A Family of Fast Syndrome Based Cryptographic Hash Functions

Daniel Augot, Matthieu Finiasz, Nicolas Sendrier

A Family of Fast Syndrome Based Cryptographic Hash Functions

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. Recently, some collisions have been exposed for a variety of cryptographic hash functions [19] including some of the most widely used today. Many other hash functions using similar constrcutions can however still be considered secure. Nevertheless, this has drawn attention on the need for new hash function designs.
In this article is presented a familly of secure hash functions, whose security is directly related to the syndrome decoding problem from the theory of error-correcting codes.
Taking into account the analysis by Coron and Joux [4] based on Wagner's generalized birthday algorithm [18] we study the asymptotical security of our functions. We demonstrate that this attack is always exponential in terms of the length of the hash value.
We also study the work-factor of this attack, along with other attacks from coding theory, for non asymptotic range, i.e. for practical values. Accordingly, we propose a few sets of parameters giving a good security and either a faster hashing or a shorter desciption for the function.

Key Words: cryptographic hash functions, provable security, syndrome decoding, NP-completeness, Wagner's generalized birthday problem.

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 [6,12]. Classical compression functions are very fast [13,15] but, in general, cannot be proven secure. However, provable security may be achieved with compression functions designed following public key principles, at the cost of being less efficient. This has been done for instance by Damgård in [6], where he designed a hash function based on the Knapsack problem. Accordingly, this function has been broken by Granboulan and Joux [8], using lattice reduction algorithms. The present paper contributes to the hash function family by designing functions based on the syndrome decoding problem, which is immune to lattice reduction based attacks.

Unlike most other public key cryptosystems, the encryption function of the McEliece cryptosystem [10] (or of Niederreiter's version [14]) is nearly as fast

as a symmetric cipher. Using this function with a random matrix instead of the usual parity check matrix of a Goppa code, a provably secure one-way function has been constructed in [1]: since there is no trapdoor, its security can be readily related to the difficulty of syndrome decoding. For instance, there is no polynomial time algorithm to decode a random code, thus there is no polynomial time algorithm to invert the compression function and/or find a collision.

However, for the practical parameters which have been proposed in [1], there is an efficient attack with a cost as low as 2^{43} (or 2^{62} depending on the set of parameters), as demonstrated by Coron and Joux [4], using Wagner's method for the generalized birthday problem [18].

The purpose of this paper is to propose updated parameters for the hash function family presented in [1]. We do not only extend the parameters to be out of reach of Coron-Joux attack, but we also thoroughly study the asymptotical behavior of their attack. We shall establish that this attack is exponential, such that the design for the hash function is sound.

The paper is organized as follows. In Section 2, we introduce the *Fast Syndrome Based* (FSB) compression function, derived from a hard problem similar to syndrome decoding. In Section 3 we show that the security of FSB is reduced to the average case difficulty of two new NP-complete problems. Then, in Section 4, we show how the best known decoding techniques, and the new method based on Wagner's ideas, can be adapted to the cryptanalysis of our functions. From that we can evaluate the practical security and the scalability of the system. In Section 5, we propose some choices of parameters and, eventually, we compare the obtained functions with other existing constructions. For clarity of the presentation, NP-completeness proofs are postponed in the appendix.

2 The Hash Function

We present here what is called the *Fast Syndrome Based* (FSB) hash function in [1].

2.1 General Construction of Hash Functions

We follow Merkle's and Damgård's design principle of hash functions [6,12]: iterating a compression function (here denoted $\mathcal{F}$), which takes as input s bits and returns r bits (with $s > r$). The resulting function is then chained to operate on strings of arbitrary length (see Fig. 1). The validity of such a design has been established [6,12], and its security is proven not worse than the security of the compression function. Therefore we will only concentrate on the security of the latter.

