Improved Fast Syndrome Based Cryptographic Hash Functions

Matthieu Finiasz[1,3], Philippe Gaborit[2], and Nicolas Sendrier[3]

[1] ENSTA
[2] Xlim
[3] INRIA

Abstract. At Mycrypt 2005, Augot, Finiasz and Sendrier presented a provably collision resistant family of hash functions [1]. We propose here to improve this construction in two ways: we add a final compression transform so as to achieve a security level equal to half the output length and we use a random quasi-cyclic matrix instead of a generic random matrix in order to get a shorter description for the hash function. Having a shorter description helps in multiple aspects: first the matrix can fit in the cache of a standard CPU, thus greatly improving the overall speed of the construction, second it allows new applications, for example in memory constrained environments.

Keywords: hash function, provable security, quasi-cyclic codes

1 Introduction

In the world of hash functions, provable security is something which is quite singular. Combining it with efficiency makes it even more singular. Recently, however, a few example of such constructions have appeared, relying on hard problems from different domains of cryptography: the Fast Syndrome Based hash function [1] relying on problems from coding theory, VSH [6] relying on a new problem of number theory, or LASH [2] relying on lattice problems. Though efficient, these constructions are still far from reaching throughputs similar to those of MD5 [17] or SHA-1 [15]. However, considering the recent attacks on a wide variety of hash functions, provable security has become an important, if not essential, aspect of hashing, and a slower hash rate is not too much of an issue for many applications. We therefore find it very important to closely study these constructions, propose new ones, or improve them when possible.

In this article we focus on the Fast Syndrome Based hash function presented by Augot, Finiasz, and Sendrier [1], and modify it so as to remove some of its drawbacks while at the same time notably improving its speed. We start by recalling how this construction works in Section 2. Then, as for LASH, FSB manipulates a large inner state for hashing and directly outputting it as the final hash makes collision search easier than half the size of the hash. In Section 3 we discuss how to add a final transform in order to reduce the hash size, without reducing the security. After this, the main drawback of the FSB construction is addressed: it requires to use a large random matrix which we propose to reduce as much as possible. First in Section 4 we propose to use quasi-cyclic codes so as to store a single row of the matrix instead of the whole matrix. Then in Section 5 we propose some other techniques to reduce the size even more. We finally present some implementation results in Section 6 and propose some parameters allowing to hash only two times slower than SHA-256 with the same security level. Finally, a small note in Appendix A shows how to slightly improve inversion attacks on the original construction.

2 The Original Augot-Finiasz-Sendrier Construction

The Augot-Finiasz-Sendrier construction [1] is based on the Merkle-Damgård hash function design [7,13] and thus consists in iterating a compression function, denoted $\mathcal{F}$. The function $\mathcal{F}$ takes an input of s bits and outputs r bits (with $r < s$ in order for it to compress), and uses a random binary matrix $\mathcal{H}$ of size $r \times n$. This compression function simply consists in XORing w columns of $\mathcal{H}$ (among n possible) to obtain an r bit output. The s input bits thus have to be converted into a word of length n and Hamming weight w, the non-zero positions of this word correspond to the columns of $\mathcal{H}$ to XOR together. Many solutions exist to map a sequence of bits to a word of given length and weight, from the exact solution mapping $s = \log_2 \binom{n}{w}$ bits which is, unfortunately, very slow, to more efficient, but less optimal, solutions like using *regular words* as proposed in [1].

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.*

The hash algorithm can then be written as:

The compression function $\mathcal{F}$

Input: s bits of data

1. split the s input bits into w parts $s_1, \ldots, s_w$ of $\log_2(\frac{n}{w})$ bits each,
2. convert each s_i to an integer between $i \times \frac{n}{w} + 1$ and $(i+1) \times \frac{n}{w}$,
3. choose the corresponding columns in $\mathcal{H}$,
4. XOR the w chosen columns to obtain a binary string of length r.

Output: r bits of hash

2.1 Security of the Original Construction

If one considers the matrix $\mathcal{H}$ as the parity check matrix of a linear code, then what the compression function $\mathcal{F}$ does, is simply compute the syndrome of a word of low weight w. Inverting the compression function $\mathcal{F}$ thus consists in solving an instance of the Syndrome Decoding (SD) problem which is known to be NP-complete [3]. Similarly, finding a collision on the function will require to find a non-null word of even weight smaller than $2w$ with a null syndrome. This also corresponds to an instance of the SD problem. However, the use of regular words is not usual, and one could believe that such instances of SD might be easier to solve than generic instances. For this reason, Augot, Finiasz, and Sendrier provide a proof that the exact problems which have to be solved to find a collision/invert the function $\mathcal{F}$ are also NP-complete.

