, it is possible to compute inverses for the compression function using $\mathcal{H}$, and so obtain an input with hash $\mathcal{S}$. This means that the w columns corresponding to this input, when added together, sum to $\mathcal{S}$. A solution to the given instance of SD has been found.

Finding a *collision* for this scheme consists in finding two different inputs hashing to the same string. Now suppose an algorithm $\mathcal{A}'$ is able to find collisions for this compression function, and a given instance $(\mathcal{H}, 2w)$ of the NSD problem has to be solved. The algorithm $\mathcal{A}'$ can compute two inputs hashing (through $\mathcal{H}$) to the same string. These inputs correspond to two different words of weight w which when added together give a word m of weight $\leq 2w$. Due to linearity, the product $\mathcal{H}m^T$ is 0. The word m is a solution to the instance of NSD.

So, finding either a collision or an inversion for the SB hash function is at least as hard as solving an instance of the SD or NSD problems.

Security of the FSB Hash Function. For this version of the hash function the security reduction can be done exactly like for the SB function. Inverting the compression function can be reduced to the RSD problem, and collision to the 2-RNSD problem.

These reductions to NP-complete problems do not prove that all instances are difficult but only that some instances are difficult. For a cryptographic security this is clearly not enough. However, in the same manner as Gurevich and Levin [8, 10] have discussed it for SD, we believe that all these NP-complete problems are difficult in average (for well chosen parameters).

4 Practical Security

This section is dedicated to the study of the practical security of the two versions of the hash function. As for the security reduction, attacking the hash function as a whole is equivalent to attacking a single round and takes the same amount of computation. Therefore we need to identify the possible attacks on one round of the system, and then study the minimal workfactors required to perform these attacks.

4.1 Existing Attacks

Decoding in a random linear code is at least as hard as giving an answer to SD. This problem has been extensively studied along the years and many attacks against it have been developed (see [1]): split syndrome decoding, gradient-like decoding, information set decoding... All these attacks are exponential. Still, as stated by Sendrier [17], the most efficient attacks seem to be all derived from *Information Set Decoding* (ISD).

Definition 2. *An* information set *is a set of $k = n - r$ (the dimension of the code) positions among the n positions of the support.*

Definition 3. *Let $(\mathcal{H}, w, \mathcal{S})$ be an instance of SD. An information set will be called* valid *if there exists a solution to this SD problem which has no 1s among the chosen k positions.*

The ISD technique consists in picking information sets, until a valid one is found. Checking whether the information set is valid or not mainly consists in performing a Gaussian elimination on an $r \times r$ submatrix of the parity check matrix $\mathcal{H}$ of the code. Once this is done, if a solution exists it is found in constant time. Let $GE(r)$ be the cost of this Gaussian elimination and $\mathcal{P}_w$ the probability for a random information set to be valid. Then the complexity of this algorithm is $GE(r)/\mathcal{P}_w$.

This technique was improved several times [4, 9, 18]. The effect of these improvements on the complexity of the algorithm is to reduce the degree of the polynomial part. The complexity is then $g(r)/\mathcal{P}_w$, with $d^\circ(g) < d^\circ(GE)$. However, all the different versions were designed for instances of SD having only one (or a few) solutions. For SB, as we will see in section 4.4, the range of parameters we are interested in leads to instances with many more solutions (over 2^{400}). Anyhow, ISD attacks still be the most suitable and behave the same way as when there is a single solution.

Applied to the FSB hash function, this technique will, in addition, have to return regular words. This will decrease considerably the number of solutions and, in this way, decrease the probability for a given information set to be valid, thus enhancing security.

4.2 Analysis of Information Set Decoding

We have seen that the complexity of an information set decoding attack can be expressed as $g(r)/\mathcal{P}_w$. Hence, it is very important to evaluate precisely $\mathcal{P}_w$ in both versions of the hash function and for both kind of attacks. The polynomial part has less importance and is approximately the same in all four cases.

The probability $\mathcal{P}_w$ we want to calculate will depend on two things: the probability $\mathcal{P}_{w,1}$ that a given information set is valid for one given solution of SD, and the expected number $\mathcal{N}_w$ of valid solutions for SD. Even though the probabilities we deal with are not independent, we shall consider

$$\mathcal{P}_w = 1 - (1 - \mathcal{P}_{w,1})^{\mathcal{N}_w}.$$

It is important to note that $\mathcal{N}_w$ is the average number of valid solutions to SD. For small values of w, $\mathcal{N}_w$ can be much smaller than one; the formula is valid though.