2.2 Description of the Compression Function

The core of the compression function is a random binary matrix $\mathcal{H}$ of size $r \times n$. The parameters for the hash function are:

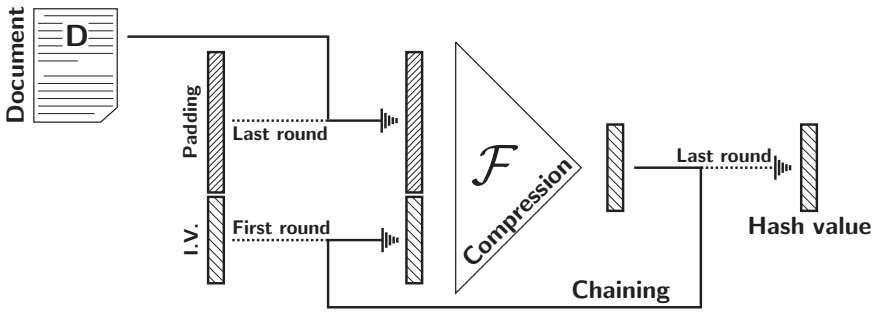

Fig. 1. A standard hash function construction.

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

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 $\left[\!\left[(i-1)\frac{n}{w}; i\frac{n}{w}\right]\!\right]_{i=1..w}$. We call a block such an interval.*

To code a regular word of length n and Hamming weight w, $s = w\log_2(\frac{n}{w})$ bits are needed. This is the size in bits of the input of the compression function $\mathcal{F}$. When practical parameters will be chosen, it will be made in such a manner that round figures for $\log_2(\frac{n}{w})$ are obtained. That is $\frac{n}{w}$ has to be a power of 2 and ideally, for software efficiency, $\log_2(\frac{n}{w})$ too.

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

FSB compression function

Input: s bits of data

1. split the s input bits in w parts $s_1, \ldots, s_w$ of $\log_2(\frac{n}{w})$ 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

The speed of $\mathcal{F}$ is directly related to the number of XORs required at each round: one needs to XOR w columns of r bits, that is wr binary XORs. The number of bits read in the document at each round is $w\log_2\frac{n}{w} - r$ (input size minus chaining size). Hence, the average number of binary XORs required for each document input bit is:

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

This value will be the right measure for the global speed of the FSB hash function.

3 Theoretical Security

As stated in [11,16], 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 the hash function is collision free and resistant to inversion. We show that the inversion and collision finding are related to two problems very close to syndrome decoding, which is a hard problem [3]. We describe them here and show (in appendix) that they are also NP-complete.

3.1 Two New NP-complete Problems

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

Definition 2. *A* 2-regular word *is a word of weight less than or equal to* $2w$*, which contains either* 0 *or* 2 *non zero positions in each block. It is the sum of two regular words.*

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

Thus solving **2-Regular Null Syndrome Decoding** requires to find a non null 2-regular word of weight less than or equal to $2w$.

3.2 Security Reduction

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

Let us be given an algorithm for the *inversion* of the compression function, which, given a hash value $\mathcal{S}$, finds an inverse m such that $\mathcal{F}(m) = \mathcal{S}$, in time T with probability p of success. Then it is a tautology that this algorithm is able to solve any instance of the problem **Regular Syndrome Decoding**, in the same time and with the same probability of success. Similarly an algorithm able to find a collision gives in fact two different regular words c_1 and c_2 of weight w such $Hc_1^t = Hc_2^t$. Then $c = c_1 \oplus c_2$ is a non null 2-regular word and has a null syndrome. So c is directly a solution for **2-Regular Null Syndrome Decoding**.

These reductions to NP-complete problems only prove worst case difficulty. However, following Gurevich and Levin [7,9] discussion for syndrome decoding, we believe that both these NP-complete problems are difficult in average.

3.3 Average case security

– instance faibles ?
– toutes les instances sont utilisees

4 Practical Security

We recall the possible practical attacks on the compression function $\mathcal{F}$, and study the minimal work-factors required to perform these attacks. There are two kinds of algorithms: Information Set Decoding and Wagner's Generalized Birthday Paradox. We will survey the results on Information Set Decoding algorithm, which has been studied in [1]. As for Wagner's Generalized Birthday Paradox [18], we will give an extended analysis: first we slightly generalize Wagner's algorithm, then we describe how its complexity is exponential when the length of the hash value goes to infinity.

4.1 Information Set Decoding

This problem of decoding a random code has been extensively studied and many algorithms devoted to this task have been developed (see [2] for a survey). All these algorithms are exponential. Still, as stated by Sendrier [17], the most efficient attacks all seem to be derived from *Information Set Decoding* (ISD).