Still, from a practical point of view, the best attacks on the construction are not too inefficient. Depending on the parameters n, r, and w, the best attack will either be a decoding attack (decoding a random binary code) using the Canteaut-Chabaud algorithm [4], or Wagner's generalized birthday technique [19] which is particularly efficient for solving decoding problems which have a large number of solutions (as it is the case here for inversion or collision search). In most cases, Wagner's technique will be the most efficient: it consists in building 2^a lists of r bit strings, each string being the XOR of $\frac{2w}{2^a}$ (or $\frac{w}{2^a}$ for inversion) columns of $\mathcal{H}$, and then, merge these lists pairwise to end up with, in average, a single combination of $2w$ (or w for inversion) columns of $\mathcal{H}$ XORing to 0 (or any given target). The parameter a is essential as it directly determines the complexity of the attack, but a cannot be chosen freely: the 2^a

initial lists must contain enough elements for the attack to succeed. In the end, the authors obtain that for any a such that:

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

a collision can be found with complexity $\mathcal{O}(2^{\frac{r}{a+1}})$. Similarly, inversion is possible using Wagner's technique with any parameter a verifying:

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

and also has a cost $\mathcal{O}(2^{\frac{r}{a+1}})$ (but, as for given parameters w, n, and r, the maximum value of a will be smaller than for collision search, the complexity of inversion is higher in practice).

2.2 Parameter Selection

When selecting parameters for this construction the two main issues are speed and security. As the compression function $\mathcal{F}$ only consists of XORs, speed can be measured by the number of binary XORs required to hash one bit of message. This is given by the formula:

$$\mathcal{N}_{XOR} = \frac{r \cdot w}{w \log_2 \frac{n}{w} - r}. \tag{3}$$

As explained in [1], efficient parameters (that is, parameters for which $\mathcal{N}_{XOR}$ is small) always allow to select $a = 4$ in equation (1). What this means is that for a given security level of 2^λ binary operations, the output of the compression function must be of at least 5λ bits. This has two consequences: first the usual requirement that the security of a hash function must be half its output size is not respected, second, the value of r being large, the matrix $\mathcal{H}$ will necessarily be very large too.

Looking a little more into the details, it appears that parameters allowing larger values of a in equation (1) (while keeping a constant security level) will also allow smaller values of $\mathcal{N}_{XOR}$, and that larger values of n can also improve the hash speed. This means that the size of $\mathcal{H}$ is a limiting factor for the efficiency of $\mathcal{F}$. Indeed, the *intermediate* parameters proposed in [1], with a matrix of 8.3 Mbits, could be made twice as fast with a matrix of 1 Gbits. A size of a few Mbits is already too much for constrained platforms like RFID or even smartphones, but a size of 1 Gbit is not fit for *any* application.

2.3 Conclusion about the Original Construction

Augot, Finiasz, and Sendrier have proposed a construction for fast and provably collision resistant hash functions. This is a great achievement as only very few other constructions offer similar properties. However their construction suffers from two major limitations: the output size of the hash function is larger than twice the security goal and the size of the matrix $\mathcal{H}$ limits both the speed and the possible applications of this construction. To our knowledge, Wagner's generalized birthday attack is the best published against this construction: in the rest of this article we will take it as the reference attack to assess the security of the parameters we propose.

3 Reducing the Output Size

When it comes to hash functions, the cost of collision search is expected to be exactly half the output size of the hash function. That is to say: the best collision search attack should be the standard birthday technique. For this reason, when using a compression function for which collision search is easier, designers usually add a final transform in order to reduce the hash size to what is expected: twice the cost of collision search. In [1], the proposed parameters correspond to a security level of 2^{80} operations, which means that a final compression function g should be added to reduce the hash from 400 bits (or 320, or 480, depending on the parameter set) to 160 bits. How can this be done without loosing the security proof?

3.1 Conditions on the Final Compression Function

Let us denote by f the complete hash function defined using the compression function $\mathcal{F}$. The hash function f takes an input of arbitrary length and outputs a string of r bits. We add a compression function g with an input of r bits and an output of r' bits so that $g \circ f$ takes an input of arbitrary length and outputs r' bits.

Problem 1. Given f, a collision resistant hash function, what necessary/sufficient conditions should g verify for $g \circ f$ to be collision resistant too?

It is clear that if g is collision resistant, then $g \circ f$ is also collision resistant. However, it is also clear that, as f is collision resistant, the necessary condition on g can be much weaker: even if one can easily construct collisions on g, he will still have to invert f to get collisions on $g \circ f$. This means that finding collisions on $g \circ f$ requires to find collisions on g which have special properties for f.