In this section, for the sake of simplicity, we will use the approximation $\mathcal{P}_w \simeq \mathcal{P}_{w,1} \times \mathcal{N}_w$. When calculating the security curves (see section 4.3) and choosing the final parameters (see 4.4), we have used the exact formulas for the calculations.

Solving SD: attacking the SB hash function. For *inversion*, the problem is the following: we have a syndrome which is a string $\mathcal{S}$ of r bits. We want to find an information set (of size $k = n - r$) which is valid for one inverse of $\mathcal{S}$ of weight w.

For one given inverse of weight w the probability for a given information set to be valid is:

$$\mathcal{P}_{w,1} = \frac{\binom{n-w}{k}}{\binom{n}{k}} = \frac{\binom{n-k}{w}}{\binom{n}{w}} = \frac{\binom{r}{w}}{\binom{n}{w}}.$$

The average number of solutions (of inverses) of weight w is:

$$\mathcal{N}_w = \frac{\binom{n}{w}}{2^r}.$$

In the end, we get a total probability of choosing an information set valid for inversion of:

$$\mathcal{P}_{\text{inv}} = \mathcal{P}_{w,1} \times \mathcal{N}_w = \frac{\binom{r}{w}}{2^r}.$$

To find a *collision*, it is enough to find a word of even weight $2i$ with $0 < i \leq w$. We have exactly the same formulas as for inversion, but this time the final probability is:

$$\mathcal{P}_{\text{col}} = \sum_{i=1}^{w} \frac{\binom{r}{2i}}{2^r}.$$

Solving RSD: inverting the FSB hash function. In that case, one needs to find a *regular* word of weight w having a given syndrome $\mathcal{S}$. Since the word is regular there will be less solutions for each syndrome, and even if each of them is easier to find, in the end the security is increased compared to SB.

The number of regular solutions to RSD is, in average:

$$\mathcal{N}_w = \frac{\left(\frac{n}{w}\right)^w}{2^r}.$$

The probability of finding a valid information set for a given solution is however a little more intricate. For instance, as the solutions are not random words, the attacker shouldn't choose the information set at random, but should rather choose the sets which have the best chance of being valid. In our case it is easy to see that the attacker will maximize his chances when taking the same number of positions in each block, that is, taking k/w positions w times. The probability of success is then:

$$\mathcal{P}_{w,1} = \left(\frac{\binom{n/w-1}{k/w}}{\binom{n/w}{k/w}} \right)^w = \frac{\left(\frac{r}{w}\right)^w}{\left(\frac{n}{w}\right)^w}.$$

The final probability is:

$$\mathcal{P}_{\text{inv}} = \mathcal{P}_{w,1} \times \mathcal{N}_w = \frac{\left(\frac{r}{w}\right)^w}{2^r}.$$

One can check that this probability is much smaller than for the SB hash function (a ratio of $w!/w^w$). Using the fast constant weight encoder, and restricting the set of solutions to RSD, has strengthened the system.

Solving 2-RNSD: collisions in the FSB hash function. When looking for *collisions* one needs to find two regular words of weight w having the same syndrome. However these two words can coincide on some positions. With the block structure of regular words, this means that we are looking for words with a null syndrome having some blocks (say i) with a weight of 2 and the remaining blocks with a weight of 0. The number of such words is:

$$\mathcal{N}_i = \frac{\binom{w}{i}\binom{n/w}{2}^i}{2^r}.$$

Once again, the attacker can choose his strategy when choosing information sets. However this time the task is a little more complicated as there is not a single optimal strategy. If the attacker is looking for words of weight up to $2w$ then the strategy is the same as for RSD, choosing an equal amount of positions in each set. For each value of i, the probability of validity is then:

$$\mathcal{P}_{i,1} = \frac{\binom{w}{i}\binom{n/w-k/w}{2}^i}{\binom{w}{i}\binom{n/w}{2}^i} = \frac{\binom{r/w}{2}^i}{\binom{n/w}{2}^i}.$$

The total probability of success for one information set is then:

$$\mathcal{P}_{\text{col total}} = \sum_{i=1}^{w} \frac{\binom{w}{i}\binom{r/w}{2}^i}{2^r} = \frac{1}{2^r}\left[\binom{r/w}{2}+1\right]^w.$$

