Cryptanalysis of a Hash Function Based on Quasi-Cyclic Codes

Pierre-Alain Fouque, Gaëtan Leurent

École Normale Supérieure – Département d'Informatique,
45 rue d'Ulm, 75230 Paris Cedex 05, France
{Pierre-Alain.Fouque,Gaetan.Leurent}@ens.fr

Abstract. At the ECRYPT Hash Workshop 2007, Finiasz, Gaborit, and Sendrier proposed an improved version of a previous provably secure syndrome-based hash function. The main innovation of the new design is the use of a quasi-cyclic code in order to have a shorter description and to lower the memory usage.
In this paper, we look at the security implications of using a quasi-cyclic code. We show that this very rich structure can be used to build a highly efficient attack: with most parameters, our collision attack is faster than the compression function!
Key words: hash function, provable security, cryptanalysis, quasi-cyclic code, syndrome decoding.

1 Introduction

Following the breakthrough collision attacks by Wang *et al.* against the most widespread hash functions (MD5 in [9], SHA-1 in [8]) the crypto community is trying to design new hash functions. One interesting approach is the construction of *provably secure* hash functions, in which the security of the hash function is proven to rely on some computationally hard problem.

At Mycrypt 2005, Augot, Finiasz, and Sendrier proposed a family of provably secure hash functions based on the syndrome decoding problem called FSB [1]. An improved version of this design (IFSB) was presented at the ECRYPT Hash Workshop 2007 by Finiasz, Gaborit and Sendrier [5]. The new idea introduced in IFSB is to use of a quasi-cyclic code instead of a random code. This allows to store a smaller description of the code: there is a huge speedup when it fits into CPU cache. This modification was assumed not to lower the security of the construction.

However, this new proposal was broken by Saarinen using a simple linearization technique [6]. The attack does not take advantage of the new elements of the design (*ie.* the quasi-cyclic codes) and can also be used to break the initial function; it is only based on the fact that the output size r is relatively small ($r < 2w$). Therefore, it does not invalidate the approach of [5]; we can still use a hash function based on a quasi-cyclic code, we just need to choose a larger r.

In this paper, we look at the security implications of using a quasi-cyclic code. We show that this very rich structure can be used to build a highly efficient attack: with most parameters, our collision attack is faster than the compression function!

Our Results. We first point out a strange property of the FSB/IFSB family in Section 3: the mixing of the chaining value with the message is very weak. This has two main consequences: a collision attack is essentially the same as a pseudo-collision attack, and the compression function can't be used as a PRF.

In section 4, we introduce our new collision attacks on IFSB based on the structure of quasi-cyclic codes. The main result is an attack using piecewise periodic messages, and a new algorithm

to solve a system of cyclic equations. Here is a brief comparison (for pseudo-collisions) of the previous linearization attack and our new cyclic attack (see Table 1 on page 12 for practical figures):

Attack	Conditions	Complexity	Remarks
Linearization	$r \leq 2w$	r^3	r is typically 1024
	if r is bigger	$(4/3)^{r-2w} \cdot r^3$	$\log_2(4/3) \approx 0.415$
Cyclic	$r \leq 4w$	$(n/4w)^3$	$n/4w$ is typically 64
	if r is bigger	$2^{\frac{n(r-4w)}{4wr}} \cdot (n/4w)^3$	$n/4wr$ is typically $1/16$

These attack are both very efficient when the ratio r/w is below a given threshold, and can be extended to work above the threshold, but the complexity grows exponentially with r. The cyclic attack has a bigger threshold, and when the threshold is exceeded it has a lower factor in the exponent. Note that the linearization attack can break FSB as well as IFSB, whereas our cyclic attack relies on the property of the quasi-cyclic code: it can only break IFSB and also requires r to be a power of 2.