Definition 3. *An* information set *is a set of $k = n - r$ (the dimension of the code) positions among the n positions of the support.*

Definition 4. *Let $((\mathcal{H}_i)_{1\leq i\leq w}, w, \mathcal{S})$ be an instance of RSD. An information set will be called* valid *with respect to the instance if there exists a solution to this problem which has no 1s among the k positions of the set.*

The ISD algorithm consists in picking information sets at random, until a valid one is found. Checking whether the information set is valid or not is done in polynomial time, so the exponential nature of the algorithm originates from the exponentially small probability of finding a valid information set.

Using simple counting arguments we can calculate the probabilities of picking a valid information set either when trying to inverse $\mathcal{F}$ or to find collisions. We obtain:

$$\mathcal{P}_{\text{inv}} = \frac{\binom{r}{w}^w}{2^r},$$

$$\mathcal{P}_{\text{col}} = \frac{1}{2^r}\left[\binom{\frac{r}{w}}{2}+1\right]^w \leq 2^{\frac{-r}{3.3}}.$$

We can hence lower bound the complexity of finding a collision using this attack by $\mathcal{O}\left(2^{\frac{r}{3.3}}\right)$. However this attack is mainly efficient when the number of solutions is close to 1. Here the number of collisions for a given matrix is huge and the attack we present next can hence do much better.

4.2 Wagner's Generalized Birthday Problem

We now describe the attack from Coron and Joux [4], which relies on the Generalized Birthday Problem introduced by Wagner [18], who established:

Theorem 1. *Let L_1, L_2,...,L_{2^a} be 2^a lists of r bits strings, each list being of size $L = 2^{\frac{r}{a+1}}$. Then a solution to the equation $x_1 \oplus x_2 \oplus \ldots \oplus x_{2^a} = 0$, with $x_i \in L_i$, can be found in time $O(2^a 2^{\frac{r}{a+1}})$.*

Let us recall the algorithm. At first step there are 2^a lists of size L. Then the lists are merged pair by pair to create a new list: for instance, the merge $L_1 \bowtie L_2$ of the lists L_1 and L_2 is made from the sum of the elements x_1, x_2 of L_1, L_2 such that $x_1 \oplus x_2$ is zero on the first $\frac{r}{a+1}$ bits. One readily checks that the size $L_1 \bowtie L_2$ is still $2^{\frac{r}{a+1}}$ in average. Using sorting algorithms, this merge operation can be done in time $O(r2^{\frac{r}{a+1}})$ (one can gain the r factor, and thus obtain Wagner's complexity, using hash tables, however this will require larger memory space). Once the 2^{a-1} new lists are obtained, one proceeds recursively, until there are only two remaining lists (See Fig. 2). Then a collision is found between these two lists using the classical birthday paradox. Since there are 2^a merge operations, the total complexity is $O(r2^a 2^{\frac{r}{a+1}})$.

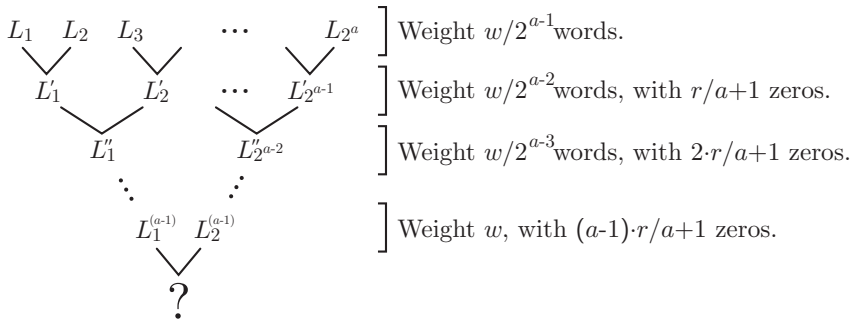

Fig. 2. Application of Wagner's algorithm to collision search. All the lists remain of constant size $L = 2^{\frac{r}{a+1}}$. In average, there remains a single solution at the end of the algorithm.

This algorithm translates into an attack for collisions as follows. Each list L_i is associated to $\frac{w}{2^a}$ blocks of the matrix, and contains the syndromes of 2-regular words, of weight less than or equal to $\frac{2w}{2^a}$, defined over these blocks. Finding a solution $Hx_1 \oplus Hx_2 \oplus \ldots \oplus Hx_{2^a} = 0$ gives a collision.