But the attacker can also decide to focus on some particular words. For example he could limit himself to words with non-zero positions only in a given set of $w_0 < w$ blocks, take all the information set points available in the remaining $w - w_0$ blocks and distribute the rest of the information set in the w_0 chosen blocks. He then works on less solutions, but with a greater probability of success. This probability is:

$$\mathcal{P}_{\text{col } w_0} = \frac{1}{2^r}\left[\binom{\frac{n}{w}-\frac{k_0}{w_0}}{2}+1\right]^{w_0},$$

with $\quad k_0 = k - (w - w_0) \times \dfrac{n}{w} = \dfrac{n\ w_0}{w} - r$. This can be simplified into:

$$\mathcal{P}_{\text{col } w_0} = \frac{1}{2^r}\left[\binom{\frac{r}{w_0}}{2}+1\right]^{w_0}.$$

Surprisingly, this no longer depends of w and n. As the attacker has the possibility to choose the strategy he prefers (depending on the parameters of the system) he will be able to choose the most suitable value for w_0. However, as $w_0 \leq w$ he might not always be able to take the absolute maximum of $\mathcal{P}_{\text{col } w_0}$.

The best he can do will be:

$$\mathcal{P}_{\text{col optimal}} = \frac{1}{2^r} \max_{w_0 \in [\![1;w]\!]} \left[\binom{\frac{r}{w_0}}{2}+1\right]^{w_0}.$$

This maximum will be reached for $w_0 = \alpha \cdot r$ where $\alpha \approx 0.24$ is a constant. Hence, if $w > 0.24r$ we have:

$$\mathcal{P}_{\text{col optimal}} = \frac{1}{2^r}\left[\binom{\frac{1}{\alpha}}{2}+1\right]^{\alpha r} \simeq 0.81^r$$

4.3 Some security curves

These are some curves obtained when plotting the exact versions of the formulas above. For $n = 2^{14}$ and $r = 160$, we see on Fig. 2 that the security of the FSB hash function is much higher than that of the SB version. This is the case for the chosen parameters but also for about any sound set of parameters.

Fig. 3 focuses on the security of the FSB hash function against inversion and collision. We see that by choosing different parameters for the hash function we can obtain different security levels. This level does not depend significantly on n or w but is mainly a function of r. With $r = 160$ we get a probability of $2^{-47.7}$ that an information set is valid. With $r = 224$ we decrease this probability to $2^{-66.7}$. This may seem far below the usual security requirements of 2^{80} binary

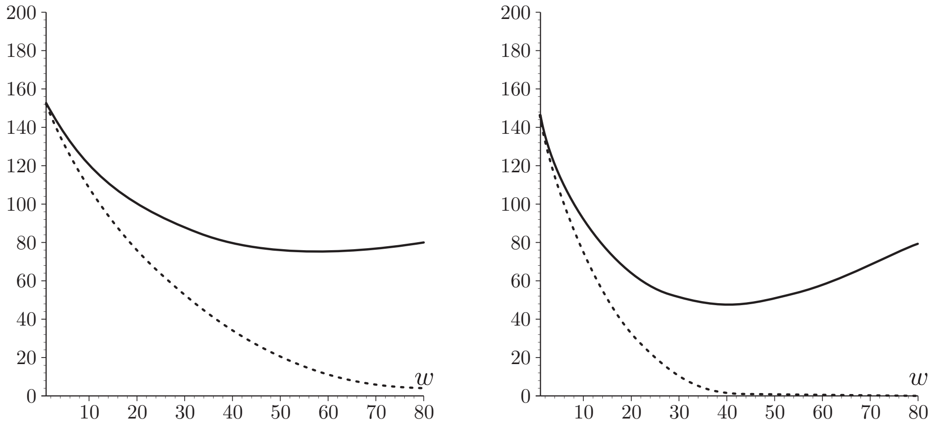


Fig. 2. These two curves show the $\log_2$ of the inverse of the probability that an information set is valid, as a function of w. The dotted line corresponds to the SB version of the scheme and the plain line to the FSB version. On the left the attack is made for inversion, on the right for collision. The curves correspond to the parameters $n = 2^{14}$, $r = 160$.