Since these attack rely on a low r/w ratio, they do not rule out the IFSB construction, but only the parameter proposed in [5]. For instance, one could use the IFSB construction with parameters based on [1], which have a higher r/w. However, the first step of our attack can also be used together with Wagner's attack, so as to remove the dependency in the ration r/w (there is still a requirement that r is a multiple of $n/2w$):

Attack	Complexity	Remarks
Wagner	$r2^{a'} \cdot 2^{r/(a+1)}$	Used as the security parameter
Cyclic + Wagner	$\frac{n}{2w}2^{a'} \cdot 2^{\frac{n}{2w}/(a'+1)}$	a' is a or $a-1$

Since $n/2w$ is typically between $r/2$ and $r/8$, the new attack will have a complexity between the square root and the eighth root of the security parameter. These attacks basically show that the IFSB construction with a quasi-cyclic code is not secure if r, the length of the code has many divisors. The easy choice of a power of two is highly insecure.

In [5], the authors provided some security argument when r is prime and 2 is a primitive root of $\mathbb{Z}/2\mathbb{Z}$. Since this was not a real proof, and they were confident in the security of quasi-cyclic codes even without this argument, some of their parameters don't respect this constraint. Our attacks show that IFSB should only be used with a prime r.

About Provable security. The main motivation for the design of FSB is to have a *proof of security*. In [1], the authors of FSB defined the 2-RNSD problem (2-Regular Null Syndrome Decoding) so that finding a collision in FSB given the matrix $\mathcal{H}$ is equivalent to solving 2-RNSD on the matrix $\mathcal{H}$. They also prove that 2-RNSD is a NP-complete problem, which is a good evidence that there is no polynomial time algorithm to break FSB. However, this does not really prove that finding a collision is hard...

– The fact that there is no polynomial time algorithm to break the function is an *asymptotic* property, but in practice the function is used with a *fixed size*; there might be an algorithm that break the function up to a given size in very little time. Usually, designers look at the best known attack and choose the size of the function so that this attack is unpractical, but there could be a more efficient algorithm (though superpolynomial). Indeed, the first version of FSB did not consider Wagner's generalized birthday attack [3], and the parameters had to be changed.

– More importantly, the fact that a problem is NP-Complete means that there are *some* hard instances, not that *every* instance is hard. For instance, SAT is an NP-Complete problem, but if the specific formula you try to satisfy happen to be in 2-SAT there is a polynomial time algorithm. In the case of FSB, the 2-RNSD problem is NP-Complete but the result of [6] is that it becomes easy if the parameters are such that $r < 2w$, which is the case in the proposed settings of IFSB.
– Moreover, IFSB is an improved version of FSB, but the new components (the final transformation, the change to a quasi-cyclic matrix, and the possibility to use a new constant weight encoder) have no security proof... Indeed, our main result is an algorithm that breaks the 2-RNSD problem when the matrix is quasi-cyclic and r is a power of 2, which is the case for most proposed settings of IFSB.
– Lastly, the security proof considers some specific attack, such as collision, or preimage, but there might be some other undesirable property in the design. In Section 3, we show that the FSB design does not mix properly the chaining value with the message.

2 Design of IFSB and previous cryptanalysis

The Fast Syndrome Based (FSB) and Improved Fast Syndrome Based (IFSB) follow the Merkle-Damgård construction. The compression function is built of two steps:

– A constant weight encoder φ that maps an s-bit word to a n-bit word of Hamming weight w.
– A random matrix $\mathcal{H}$ of size $r \times n$. Typically, r is about one hundred, and n is about one million.

The compression function of FSB takes s bit as input (r bits of chaining variable, and $r - s$ bits of message) and outputs r bits; it is defined by:

$F(x) = \mathcal{H} \times \varphi(x)$
$\mathcal{H}$: random $r \times n$ matrix φ: encodes s bits to n bits with weight w

In the case of IFSB, the matrix $\mathcal{H}$ is quasi-cyclic: $\mathcal{H} = \mathcal{H}_0||\mathcal{H}_1||...\mathcal{H}_{\sigma-1}$ and each $\mathcal{H}_i$ is a cyclic $r \times r$ matrix. There is also a final transformation to reduce the size of the digest, but it is not used in our collision attacks.