The adversary will try to optimize a in order to get the lowest complexity. But the auxiliary a is subject to the following constraint: each list (of size $L = 2^{\frac{r}{a+1}}$) can not contain more than the number of words of weight $\frac{2w}{2^a}$ (or lower), which are part of a 2-regular word, with a given set of blocks. Since in each block there

are $\binom{\frac{n}{w}}{2} + 1$ words of weight 2 or 0, this gives:

$$\frac{r}{a+1} \leq \frac{w}{2^a} \log_2 \left[\binom{\frac{n}{w}}{2} + 1 \right],$$

or equivalently:

$$\frac{2^a}{a+1} \leq \frac{w}{r} \log_2 \left[\binom{\frac{n}{w}}{2} + 1 \right]. \quad (1)$$

This shows that a can not grow as desired by the adversary. In the case of inversion search, the constraint is that the size of the list must be lower than the number of regular words of weight $\frac{w}{2^a}$, with 1's in some $\frac{w}{2^a}$ blocks. This gives, similarly to the collision case:

$$\frac{2^a}{a+1} \leq \frac{w}{r} \log_2(\frac{n}{w}). \quad (2)$$

4.3 Extension of Wagner's Algorithm to Non-Fitting Cases

In general, it may be the case that size $L = 2^\ell$ of the lists to be dealt with is not exactly $2^{\frac{r}{a+1}}$.

We first deal with the case when $\ell < \frac{r}{a+1}$. In that case, we apply Wagner's method, with the constraint of zeroing ℓ bits of the partial sums (instead of $\frac{r}{a+1}$) during each merge operation, hence keeping lists of constant size. So, the two remaining lists, at the end of the recursion, will only have $(a-1)\ell$ bits equal to zero. Then the probability to have a collision between these two lists is $\frac{2^{(a+1)\ell}}{2^r}$. If the algorithm fails and there is no collision, then the whole process is started from the beginning, choosing another set of bits to be set to zero. Since the complexity of building the two final lists is $O(\ell 2^a 2^\ell)$, the total cost is $O(\ell 2^{r+a-a\ell})$. Again, for this complexity to hold, the size of the list must be smaller than $2^{\frac{r}{a+1}}$. The contrary would correspond to having more than one collision in the end, which won't help improving the complexity.

Secondly, we deal with the case when $\ell > \frac{r}{a+1}$. Then the strategy is to prepare each list using a precomputation step, by zeroing α bits in a each list. The size of the lists is then in average $2^{\ell-\alpha}$. In the context of the hash function, this precomputation step is performed by using two sublist and merging them using the birthday paradox. Then Wagner's algorithm is applied to set to zero the $r' = r - \alpha$ remaining bits. Ideally α is chosen to fit to the new parameters of Wagner's algorithm: we must have $\ell - \alpha = \frac{r'}{a+1}$. Solving these two equations gives:

$$\alpha = \frac{\ell(a+1) - r}{a} \text{ and } r' = \frac{a+1}{a}(r - \ell),$$

and the total cost of Wagner's algorithm is $O(r' 2^a 2^{\frac{r'}{a+1}})$. Note that preparing all the lists, with the birthday paradox, costs $O(2^a 2^{\frac{\ell}{2}})$, so there might be a concern that this step becomes preponderant. Solving the inequalities tells us that this is only the case when $\ell > \frac{2r}{a+2}$, which means that we fall in the range where $a+1$ could have been used for the attack (see Equations (1) and (2)).

4.4 Some Security Curves

The curves on Fig. 3 show how the different attacks described in this Section behave when applied to concrete parameters. It is clear that the attack based on Wagner's Generalized Birthday Paradox gives far better results than Information Set Decoding techniques applied to regular words.