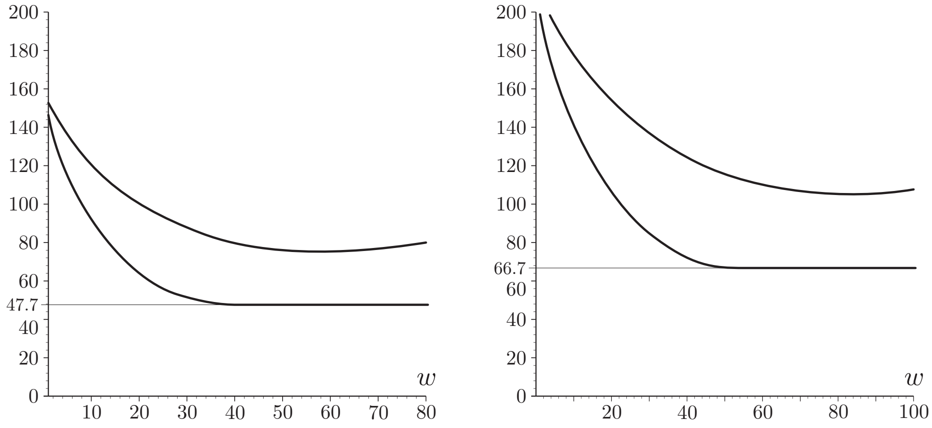


Fig. 3. On the left is plotted the security (inverse of the probability of success) of the FSB hash function for the parameters $n = 2^{14}$, $r = 160$ when using the optimal attacks. The curve on the right corresponds to $n = 3 \cdot 2^{13}$, $r = 224$.

operations, however, once an information set is chosen, the simple fact of verifying whether it is valid or not requires to perform a Gaussian elimination on a small $r \times r$ matrix. This should take at least $O(r^2)$ binary operations. This gives a final security of $2^{62.3}$ binary operations for $r = 160$ which is still too small. For $r = 224$ we get $2^{82.3}$ operations which can be considered as secure.

For extra strong security, when trying to take into account some statements in appendix B, one could try to aim at a probability below 2^{-80} (corresponding to an attacker able to perform Gaussian eliminations in constant time) or 2^{-130} (for an attacker using an idealistic algorithm). This is achieved, for example, with $r = 288$ with a probability of choosing a valid information set of $2^{-85.8}$ or $r = 512$ with a probability of $2^{-152.5}$. These levels of security are probably far above what practical attacks could do, but it is interesting to see how they can be reached using reasonable parameters.

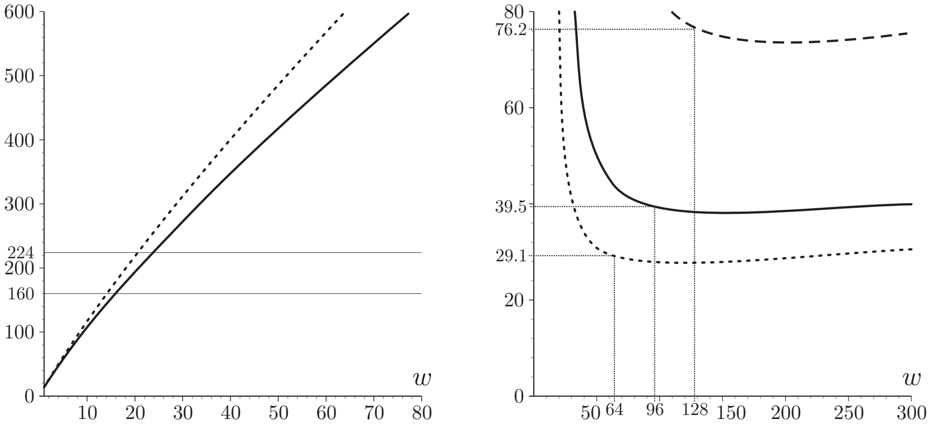


Fig. 4. On the left is the number of bits of input for one round of the SB (dotted line) or the FSB (plain line) hash functions as a function of w. These are calculated for a fixed $n = 2^{14}$. On the right are the curves corresponding to the number of XORs per bit of input as a function of w for the FSB hash function with $n = 2^{14}$, $r = 160$ (dotted line), $n = 3 \cdot 2^{13}$, $r = 224$ (plain line) and $n = 2^{13}$, $r = 288$ (dashed line).