2.1 Choosing a Constant Weight Encoder

Three constant word encoders are proposed in [5]. The regular encoder is a very simple one, used in most parameters. The optimal encoder and the tradeoff encoder are introduced in order to reduce the size of the matrix: they can use more input bits with the same parameters n and w.

Notations. We will use the following notations:

– `0` and `1` are bit-strings of length one, as opposed to 0 and 1.
– if x is a bit-string, x^k is the concatenation of k times x.
– $x^{[i]}$ is the i-th bit of x.
– $[f(i)]_{i=0}^{p-1}$ is the matrix whose *columns* are the $f(i)$'s.

The regular encoder. The regular encoder was introduced in [1]; it is the only encoder defined for the FSB family, and the main one for the IFSB family. It is designed for efficiency and is very simple: the output word is divided into chunks of n/w bits and each chunk contains exactly one non-zero bit, with its position defined by $\log(n/w)$ bits of the message (all the $\log$ in this paper are base 2). For an efficient implementation, we often choose $\log(n/w) = 8$.

Let us define $\psi : \{0..n/w-1\} \mapsto \{0,1\}^{n/w}$ such that $\psi(x)^{[i]} = 1 \iff i = x$. We construct φ_i that encode $\log(n/w)$ bits into a word a weight 1, such that the non-zero bit is in the i-th chunk: $\varphi_i(x) = 0^{in/w}\psi(x)0^{n-(in+1)/w}$. Then we have

$$\varphi(M) = \varphi(m_0, m_1, ...) = \bigoplus_{i=0}^{w-1} \varphi_i(m_i).$$

FSB with the regular encoder just selects one particular column of the matrix $\mathcal{H}$ for each message chunk of size $\log(n/w)$:

$$F(M) = \mathcal{H} \times \bigoplus_{i=0}^{w-1} \varphi_i(m_i) = \bigoplus_{i=0}^{w-1} \mathcal{H}_{in/w+m_i}.$$

The optimal encoder. The optimal encoder tries to have all the words of weight w in its range, whereas the regular encoders only output regular words. There are $\binom{n}{w}$ such words, as opposed to $(n/w)^w$ regular words; this allows the optimal encoder to use more input bits. The optimal encoder will actually map $\lfloor \log \binom{n}{w} \rfloor$ bits to a subspace of $2^{\lfloor \log \binom{n}{w} \rfloor}$ words. We do not consider the details of the computation, we will only assume that φ and φ^{-1} are efficiently computable.

The tradeoff encoder The main problem of the optimal encoder is that it requires some computations with very large integers. Therefore, the tradeoff encoder uses a combination of the optimal encoder and the regular encoder. Here again, we do not need the details of the construction, we only use the fact that φ and φ^{-1} are efficiently computable.

2.2 Wagner's Generalized Birthday

Wagner's generalized birthday attack [7] is a clever trick to solve the k-sum problem: given some lists $L_1, L_2, .., L_k$ of r-bit values, we want to find $l_1 \in L_1, .., l_k \in L_k$ such that $\bigoplus_{i=1}^{k} l_k = 0$. If each list contains at least $2^{r/k}$ elements there is a good probability that a solution exists, but we have no algorithm to find it more efficiently than using the simple birthday paradox in time and memory $2^{r/2}$.

Indeed, Wagner's algorithm requires more elements in the lists; for instance with $k = 4$ we need lists of size $2^{r/3}$ and the algorithm will find one solution using $2^{r/3}$ time and memory.