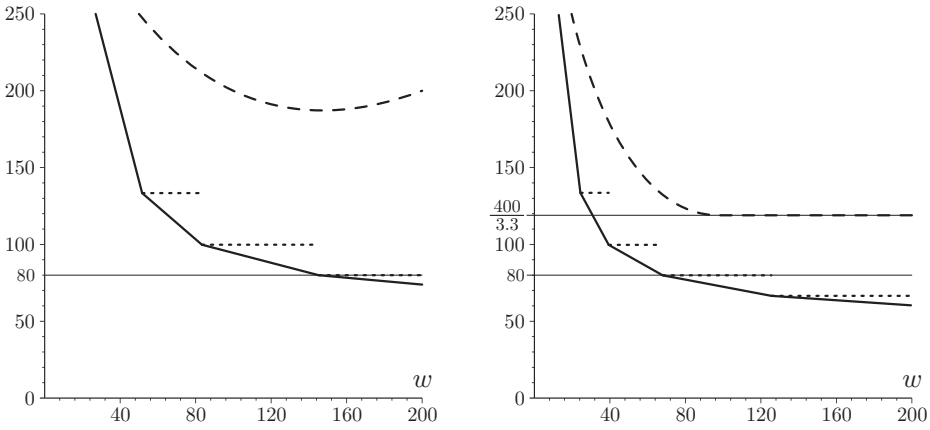

Fig. 3. Comparison of the costs of the different attacks as a function of w when $r = 400$ and $n = 2^{16}$. On the left when applied to inversion and on the right to collision search. The vertical scale corresponds to the logarithm of the work-factor required to perform Information Set Decoding (dashed line), Wagner's Generalized Birthday Paradox (dotted line) or Extended Wagner Paradox (plain line).

It is also important to note that, for a same security level, the scope of available parameters is much wider if only looking for a one-way function (no collision resistance). For instance, with $r = 400$ and $n = 2^{16}$, for a security of 2^{80}operations, w could be chosen anywhere in the interval $[\![0; 145]\!]$ instead of $[\![0; 67]\!]$.

4.5 Asymptotical Behavior

We want to prove that, even though a can vary depending on the parameters, when r goes to infinity a can not grow fast enough to make Wagner's attack sub-exponential. The only condition on a is:

$$\frac{2^a}{a+1} \leq \frac{w}{r} \log_2 \left(\frac{n}{w}\right).$$

If we consider n and w as polynomial in r (noted $\mathcal{P}oly\,(r)$) we then have:

$$\frac{2^a}{a+1} \leq \mathcal{P}oly\,(r) \log_2 \left(\mathcal{P}oly\,(r)\right).$$

From this we deduce $a = \mathcal{P}oly(\log_2 r)$. Asymptotically, the best attack having a cost of $O(r2^a 2^{\frac{r}{a+1}})$ remains thus exponential in r. Moreover, it should be possible to find parameters which scale well when trying to increase the security level.

The simplest solution to achieve so is to scale the parameters linearly with r. For instance, suppose we have two constants ω and ν such that $w = \omega \times r$ and $n = \nu \times r$. We get:

$$\frac{2^a}{a+1} \leq \omega \log_2\left(\frac{\nu}{\omega}\right) \quad \text{so} \quad a \simeq \log_2 \omega \log_2 \log_2 \frac{\nu}{\omega} = \kappa \quad \text{a constant.}$$

This means that the best a an attacker can use will remain constant whatever the value of r. Asymptotically this construction will scale with:

- exponential security: $2^{\frac{r}{a+1}} = 2^{\mathcal{O}(r)}$,
- constant block size: $\log_2 \frac{n}{w} = \log_2 \frac{\nu}{\omega} =$ constant ($\frac{n}{w}$ remains a power of 2),
- linear hash cost: $\mathcal{N}_{XOR} = \frac{r^2\omega}{r(\omega \log \frac{\nu}{\omega} - 1)} = \mathcal{O}(r)$,
- quadratic matrix size: $r \times n = r^2\nu = \mathcal{O}\left(r^2\right)$.

Using this method it is possible to convert any set of efficient parameters to another efficient set giving any other required security, by simply scaling r.

5 Proposed Parameters

Usually hash functions have a security of $2^{\frac{r}{2}}$ against collisions, that is, the best attack is based on the classical birthday paradox. In our case this would correspond to a being equal to 1 at most. However, if this is the case, $\mathcal{F}$ will necessarily have an input shorter than its output and this is incompatible with the chaining method. If we want to have an input size larger than the output size, then the attacker will always have the possibility to choose at least $a = 3$ (when looking for collisions). If we suppose our parameters place us exactly at the point where an attacker can use $a = 3$, we have:

$$r = \frac{(3+1)w}{2^3} \log_2 \left[\binom{\frac{n}{w}}{2} + 1\right] \geq \frac{w}{2} \log_2 \left[\left(\frac{n}{w}\right)^2 \times \frac{1}{2}\right] = w \log_2\left(\frac{n}{w}\right) - \frac{w}{2}.$$

If this is the case we will then have:

$$\mathcal{N}_{XOR} \geq \frac{rw}{\frac{w}{2}} = 2r.$$

For a security of 2^{80} and with $a = 3$ we would need at least $r = 320$ and hence at least 640 binary XORs per input document bit. This is not so huge in practice but it would still give a relatively slow hash rate. For instance, it is just above 10 Mbits/s, using a vanilla C implementation on a Pentium 4.