Let us study the following example. We note $h_0 = f(m_0)$ and $h_1 = f(m_1)$ for two message m_0 and m_1 and define g by:

$$\begin{cases} g(h) = \text{the first } r' \text{ bits of } h, \\ g(h_0) = 0, \\ g(h_1) = 0. \end{cases}$$

The function g only consists in truncating its input, except for the two special inputs h_0 and h_1. In this case $g(h_0) = g(h_1)$ and both h_0 and h_1 have known preimages for f, which means that we can find a collision on $g \circ f$. However, if g is defined without any calls to f, finding a collision on g and trying to find preimages of the colliding inputs for f will never be possible.

We modify our example and remove the two special inputs h_0 and h_1. It is still possible to find collisions on $g \circ f$ without finding collisions on f: it is enough to find near-collisions on f, that is, message hashes matching on the first r' bits only.

What we can conclude from this is that the necessary condition on g is not absolute, it is relative to f. We will thus fall back to the simpler case where we choose g to be collision resistant.

3.2 Choosing a Collision Resistant Final Transform

If we want $g \circ f$ to be provably collision resistant we need g to be provably collision resistant. Unfortunately the choice for provably resistant compression functions is very limited. The

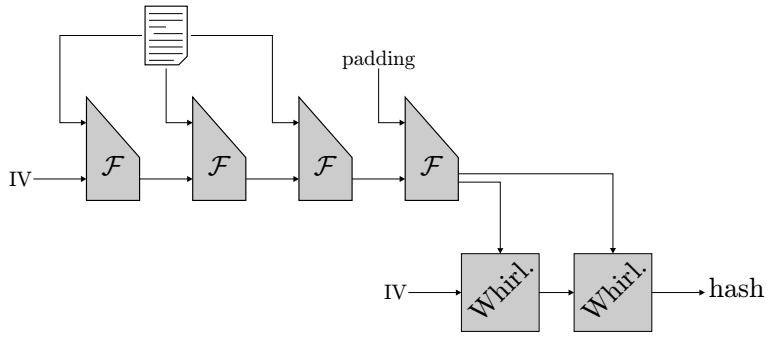

Fig. 1. Adding Whirlpool as a final compression function.

first idea would be to use the same construction as $\mathcal{F}$, but we have seen that the output can never be as short as twice the security level. If we want to preserve efficiency we also need a fast function, especially with a small initialization overhead.

We propose to use the hash function Whirlpool [16] and then truncate its output to r' bits. As the output of f is of constant size, we can simplify the padding in Whirlpool, and simply split the r bits into blocks of 512 and pad with zeros. This is depicted on Fig. 1.

This construction as several advantages:

- it is very flexible and can adapt to any parameters as long as $r' \leq 512$,
- it is quite fast, even with 512 bits blocks,
- even if collisions on Whirlpool can be found, it is very unlikely that this will be done easily, and no generic property making near-collision search on f easier is likely to exist. More specifically, our construction is weak when using a linear final transformation, but Whirlpool resists linear cryptanalysis well.

4 Reducing the Matrix Size with Quasi-Cyclic Codes

Quasi-cyclic codes have been known for more than 40 years, they are compact codes which can lead to good code constructions [12]. Meanwhile since cyclic codes had even more interesting properties, for a long time quasi-cyclic have been considered more as interesting objects rather than codes suitable for applications.

But recently things have changed and they have been used for new applications in different contexts. The main idea is to use their compactness (the fact that they can be easily described by giving only a few rows) to reduce the size of the objects required for the application. They are used in coding theory in standards for LDPC codes [8], but also in cryptography to reduce the size of parameters for code based cryptography [9] and for stream ciphers with a code-based security reduction [10].

In this section we explain how they can also be used in a context of a hash function with a security reduction.

4.1 Quasi-Cyclic random codes

In this part we recall basic facts about quasi-cyclic (QC) codes.

Definition 2. *A code of length n is called* quasi-cyclic *of index σ (for n a multiple of σ), if every cyclic shift of a codeword by σ coordinates is again a codeword.*

Remark 1. A cyclic code is a quasi-cyclic code with $\sigma = 1$.

Remark 2. Equivalently, QC codes of index σ can be seen as codes invariant under a permutation obtained as the concatenation of σ intersection-free permutation cycles of size $\frac{n}{\sigma}$.

A particular class of QC codes is the class of QC codes obtained by the concatenation of cyclic (circulant) codes. For $n = r \times \sigma$, a QC code of index σ can be constructed as the concatenation of σ circulant matrices of size $r \times \sigma$. In that case the knowledge of the first row permits to recover a whole generator matrix for the code by applying $r - 1$ times, a permutation with σ cycles of size $\frac{n}{\sigma}$. All QC codes of index σ, length $n = r\sigma$ and dimension r are described by the previous construction. Moreover when the first row is random one says that the code is random QC.