In terms of efficiency, what we are concerned in is the required number of binary XORs per bit of input. We see on Fig. 4 (left) that the FSB version of the hash function is a little less efficient than the SB version as, for the same number of XORs, it will read more input bits. The figure on the right shows the number of bit XORs required for one bit of input. It corresponds to the following formula:

$$\mathcal{N}_{XOR} = \frac{r \cdot w}{w \log_2(n/w) - r}.$$

This function will always reach its minimum for $w = r \cdot \ln 2$, and values around this will also be nearly optimal as the curve is quite flat. A smaller w will moreover yield smaller block size for a same hashing cost, which is better when hashing small files.

Note that $\mathcal{N}_{XOR}$ can always be reduced to have a faster function by increasing n, however this will increase the block size of the function and will also increase the size of the random matrix to be used. In software, as soon as this matrix becomes larger than the machine's memory cache size, the speed will immediately drop as the number of cache misses will become too large.

4.4 Proposed Parameters for Software Implementation

The choice of all the parameters of the system should be done with great care as a bad choice could strongly affect the performances of the system. One should first choose r to have the output size he wishes and the security required. Then, the choice of w and n should verify the following simple rules:

- $\frac{n}{w}$ is a power of 2, so as to read an integer number of input bits at a time. This may be 2^8 to read the input byte by byte.
- $n \times r$ is smaller than the cache size to avoid cache misses.
- w is close to its optimal value (see Fig. 4).
- r is a multiple of 32 (or 64 depending on the CPU architecture) for a full use of the word-size XORs

Thus we make three proposals. Using $r = 160$, $w = 64$ and $n = 2^{14}$ and a standard well optimized C implementation of the FSB we obtained an algorithm faster than the standard md5sum linux function and nearly as fast as a C implementation of SHA-1. That is a rate of approximately 300 Mbits of input hashed in one second on a 2GHz pentium 4. However, these parameters are not secure enough for collision resistance (only $2^{62.3}$ binary operations). They could nevertheless be used when simply looking for pre-image resistance and higher output rate, as the complexity of inverting this function remains above the limit of 2^{80} operations.

With the secure $r = 224$, $w = 96$, $n = 3 \cdot 2^{13}$ parameters (probability of $2^{-66.7}$ and $2^{82.3}$ binary operations) the speed is a little smaller with only up to 200 Mbits/s. With $r = 288$, $w = 128$, $n = 2^{13}$ (probability below 2^{-80}), the speed should be just above 100 Mbits/s.

5 Conclusion

We have proposed a family of fast and provably secure hash functions. This construction enjoys some interesting features: both the block size of the hash function and the output size are completely scalable, the security depends directly of the output size and can hence be set to any desired level, the number of XORs used by FSB per input bit can be decreased to improve speed. Note that collision resistance can be put aside in order to allow parameters giving an higher output rate.

However, reaching very high output rates requires the use of a large matrix. This can be a limitation when trying to use FSB on memory constrained devices. On classical architectures this will only fix a maximum speed (most probably when the size of the matrix is just below the memory cache size).

Another important point is the presence of weak instances of this hash function: it is clear that the matrix $\mathcal{H}$ can be chosen with bad properties. For instance, the all zero matrix will define a hash function with constant zero output. However, these bad instances only represent a completely negligible proportion of all the matrices and when choosing a matrix at random there is no risk of choosing a weak instance.