If we instead choose to limit the attacker to $a = 4$ we will have much more freedom in the parameter choice. First of all, we get:

$$\mathcal{N}_{XOR} = \frac{rw}{\left(1 - \frac{5}{8}\right) \log_2\left(\frac{n}{w}\right) + \frac{5}{16} w}.$$

Changing the values of n and w (which are still linked by the constraint $a = 4$) will let us change the hash cost. However, as we see on Fig. 4, the lowest values for $\mathcal{N}_{XOR}$ also correspond to the largest values of n and so, to larger matrix sizes. Table 1 collects a list of parameter sets all corresponding to $a = 4$ and $r = 400$, that is, a security of 2^{80}. In fact, practical security will be a little higher as we have neglected a $r2^a$ factor. The security should hence rather be around 2^{92} operations, and an exact security of 2^{80} would be achieved with $r = 340$. However an attacker with huge memory can get rid of the r factor. We will hence stick to the $2^{\frac{r}{a+1}}$ approximation of the security.

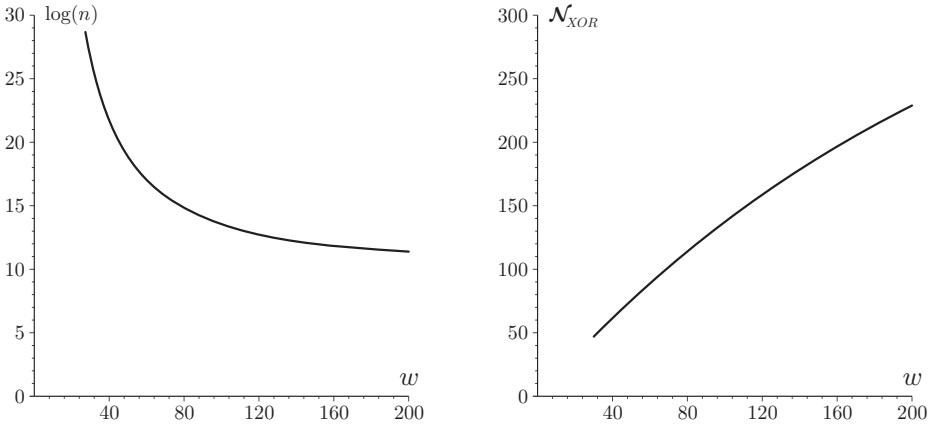

Fig. 4. Evolution of $\log_2 n$ (on the left) and $\mathcal{N}_{XOR}$ (on the right) as a function of w for $r = 400$ when always staying as close as possible to the point $a = 4$.

If we choose to use the set of parameters where $\log_2 \frac{n}{w} = 8$, which is very convenient for software implementation, we will have at the same time a good efficiency (7 times faster than when trying to force $a = 3$) and a reasonable matrix size. As we have seen in Section 4.5, we can then scale these parameters. We have $\omega = \frac{85}{400} = 0.2125$ and $\nu = 256 \times \omega = 54.4$. If we now want to hash with a security just a little higher like 2^{96} we simply need to use $r = 96 \times (a+1) = 480$ and so $w = 480 \times \omega = 102$ and $n = 26112$.

We propose three sets of parameters giving a security of 2^{80} against collision search for different output hash sizes. Each of these sets can be scaled linearly to obtain a better security.

- **Short Hash:** $r = 320$, $w = 42$ and $\log_2 \frac{n}{w} = 8$. This solution has a hash size of 320 bits only, but is quite slow with a throughput around 10 Mbits/s.
- **Fast Hash:** $r = 480$, $w = 170$ and $\log_2 \frac{n}{w} = 8$. This solution will be very fast (around 90 Mbits/s) with still a reasonable matrix size (16 Mbits).
- **Intermediate:** $r = 400$, $w = 85$ and $\log_2 \frac{n}{w} = 8$. This is in our opinion the best compromise, with reasonable hash length and matrix size and still a good efficiency (around 70 Mbits/s).