We propose to use such a QC code to define $\mathcal{H}$. This way, giving only the first row of $\mathcal{H}$ is enough to entirely describe $\mathcal{H}$. This gains a factor r on the size of the matrix to store, which is not negligible when $r \approx 512$. It is also important to note that each column of $\mathcal{H}$ is simply a cyclic shift of a r bit block of the first row of $\mathcal{H}$. As cyclic shifts can be implemented efficiently it is not necessary to rebuild $\mathcal{H}$ entirely before using it, which means that the size gain is also gained in memory at runtime.

4.2 Security of Quasi-Cyclic Matrices

The usual problem considered in code based cryptography is the SD problem. The confidence in the security of the problem relies on two main elements, first the semantic security of the problem which has been proven NP-complete for a random matrix in [3] and second the property of random codes in term of minimum distance. The minimum distance of a code is a security parameters used to evaluate the complexity of the best decoding algorithm for random codes, namely the information set decoding to which the most efficient associated algorithm is the Canteaut-Chabaud algorithm [4].

It is well known that, in term of minimum distance, codes defined by a random dual matrix are good (this is the Gilbert-Varshamov bound). Consider a $[n, \geq k, d]$ code defined by its dual, a random $(n - k) \times n$ matrix. Then the probability that a random element of F_2^n belongs to the code is $2^{-(n-k)}$ (it comes from the randomness of the matrix) and the Gilbert-Varshamov bound implies that there exists a code with minimum distance d such that:

$$\sum_{i=0}^{d-1} \binom{n}{i} < 2^{n-k}$$

In fact there is an even stronger result which stands that asymptotically almost all random codes lie on the bound. This bound and the uniform probability for an element of F_2^n to belong to a random code are the main parameters (associated to the decoding algorithm) to evaluate the practical security of the problem.

Now consider random QC codes. For such codes this result does not hold directly but if we accept a small constraint on the length of the code, it is possible to get a similar result.

We saw that a random QC code could be seen, in our case, as the concatenation of random cyclic matrices of size r. For such codes, as we cannot rely on the complete randomness of the matrix, it is not obvious that the syndromes are distributed uniformly. However we can obtain such a property by considering special lengths r. If r is prime and such that 2 is primitive root

of Z/rZ, then $x^r - 1 = (x+1)(1 + x + x^2 + \cdots + x^{r-1})$ and in that case the circulant matrix generated by any random word of odd weight is invertible. This ensures, in terms of minimum weight and probability distribution of the syndrome, that we get the same type of properties that for random linear codes. This result was shown by Chen, Peterson and Weldon in [5] (see also [12], p.507 and [11]). Although it is not known whether an infinite number of such r exist, such r are known up to 10^{50} and for the cases we are interested in (small r under 2048) many are known.

Hence if we start from a random QC code C $[n, n-r]$ given by its dual code of length $n = r\sigma$ constructed as a concatenation of σ random circulant matrices of length r such that 2 is a primitive root of Z/rZ we obtain that the probability that a random codeword belongs to C is close to 2^{-r} and hence such codes behave like a purely random code with the same length and dimension (in fact it is even a little better, see [11] for more details). In particular it means that such codes satisfy the Gilbert-Varshamov bound.

For semantic security it is not known whether decoding random QC codes is NP-complete (although heuristic elements seem to go in that direction), but eventually the semantic security is less important in practice than the best known algorithm to solve a problem. As far as we know, all classical algorithms used to decode random codes are not more efficient for decoding random QC codes and the best decoding algorithm for decoding random QC codes is still [4] which does not take advantage of the particular structure of these codes. Moreover the existence of an algorithm able to decode up to the GV bound in polynomial or sub-exponential time a family of such QC codes would certainly be a major breakthrough in coding theory. Similarly, concerning Wagner's generalized birthday technique, the quasi-cyclicity of the matrix does not change the average number of solutions to one instance, and thus cannot improve the complexity or the algorithm.

The conclusion of this section is that as soon as we take random QC codes with the previous constraint on the length we can be fairly confident that decoding QC codes has the same complexity than for random codes.

Remark 3. One may wonder why not use other kind of compact description of a matrix, for instance why not take a matrix defined by a LFSR. It is of course possible and one would get a compact matrix but no security argument. QC codes ensure a security argument concerning their minimum weight and the distribution of the syndrome and because of their cyclic structure are really easy to handle (even more than LFSR).

5 Further Reducing the Matrix Size

If we want to reduce the matrix more, it seems difficult to do it without modifying the construction a little. There are two main directions we can follow: change the *constant weight encoder* (the algorithm which converts the s input bits into a word of given weight), or work in a different finite field, say GF(2^m) instead of GF(2). Note that these improvements can also directly apply to the original construction: if for some reason one would not want to use a quasi-cyclic matrix, he can still benefit from these two improvement. In particular, using a larger finite field can represent a huge gain as in addition to reducing the size of the rows of $\mathcal{H}$, it also reduces the number of rows of the matrix.