References

1. A. Barg. Complexity issues in coding theory. In V. S. Pless and W. C. Huffman, editors, *Handbook of Coding theory*, volume I, chapter 7, pages 649–754. North-Holland, 1998.
2. 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.
3. J. Black, P. Rogaway, and T. Shrimpton. Black box analysis of the block cipher based hash-function constructions from PGV. In *Advances in Cryptology - CRYPTO 2002*, volume 2442 of *LNCS*. Springer-Verlag, 2002.
4. 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.
5. I.B. Damgard. A design principle for hash functions. In Gilles Brassard, editor, *Advances in Cryptology - Crypto' 89*, LNCS, pages 416–426. Springer-Verlag, 1989.
6. J.-B. Fischer and J. Stern. An efficient pseudo-random generator provably as secure as syndrome decoding. In Ueli M. Maurer, editor, *Advances in Cryptology - EUROCRYPT '96*, volume 1070 of *LNCS*, pages 245–255. Springer-Verlag, 1996.
7. P. Guillot. Algorithmes pour le codage à poids constant. Unpublished.
8. Y. Gurevich. Average case completeness. *Journal of Computer and System Sciences*, 42(3):346–398, 1991.
9. 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*, volume 330 of *LNCS*, pages 275–280. Springer-Verlag, 1988.
10. L. Levin. Average case complete problems. *SIAM Journal on Computing*, 15(1):285–286, 1986.
11. 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.
12. A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1996.
13. R. C. Merkle. One way hash functions and DES. In Gilles Brassard, editor, *Advances in Cryptology - Crypto' 89*, LNCS. Springer-Verlag, 1989.
14. National Insitute of Standards and Technology. *FIPS Publication 180: Secure Hash Standard*, 1993.
15. H. Niederreiter. Knapsack-type crytosystems and algebraic coding theory. *Prob. Contr. Inform. Theory*, 15(2):157–166, 1986.
16. R.L. Rivest. The MD4 message digest algorithm. In A.J. Menezes and S.A. Vanstone, editors, *Advances in Cryptology - CRYPTO '90*, LNCS, pages 303–311. Springer-Verlag, 1991.
17. N. Sendrier. On the security of the McEliece public-key cryptosystem. In M. Blaum, P.G. Farrell, and H. van Tilborg, editors, *Information, Coding and Mathematics*, pages 141–163. Kluwer, 2002. Proceedings of Workshop honoring Prof. Bob McEliece on his 60th birthday.
18. J. Stern. A method for finding codewords of small weight. In G. Cohen and J. Wolfmann, editors, *Coding theory and applications*, volume 388 of *LNCS*, pages 106–113. Springer-Verlag, 1989.

A NP-completeness Proofs

The most general problem we want to study concerning syndrome decoding with regular words is:

b-Regular Syndrome Decoding (b-RSD)
Input: w binary matrices $\mathcal{H}_i$ of dimension $r \times n$ and a bit string $\mathcal{S}$ of length r.
Property: there exists a set of b$\times w'$ columns (with $0 < w' \leq w$), 0 or b columns in each $\mathcal{H}_i$, adding to $\mathcal{S}$.

Note that in this problem b is not an input parameter. The fact that for any value of b this problem is NP-complete is much stronger than simply saying that the problem where b is an instance parameter is NP-complete. This also means that there is not one, but an infinity of such problems (one for each value of b). However we consider them as a single problem as the proof is the same for all values of b.

The two following sub-problems are derived from the previous one. They correspond more precisely to the kind of instances that an attacker on the FSB hash function would need to solve.

Regular Syndrome Decoding (RSD)
Input: w matrices $\mathcal{H}_i$ of dimension $r \times n$ and a bit string $\mathcal{S}$ of length r.
Property: there exists a set of w columns, 1 per $\mathcal{H}_i$, adding to $\mathcal{S}$.

2-Regular Null Syndrome Decoding (2-RNSD)
Input: w matrices $\mathcal{H}_i$ of dimension $r \times n$.
Property: there exists a set of $2 \times w'$ columns (with $0 < w' \leq w$), taking 0 or 2 columns in each $\mathcal{H}_i$ adding to 0.

It is easy to see that all of these problems are in NP. To prove that they are NP-complete we will use a reduction similar to the one given by Berlekamp, McEliece and van Tilborg for Syndrome Decoding [2]. We will use the following known NP-complete problem.

Three-Dimensional Matching (3DM)
Input: a subset $U \subseteq T \times T \times T$ where T is a finite set.
Property: there is a set $V \subseteq U$ such that $|V| = |T|$ and no two elements of V agree on any coordinate.

Let's study the following example: let $T = \{1, 2, 3\}$ and $|U| = 6$

$$\begin{aligned} U_1 &= (1,2,2) \\ U_2 &= (2,2,3) \\ U_3 &= (1,3,2) \\ U_4 &= (2,1,3) \\ U_5 &= (3,3,1) \end{aligned}$$

One can see that the set consisting of U_1,U_4 and U_5 verifies the property. However if you remove U_1 from U then no solution exist. In our case it is more convenient to represent an instance of this problem in another way: we associate a $3|T| \times |U|$ binary incidence matrix A to the instance. For the previous example it would give:

	122	223	132	213	331
1	1	0	1	0	0
2	0	1	0	1	0
3	0	0	0	0	1
1	0	0	0	1	0
2	1	1	0	0	0
3	0	0	1	0	1
1	0	0	0	0	1
2	1	0	1	0	0
3	0	1	0	1	0

A solution to the problem will then be a subset of $|T|$ columns adding to the all-1 column. Using this representation, we will now show that any instance of this problem can be reduced to solving an instance of BSD, hence proving that BSD is NP-complete.