$\log_2\left(\frac{n}{w}\right)$	w	n	$\mathcal{N}_{XOR}$	matrix size
16	41	2 686 976	64.0	$\sim$ 1 Gbit
15	44	1 441 792	67.7	550 Mbits
14	47	770 048	72.9	293 Mbits
13	51	417 792	77.6	159 Mbits
12	55	225 280	84.6	86 Mbits
11	60	122 880	92.3	47 Mbits
10	67	68 608	99.3	26 Mbits
9	75	38 400	109.1	15 Mbits
8	85	21 760	121.4	8.3 Mbits
7	98	12 544	137.1	4.8 Mbits
6	116	7 424	156.8	2.8 Mbits
5	142	4 544	183.2	1.7 Mbits
4	185	2 960	217.6	1.1 Mbits

Table 1. Possible parameters for $r = 400$ and $a = 4$.

If looking only for a one-way function (no collision resistance) then we have the choice to either be faster, or have a smaller output.

- **Short One-Way:** $r = 240$, $w = 40$ and $\log_2 \frac{n}{w} = 8$. This solution has an output of only 240 bits and should work at around 70 Mbits/s.
- **Fast One-Way:** $r = 480$, $w = 160$ and $\log_2 \frac{n}{w} = 16$. This solution uses a very large matrix (4 Gbits) but should have a throughput above 200 Mbits/s.

6 Comparison with existing hash functions

As stated at the begining of Section 5, from a practical security point of view, our functions are somehow weaker than other existing functions: we will never be able to reach a security of $\mathcal{O}\left(2^{\frac{r}{2}}\right)$ against collision search. Accordingly, the output size of our functions will always have to be above 320 bits.

The description of one of our function will also always be much larger than that of other functions: the matrix should be included in the description and is always quite large when looking for fast hashing. However, as long as one uses parameters for which the matrix isn't of many Gigabits this shouldn't cause any problem.

From a speed point of view, our functions also seem much slower than existing functions. For instance, as seen in Table 2, using Wei Dai's crypto++ library, other hash functions are much faster than the 90 Mbits/s of **Fast Hash**. One should however take into account the fact that these 90 Mbits/s are obtained using a very basic C implementation, taking no advantage of the extended pentium operations (MMX, SSE...).

The operations in FSB are however very simple: the only costly operations are binary XORs. Hence what will slow the process will mainly be memory access problems as the matrix $\mathcal{H}$ has no chance to fit in the machine's CPU cache. Optimizing these accesses should hence speed up conciderably the process: accessing

Algorithm	Mbits/s
MD5	1730
RIPEMD-160	420
SHA-1	544
SHA-512	90

Table 2. Throughputs of some other hash functions, using the crypto++ library [5].

128 bits at a time with SSE instructions instead of 32 as in our implementation should nearly multiply the throuput by 4. Our functions, should hence remain slower than SHA, but the gap isn't as great as it appears.

Moreover, depending of the use made of the hash function, the flexibility in the parameter choice of FSB can compensate this gap. Imagine hashing for a 1024 bits RSA signature: you need to output a 1024 bits hash and at the same time do not require a security higher than 2^{80} as it would be higher than that of the RSA part of the signature. For such application, with $r = 1024$, one could use one of the following parameter sets:

a	security	$\log_2 \frac{n}{w}$	w	n	$\mathcal{N}_{XOR}$	matrix size
11	2^{85}	8	11655	2 983 680	129	3 Gbits
8	2^{113}	8	1942	497 152	137	485 Mbits

Still using our basic implementation this would yield througputs around 70Mbits/s, which is not bad for a 1024 bits hash function. With FSB the througput will depend more on the required security level than on the output hash size.

7 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 is truly exponential, it can hence be set to any desired level; the number of XORs used by FSB per input bit can be decreased to improve speed.

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.