5.1 Using an Alternate Constant Weight Encoder

If we want to change the constant weight encoder, we have to be sure how the security of the system will evolve. In our case, the complexity of Wagner's algorithm is easy to bound as it only depends on the maximum size of the lists that one can build. When looking for collisions, one need to build 2^a lists of size $2^{\frac{r}{a+1}}$, such that the XOR of any selection of one element in each list corresponds to a given pair of input messages. We know that there are exactly 2^s possible input messages, so there are at most 2^{2s} pairs of messages. This means that any parameter a suitable for Wagner's attack (that is, allowing sufficiently large lists) must verify:

$$\left(2^{\frac{r}{a+1}}\right)^{2^a} \leq 2^{2s},$$

which is equivalent to:

$$\frac{2^a}{a+1} \leq \frac{2s}{r}. \tag{4}$$

This means that the security of the compression function does not depend on the constant weight encoder, it only depends on s and r. If one wants the fastest possible hash function, he will have to use the fastest possible constant weight encoder, that is regular words, however, if one wants the smallest possible matrix and can afford to lose a little speed, other encoders might be more suitable.

As we have seen, it is always possible to decrease w and increase n in order to improve the speed of the hash function (with the same compression ratio, the number of XORs is smaller). It is thus also possible to increase w and decrease n and the size of the matrix, which means that to compare different techniques we will have to fix w. We start from parameters n, w, r, and s, allowing a given security level when used with regular words, and look for other encoders allowing parameters n', w, r, and s with the same security level.

Optimal Encoder. The first solution is to use an optimal encoder, that is, map all the words of weight w and length n'. There are $\binom{n'}{w}$ such words, so we need $s = \lfloor \log_2 \binom{n'}{w} \rfloor$. Note that, as $\binom{n'}{w}$ cannot be a power of 2, we loose a fraction of bit and some words cannot be mapped. However, $s = w \log_2 \frac{n}{w}$ so we get approximately:

$$n' = n \times \frac{w!^{\frac{1}{w}}}{w}$$

Typical values of w around 100 thus yield a reduction by a factor around 0.4. However this reduction in size has a cost: optimal constant weight encoding requires to manipulate large integers and compute some binomials: the s input bits are first converted to an integer between 0 and 2^s, and the indexes of the columns of $\mathcal{H}$ to XOR require to do some divisions of this integer. This is very costly whatever the context and slows down the computation too much.

Tradeoffs. What we are looking for is a tradeoff in order to be at the same time fast and near-optimal. An interesting approach based on information theory was presented by Sendrier in 2005 [18], but the number of input bits it takes is variable: this might be suitable in some settings where bits can be read one at a time, but for a hash function this is not very convenient

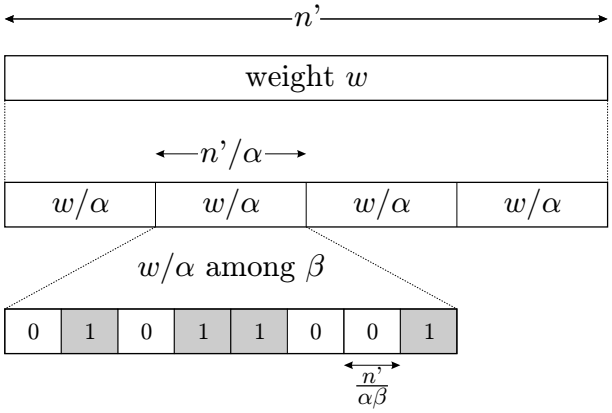

Fig. 2. A tradeoff between optimal encoding and regular word encoding.

and we prefer reading exactly $s - r$ bits from the document at each round. What we suggest here is to use a "three level" combination of regular and optimal encoding.

As depicted on Fig. 2, the word is first split in α blocs (as for regular words), of weight $\frac{w}{\alpha}$ each. Then each bloc is split again in β sub-blocs and $\frac{w}{\alpha}$ sub-blocs are chosen among the β (as for the optimal encoder). Finally, for each of the $\frac{w}{\alpha}$ sub-blocs we choose one column among the $\frac{n'}{\alpha\beta}$ (as for regular words). The total number of input bits required to code such a word is

$$s = \alpha \cdot \lfloor \log_2 \binom{\beta}{w/\alpha} \rfloor + w \cdot \log_2 \tfrac{n'}{\alpha\beta}.$$

Evaluating the efficiency of such coding is not as easy as before, and also, there are a few constraints on α and β in order to have integer values everywhere. A good choice could be $\beta = 64$ and $\alpha = \frac{w}{8}$ so that $\log_2 \binom{\beta}{w/\alpha} \approx 32.043$, and the costly step of optimal encoding can be performed using only operations on 32 bits, with no need for large integers. With these values of α and β we get:

$$s = \frac{w}{8} \cdot 32 + w \cdot \log_2 \frac{n'}{8w} = w \log_2 = w \cdot \log_2 \frac{2n'}{w}.$$

This means that compared to regular encoding, we can use $n' = \frac{n}{2}$. This is nearly as good as with the optimal encoding, but no longer requires any large integer operations. This tradeoff is probably a good choice for someone seeking the smallest possible matrix size.