Reductions of 3DM to RSD. Given an input $U \subseteq T \times T \times T$ of the 3DM problem, let A be the $3|T| \times |U|$ incidence matrix described above. For i from 1 to $|T|$ we take $\mathcal{H}_i = A$.

If we try to solve the BSD problem on these matrices with $w = |T|$ and $\mathcal{S} = (1, \ldots, 1)$ a solution will exist if and only if we are able to add $w \leq |T|$ columns of A (possibly many times the same one) and obtain a column of 1s. As all the columns of A contain only three 1s, the only way to have $3 \times |T|$ 1s at the end is that during the adding no two columns have a 1 on the same line (each time two columns have a 1 on the same line the final weight decreases by 2). Hence the $|T|$ chosen columns will form a suitable subset V for the 3DM problem.

This means that if we are able to give an answer to this RSD instance, we will be able to answer the 3DM instance we wanted to solve. Thus RSD is NP-complete.

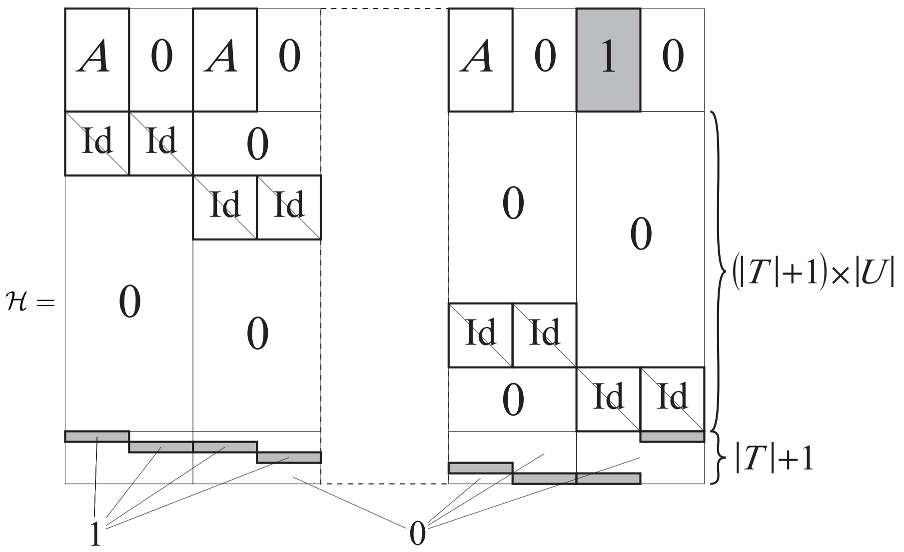
Reduction of 3DM to b-RSD. This proof will be exactly the same as the one above. The input is the same, but this time we build the following matrix:

$$B = \begin{pmatrix} \boxed{A} & & 0 \\ & \boxed{A} & \\ 0 & & \boxed{A} \end{pmatrix}$$

the block matrix with b times A on the diagonal

Once again we take $\mathcal{H}_i = B$ and use $\mathcal{S} = (1, \ldots, 1)$. The same arguments as above apply here and prove that for any given value of b, if we are able to give an answer to this b-RSD instance, we will be able to answer the 3DM instance we wanted to solve. Hence, for any b, b-RSD is NP-complete.

Reduction of 3DM to 2-RNSD. We need to construct a matrix for which solving a 2-RNSD instance is equivalent to solving a given 3DM instance. A difficulty is that, this time, we can't choose $\mathcal{S} = (1, \ldots, 1)$ as this problem is restricted to the case $\mathcal{S} = 0$. For this reason we need to construct a somehow complicated matrix $\mathcal{H}$ which is the concatenation of the matrices $\mathcal{H}_i$ we will use. It is constructed as follows:



This matrix is composed of three parts: the top part with the A matrices, the middle part with pairs of identity $|U| \times |U|$ matrices, and the bottom part with small lines of 1s.

The aim of this construction is to ensure that a solution to 2-RNSD on this matrix (with $w = |T| + 1$) exists if and only if one can add $|T|$ columns of A and a column of 1s to obtain 0. This is then equivalent to having a solution to the 3DM problem.