The basic operation of the algorithm is the general join $\bowtie_j$: $L \bowtie_j L'$ consists of all elements of $L \times L'$ that agree on their j least significant bits:

$$L \bowtie_j L' = \left\{ (l, l') \in L \times L' \;\middle|\; (l \oplus l')^{[0..j-1]} = 0^j \right\}.$$

The algorithm for $k = 4$ is described by Figure 1. We first build the list $L_{12} = L_1 \bowtie_{r/3} L_2$ and $L_{34} = L_3 \bowtie L_4$. By the birthday paradox, these lists should contain about $2^{r/3}$ elements. Next, we build $L_{1234} = L_{12} \bowtie_{2r/3} L_{34}$. Since the elements of L_{12} and L_{34} already agrees on their $r/3$ lower bits, we are only doing a birthday paradox on the bits $r/3$ to $2r/3$, so we still expect to

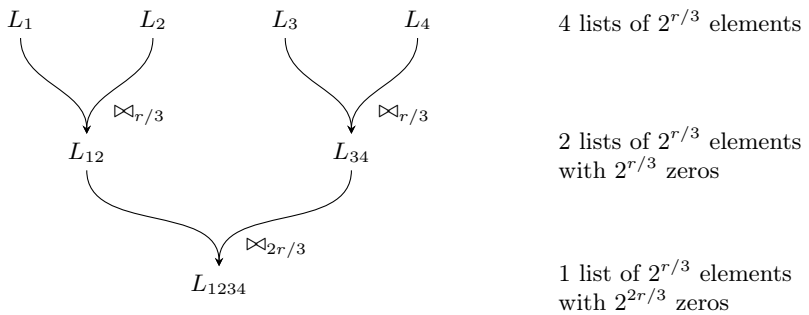

Fig. 1. Wagner's algorithm for $k = 4$

find $2^{r/3}$ elements. Finally, the birthday paradox on L_{1234} tells us that one element of this list has a sum of zero. This can be generalized to any k that is a power of two, using a binary tree: if $k = 2^a$, we need k lists of $2^{r/(a+1)}$ elements and the time and memory used by the algorithm is $2^a \cdot r2^{r/(a+1)}$.

Finding a collision in FSB with the regular encoder is just an instance of the k-sum problem, so we can use Wagner's algorithm to break it, as shown by Coron and Joux [3]. We look for M and M' such that

$$F(M) \oplus F(M') = \bigoplus_{i=0}^{w-1} \mathcal{H}_{in/w+m_i} \oplus \bigoplus_{i=0}^{w-1} \mathcal{H}_{in/w+m'_i} = 0.$$

This is an instance of the $2w$-sum problem on r bits, with lists of n/w elements (the possible values of each m_i). If $n/w < 2^{r/(\log(w)+2)}$, we cannot apply Wagner attack with $k = 2w$ as it is, but we can group the lists. For instance, we can build $w/2$ lists of $(n/w)^4$ elements, and this attack is applicable with $k = w/2$ if $(n/w)^4 \geq 2^{r/\log(w)}$. In the end, we will use the largest a that satisfies:

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

This is the main attack to break the k-sum problem, therefore it was used by the authors of FSB to define their security levels.

2.3 Linearization attack

Wagner noted in [7] that the generalized birthday problem can be solved using linear algebra when $r \leq k$, using a result from Bellare and Micciancio [2, Appendix A]. When we apply this to FSB, we can break it very efficiently when $r \leq 2w$. However, this was overlooked by the designers of ISFB, and almost all the parameters they proposed satisfy $r \leq 2w$... Saarinen rediscovered this attack in [6], and extended it to $r > 2w$ with complexity $(3/4)^{r-2w} \cdot r^3$. He also proposed a preimage attack when $r \leq w$ based on the same idea. Here, we will only describe the basic collision attack, using our notations.