5.2 Working in a Larger Field

Another solution to reduce the matrix size is to use a matrix in $\mathrm{GF}(2^m)$ instead of $\mathrm{GF}(2)$. We shall consider that this new matrix has size $n' \times r'$, so that, if we use a quasi-cyclic matrix, we have to store $m \cdot n'$ bits. The compression function then consists in converting the s input bits into a word of weight w and length n (now, each non zero coefficient can take $2^m - 1$ different values), and multiplying this vector by the matrix $\mathcal{H}$. We no longer simply XOR some columns together, we first need to multiply the columns by some scalars, then XOR them together. The reader must thus be aware that this technique can by no means be as fast as the previous technique: even using a small finite field, with the multiplication stored in a table, these multiplications will be very long compared to the other operations.

First of all, for the compression function to output r bits we need $r = m \cdot r'$. Then, if we consider that Wagner's attack is still the best attack against this modified construction, as long as s input bits are used, the construction will be secure. Whatever the constant weight encoder we use, we need the same number of bits to defined the non-zero positions in the word and then need an extra $w \cdot \log_2(2^m - 1)$ bits to choose the non-zero coefficients of the word. This means that the amount by which n can be reduced will depend on the constant weight encoder. Using simple regular words we have $s = w \cdot \log_2 \frac{n}{w}$ on GF(2) and $s = w \log_2 \frac{n'}{w} + w \log_2(2^m - 1) = w \log_2(\frac{n'}{w}(2^m - 1))$ on GF(2^m). This means one can use $n' = \frac{n}{2^m - 1}$, so the size of the matrix is $m \cdot n' = \frac{m}{2^m - 1} n$.

Choosing $m = 8$ is enough to divide the size of $\mathcal{H}$ by 32. Working in a larger field makes it possible to strongly reduce the size of $\mathcal{H}$, however it comes at a cost: first the user will have to choose between storing multiplications tables for the larger field or computing the multiplications directly, thus using some memory or some time, second, the security reduction to the Syndrome Decoding problem works fine in GF(2), but not in larger fields. Also, depending on the parameters, the best attack might no longer be Wagner's generalized birthday technique. Parameters for this variant of the construction should be chosen with care. Finally, it should be noted that this technique can only be used to reduce the size of $\mathcal{H}$ when w is much smaller than n. For parameters where w becomes closer to n (that is, parameters offering the smallest size of $\mathcal{H}$), the maximum size of the field one can use will be very limited.

This technique, which seems very efficient at first glance, ends up being not that good at providing very small matrices. The shortest matrices will probably still be obtained using a large w and a small n, together with an efficient constant weight encoder.

6 Proposed Parameters

6.1 Implementation Considerations

One of the aims of the reduction of the matrix size was to have it fit into a standard CPU cache. This way, hashing could maybe be faster. However, the cost of the cyclic shifts is not negligible. In order to avoid doing too many cyclic shifts, an idea is to precompute 7 cyclic shifts of the first row (of 1 to 7 bits) of the matrix, so that any other row of the matrix is a cyclic shift of one of the 8 precomputed rows by a multiple of 8. This way only shifts on bytes are required. Then, if we double the length of each vector, a cyclic shift can be replaced by a simple shift, which simply consists in reading the memory with a given offset (see Fig. 3).

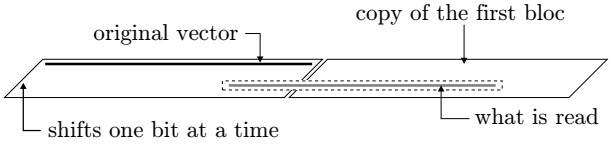

Fig. 3. Precomputation of some shifts of a vector.

However, this technique uses 16 times more memory than computing the shifts on the fly. The first row of the matrix should thus be at most one sixteenth of the cache size.