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. D. Augot, M. Finiasz, and N. Sendrier. A fast provably secure cryptographic hash function. Cryptology ePrint Archive, 2003. Available at: http://eprint.iacr.org/2003/230/.
2. 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.
3. 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.
4. J.-S. Coron and A. Joux. Cryptanalysis of a provably secure cryptographic hash function. Cryptology ePrint Archive, 2004. http://eprint.iacr.org/2004/013/.
5. Wei Dai. Crypto++ library. http://www.eskimo.com/∼weidai/.
6. I.B. Damgård. A design principle for hash functions. In Gilles Brassard, editor, *CRYPTO 89*, volume 435 of *Lecture Notes in Computer Science*, pages 416–426. Springer-Verlag, 1989.
7. Y. Gurevich. Average case completeness. *Journal of Computer and System Sciences*, 42(3):346–398, 1991.
8. A. Joux and L. Granboulan. A practical attack against knapsack based hash functions. In Alfredo De Santis, editor, *Eurocrypt 04*, Lecture Notes in Computer Science, pages 58–66. Springer-Verlag, 1994.
9. L. Levin. Average case complete problems. *SIAM Journal on Computing*, 15(1):285–286, 1986.
10. 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.
11. A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1996.
12. R. C. Merkle. One way hash functions and DES. In Gilles Brassard, editor, *Advances in Cryptology - Crypto' 89*, Lecture Notes in Computer Science. Springer-Verlag, 1989.
13. National Insitute of Standards and Technology. *FIPS Publication 180: Secure Hash Standard*, 1993.
14. H. Niederreiter. Knapsack-type crytosystems and algebraic coding theory. *Prob. Contr. Inform. Theory*, 15(2):157–166, 1986.
15. R. L. Rivest. The MD4 message digest algorithm. In A.J. Menezes and S.A. Vanstone, editors, *Advances in Cryptology - CRYPTO '90*, Lecture Notes in Computer Science, pages 303–311. Springer-Verlag, 1991.
16. P. Rogaway and T. Shrimpton. Cryptographic hash-function basics: definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In Bimal Roy and Willi Meier, editors, *Fast Software Encryption 2004*, volume 3017 of *Lecture Notes in Computer Science*, 2004.
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. D. Wagner. A generalized birthday problem. In M. Yung, editor, *CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 288–304. Springer-Verlag, 2002.

19. X. Wang, D. Feng, X. Lai, and H. Yu. Collisions for hash functions md4, md5, haval-128 and ripemd. Cryptology ePrint Archive, 2004. http://eprint.iacr.org/2004/199/.

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 $\mathcal{H}_i$, summing 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 $\mathcal{H}_i$, summing 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 $\mathcal{H}_i$ summing 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 [3]. 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 $|U| = 5$

$$U_1 = (1, 2, 2)$$
$$U_2 = (2, 2, 3)$$
$$U_3 = (1, 3, 2)$$
$$U_4 = (2, 1, 3)$$
$$U_5 = (3, 3, 1)$$

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 the matrix shown in Table 3.

A solution to the problem will then be a subset of $|T|$ columns suming to the all-1 column. Using this representation, we will now show that any instance of

	U_1	U_2	U_3	U_4	U_5
	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

Table 3. Incidence matrix corresponding to an instance of 3DM.

this problem can be reduced to solving an instance of RSD, hence proving that RSD 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 RSD 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:

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

the block matrix with b times A on the diagonal

Now 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

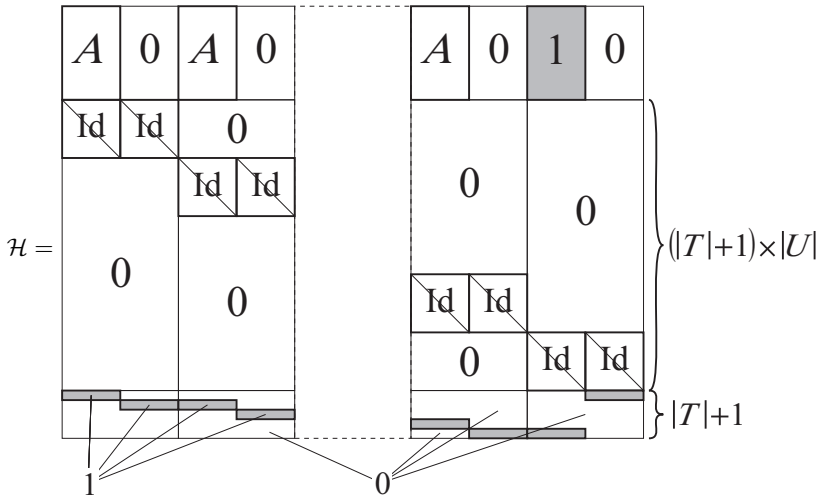

Fig. 5. The matrix used to reduce 3DM to 2-RNSD.

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 shown in Fig. 5.

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 set of w' columns of A or 1 (not necessarily distinct) summing 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 summing 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.