The top part of the matrix will be the part where the link to 3DM is placed: in the 2-RNSD problem you take 2 columns in some of the block, our aim is to take two columns in *each* block, and each time, one in the A sub-block and one in the 0 sub-block. The middle part ensures that when a solution chooses a column in $\mathcal{H}$ it has to choose the only other column having a 1 on the same line so that the final sum on this line is 0. This means that any time a column is chosen in one of the A sub-blocks, the "same" column is chosen in the 0 sub-block. Hence in the final $2w'$ columns, w' will be taken in the A sub-blocks (or the 1 sub-block) and w' in the 0 sub-blocks. You will then have a sum of w' columns of A or 1 (not necessarily distinct) adding to 0. Finally, the bottom part of the matrix is there to ensure that if $w' > 0$ (as requested in the formulation of the problem) then $w' = w$. Indeed, each time you pick a column in the block number i, the middle part makes you have to pick one in the other half of the block, creating two ones in the final sum. To eliminate these ones the only way is to pick some columns in the blocks $i - 1$ and $i + 1$ and so on, until you pick some columns in all of the w blocks.

As a result, we see that solving an instance of 2-RNSD on $\mathcal{H}$ is equivalent to choosing $|T|$ columns in A (not necessarily different) all adding to 1. As in the previous proof, this concludes the reduction and 2-RNSD is now proven NP-complete.

It is interesting to note that instead of using 3DM we could directly have used RSD for this reduction. You simply replace the A matrices with the w blocks of the RSD instance you need to solve and instead of a matrix of 1s you put a matrix containing columns equal to $\mathcal{S}$. Then the reduction is also possible.

B Modeling Information Set Decoding

Using a classical ISD attack we have seen that the average amount of calculation required to find a solution to an instance of SD is $g(r)/\mathcal{P}_w$. This is true when a complete Gaussian elimination is done for each information set chosen. However, some additional calculations could be done. In this way, each choice of information set could allow to test more words. For instance, in [9], each time an information set is chosen, the validity of this set is tested, but at the same time, partial validity is tested. That is, if there exists a solution with a few 1s among the k positions of the information set, it will also be found by the algorithm. Of course, the more solutions one wants to test for each Gaussian elimination, the more additional calculations he will have to perform.

In a general way, if $\mathcal{K}_1$ and $\mathcal{K}_2$ denote respectively the complexities in space and time of the algorithm performing the additional computations, and M denotes the amount of additional possible solutions explored, we should have:

$$\mathcal{K}_1 \times \mathcal{K}_2 \geq M.$$

Moreover, if $\mathcal{P}_w$ is the probability of finding a solution for one information set, then, when observing M times more solutions at a time, the total probability of success is not greater than $M\mathcal{P}_w$.

Hence, the total time complexity of such an attack would be:

$$\mathcal{K} \geq \frac{g(r) + \mathcal{K}_2}{M\mathcal{P}_w} \geq \frac{g(r)}{M\mathcal{P}_w} + \frac{1}{\mathcal{K}_1\mathcal{P}_w}.$$

When M becomes large the $g(r)/M\mathcal{P}_w$ term becomes negligible (the cost of the Gaussian elimination no longer counts) and we have:

$$\mathcal{K} \geq \frac{1}{\mathcal{K}_1\mathcal{P}_w}.$$

This would mean that, in order to be out of reach of any possible attack, the inverse of the probability $\mathcal{P}_w$ should be at least as large as $\mathcal{K} \times \mathcal{K}_1$. Allowing complexities up to 2^{80} in time and 2^{50} in space we would need $\mathcal{P}_w \leq 2^{-130}$.

However this is only theoretical. In practice there is no known algorithm for which $\mathcal{K}_1 \times \mathcal{K}_2 = M$. Using existing algorithm this would rather be:

$$\mathcal{K}_1 \times \mathcal{K}_2 = M \times Loss \quad \text{and} \quad \mathcal{K} \geq \frac{Loss}{\mathcal{K}_1\mathcal{P}_w},$$

where $Loss$ denotes a function of $\log M$, n, r and w. For this reason, the optimal values for the attack will often only correspond to a small amount of extra calculation for each information set. This will hence save some time on the Gaussian eliminations but will hardly gain anything on the rest. The time complexity $\mathcal{K}$ will always remain larger than $1/\mathcal{P}_w$ and will most probably be even a little above.