6.2 Experimental Results

In order to better analyse the effects of the cache/matrix size, we implemented two versions of the hash function, both using regular words for the constant weight encoding:

- H_0 - the standard function, as described in [1]
- H_8 - our variant using a quasi-cyclic matrix and the technique of Fig. 3

The results we obtained are reported in Table 1. Note that to hash the same test file of 200MB, the programs `md5sum`, `sha1sum`, and `sha256sum` respectively take 0.5, 2.6 and 4.5 seconds. These hash function implementations are however not fully optimized, so for comparison, we included in Table 1 the results obtained in [14]. The speed we reach is thus still below that of these hash functions but the gap is not that huge. Moreover, our implementation of the hash function is not fully specialized: it can accept various parameters. A fully optimized implementation would thus probably be significantly faster. Also, an implementation dedicated to a specific set of parameters would make it possible to use a quasi-cyclic code with the properties described in Section 4.2 without slowing down the hash process. This was not the case with our generic implementation, which explains why powers of 2 are used every time it is possible.

Table 1. Implementation results: comparison of H_0 and H_8. The security (secu.) is the base 2 logarithm of the complexity of Wagner's attack, the time indicated is the time taken to hash a test file of 200MB (on a Pentium D 2.8 GHz computer, using Intel's `icc` compiler), the size of $\mathcal{H}$ is given in bytes, the number of cycles per byte is the average over the 200MB test file.

						H_0 – standard			H_8 – our variant		
secu.	r	w	n	s	$\frac{n}{w}$	size of $\mathcal{H}$	time	cyc./byte	size of $\mathcal{H}$	time	cyc./byte
64	512	512	131 072	4 096	256	8 388 608	28.8s	390.6	16 384	6.6s	89.3
	512	450	230 400	4 050	512	14 745 600	43.1s	587.9	28 800	12.1s	165.1
	1 024	2^{17}	2^{25}	2^{20}	256	2^{32}	–	–	4 194 304	25.0s	339.8
80	512	170	43 520	1 360	256	2 785 280	37.7s	517.0	**5 440**	**20.5s**	**281.1**
	512	144	73 728	1 296	512	4 718 592	42.6s	581.6	9 216	17.6s	239.8
128	1 024	1 024	262 144	8 192	256	33 554 432	48.6s	669.6	**32 768**	**8.9s**	**121.0**
	1 024	904	462 848	8 136	512	59 244 544	72.4s	989.9	57 856	27.2s	371.2
	1 024	816	835 584	8 160	1 024	106 954 752	53.4s	727.6	104 448	11.8s	162.6
64	MD5					best known implementations from [14]					3.7
80	SHA-1										8.3
128	SHA-256										20.6

6.3 Two Interesting Sets of Parameters

We propose two sets of parameters: the first one is intended for standard hash applications, the second one is solely intended for constrained environments and is principally focused on reducing the size of the matrix.

For Standard Applications. We recommend to use the parameters of line 6 of Table 1 and modify them a little to fit the constraints presented in Section 4.2. We use a binary QC matrix with $r = 1061$ but only keep 1024 bits of output, that is, we drop the bottom 37 rows

of the full matrix $\mathcal{H}$. This allows to have both the desired properties concerning QC codes and read bytes (and not bits) in input. Then, we take $w = 1061$ and $n = 271616$ (so that $\frac{n}{w} = 256$), using regular words for constant weight encoding. This way, the cost of collision search using Wagner's technique is at least 2^{128}, which means that the final transform should output a hash of 256 bits. We thus apply two rounds of Whirlpool and truncate the output to 256 bits.

With these parameters our construction is just about two times slower than SHA-256 and reaches a throughput of 180 Mbits/s on a 2.8 GHz Pentium D processor. An optimized implementation could probably go even faster. However, contrarily to SHA-256, we have strong security arguments showing that finding a collision is a hard problem.

For Memory Constrained Environments. As discussed in Section 5, we propose to use the tradeoff of Fig. 2 for constant weight encoding. To have the shortest possible n we also need the largest possible w. We aim at a security level of 2^{80} binary operations and choose $r = 512$, $n = 2560$ and $w = 320$. The length n is split in 40 blocks of 64 bits and 32 input bits are used to determine a pattern of weight 8 among each blocs. The compression function thus compresses $s = 1280$ bits into 512 bits. A final round of Whirlpool can then be used to reduce the output to 160 bits. It is much slower than the previous parameters, but the matrix is only 320 bytes long! This should be small enough for almost any environment, except small RFID tags (on which implementing Whirlpool is probably already a problem).

Note that this set of parameters does not respect the constraints of Section 4.2. It is unfortunately impossible to get such a small matrix with compliant parameters. Not complying to the constraints could lead to minor improvements in the attacks, but nothing is known about this yet.

7 Conclusion

Starting from the provably secure hash function family presented in 2005 by Augot, Finiasz, and Sendrier [1], we have introduced two major improvements: a final transformation and some methods to reduce the matrix size, without any practical security loss. At the same time, this has greatly improved the speed of the hash function, which is now fast enough to compare with other constructions. We propose two different sets of parameters: one for fast implementation with a high security level of 128 bits, and another one, requiring only a very small matrix of 320 bytes, not really intended for practical use, but more to show that with our new improvements, the size of the matrix is no longer a serious issue.

References

1. D. Augot, M. Finiasz, and N. Sendrier. A family of fast syndrome based cryptographic hash functions. In E. Dawson and S. Vaudenay, editors, *Mycrypt 2005*, volume 3715 of *Lecture Notes in Computer Science*, pages 64–83. Springer, 2005.
2. K. Bentahar, D. Page, M-J.O. Saarinen, J.H. Silverman, and N.P. Smart. LASH. 2nd NIST Cryptographic Hash Workshop, 2006.
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. 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. C. Chen, W. Peterson, and E. Weldon. Some results on quasi-cyclic codes. *Information and Control*, 15:407–423, 1969.
6. S. Contini, A.K. Lenstra, and R. Steinfeld. VSH, an efficient and provable collision-resistant hash function. In S. Vaudenay, editor, *Eurocrypt 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 165–182. Springer, 2006.
7. I.B. Damgård. A design principle for hash functions. In G. Brassard, editor, *Advances in Cryptology - Crypto '89*, volume 435 of *Lecture Notes in Computer Science*, pages 416–427. Springer-Verlag, 1990.
8. M. Fossorier. Quasi-cyclic low density parity check codes from circumant permutation matrices. *IEEE Transactions on Information Theory*, 50(8):1788–1793, 2004.
9. P. Gaborit. Shorter keys for the mceliece cryptosystem. In *Proceedings of WCC 2005*, pages 81–90, 2005.
10. P. Gaborit, C. Lauradoux, and N. Sendrier. SYND: a fast code-based stream cipher with a security reduction. To appear at ISIT 2007.
11. P. Gaborit and G. Zémor. Asymptotic improvement of the gilbert-varshamov bound for linear codes. In *IEEE Conference, ISIT'2006*, pages 287–291, 2006.
12. F. J. MacWilliams and N. J. A. Sloane. *The Theory of Error-Correcting Codes*. North-Holland, 1977.
13. R.C. Merkle. One way hash functions and DES. In G. Brassard, editor, *Advances in Cryptology - Crypto '89*, volume 435 of *Lecture Notes in Computer Science*, pages 428–446. Springer-Verlag, 1990.
14. J. Nakajima and M. Matsui. Performance analysis and parallel implementation of dedicated hash functions. In L. Knudsen, editor, *Advances in Cryptology - Eurocrypt '02*, volume 2332 of *Lecture Notes in Computer Science*, pages 165–180. Springer-Verlag, 2002.
15. National Insitute of Standards and Technology. *FIPS Publication 180: Secure Hash Standard*, 1993.
16. V. Rijmen and P. Barreto. Whirlpool. Seventh hash-function of ISO/IEC 10118-3:2004, 2004.
17. R.L. Rivest. The MD5 message-digest algorithm. Request for Comments (RFC) 1321, April 1992.
18. N. Sendrier. Encoding information into constant weight words. In *IEEE Conference, ISIT 2005*, Adelaide, Australia, 2005.
19. D. Wagner. A generalized birthday problem. In M. Yung, editor, *Crypto 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 288–304. Springer, 2002.

A Improving Inversion Attacks

In the short appendix, we show that it is possible to slightly improve the complexity of the inversion on both the original construction and our improvement. Usually, inversion consists in finding a message m such that $h(m)$ is equal to a given target. However, one can choose to attack several targets at a time and try to find a message m such that $h(m)$ is contained in a set of targets. This slightly different context does not improve attacks on an ideal hash function, but can slightly improve Wagner's attack.

For standard inversion, Wagner's attack consists in choosing a parameter a such that:

$$\frac{r}{a+1} \leq \frac{s}{2^a} \quad \Leftrightarrow \quad \frac{2^a}{a+1} \leq \frac{s}{r},$$

and then provides an attack with complexity $\mathcal{O}(2^{\frac{r}{a+1}})$. If we allow multiple targets, we can choose (without increasing the complexity of the attack) up to $2^{\frac{r}{a+1}}$ different targets. This means that the previous equation can be modified into:

$$\frac{r}{a+1} \leq \frac{s + \frac{r}{a+1}}{2^a}.$$

And we obtain a new constraint:

$$\frac{2^a - 1}{a+1} \leq \frac{s}{r}.$$

This modification does not radically change the complexity of inversion, but it probably gains a little in some situations where the maximum a is small. For instance, the complexity of inversion using the one-way parameters proposed in [1] is probably a little bellow what is announced.