Let us choose four distinct elements $a, b, c, d \in \{0..n/w - 1\}$, and build a vector u and a matrix Δ:

$$\begin{aligned} u &= (\mathcal{H} \times \varphi(a^w)) \oplus (\mathcal{H} \times \varphi(c^w)) \\ \Delta_1 &= [\mathcal{H} \times (\varphi_i(a) \oplus \varphi_i(b))]_{i=0}^{w} \\ \Delta_2 &= [\mathcal{H} \times (\varphi_i(c) \oplus \varphi_i(d))]_{i=0}^{w} \\ \Delta &= \Delta_1 || \Delta_2 \end{aligned}$$

We solve the equation $\Delta \times x = u$ by linear algebra and we write $x = x_1 || x_2$ such that $\Delta_1 \times x_1 \oplus \Delta_2 \times x_2 = \Delta \times x = u$. Since the matrix Δ has size $r \times 2w$, there is a good probability to find a solution when $r \leq 2w$ (see Appendix A). Then we build the messages M and M' (note that the messages are distinct because a, b, c and d are all distinct):

$$m_i = \begin{cases} a & \text{if } x_1^{[i]} = 0 \\ b & \text{if } x_1^{[i]} = 1 \end{cases} \qquad m'_i = \begin{cases} c & \text{if } x_2^{[i]} = 0 \\ d & \text{if } x_2^{[i]} = 1 \end{cases}$$

Proof. This will give a collision because

$$\begin{aligned} \mathcal{H} \times \varphi(M) &= \mathcal{H} \times \bigoplus \varphi_i(m_i) \\ &= \mathcal{H} \times \Big(\bigoplus_{x_1^{[i]}=0} \varphi_i(a) \oplus \bigoplus_{x_1^{[i]}=1} \varphi_i(b) \Big) \\ &= (\mathcal{H} \times \varphi(a^w)) \oplus \Big(\mathcal{H} \times \bigoplus_{x_1^{[i]}=1} (\varphi_i(a) \oplus \varphi_i(b))\Big) \\ &= (\mathcal{H} \times \varphi(a^w)) \oplus (\Delta_1 \times x_1) \\ \mathcal{H} \times \varphi(M') &= (\mathcal{H} \times \varphi(c^w)) \oplus (\Delta_2 \times x_2) \\ (\mathcal{H} \times \varphi(M)) \oplus (\mathcal{H} \times \varphi(M')) &= u \oplus \Delta \times x = 0 \end{aligned}$$

□

3 An IV Weakness

When the constant weight encoder chosen is the regular encoder or the tradeoff encoder, the message and the chaining variable are treated independently by the compression function: we can write $F(M, c) = \mathcal{H} \times \varphi(M||c) = \mathcal{H}_M \times \varphi_M(M) \oplus \mathcal{H}_c \times \varphi_c(c)$, with $\mathcal{H} = \mathcal{H}_M || \mathcal{H}_c$. This means that a collision attack is just the same as a pseudo-collision attack, we only have to work with the smaller matrix $\mathcal{H}_M$. Since we have less degree of freedom, the complexity of the attack might be higher, but it works exactly in the same way.

More importantly, a collision in the compression function is a collision for *any* chaining variable. This is quite unexpected for a hash function, and we believe this should be avoided. In particular, it means that if we replace the IV by a key, we do *not* have a pseudo-random function family. There exists an adversary that knows two messages M_1 and M_2 such that $\mathcal{H}_M \times \varphi(M_1) = \mathcal{H}_M \times \varphi(M_2)$, and we have $F(M_1, k) = F(M_2, k)$ for *any* key, which gives a distinguisher against a random function. For instance, if FSB is used in the HMAC construction, we can run an existential forgery attack with only one chosen-message MAC, as long as we know one fixed collision. Even if no collisions are known, the security against a PRF-distinguisher is only $2^{n/2}$, instead of the expected 2^n. This also allows to forge signatures for virtually any hash-based signature scheme, to build cheap multi-collisions, ...

4 The Cyclic Attack

Our new attack on IFSB relies on the structure of quasi-cyclic codes and uses two new ideas. The first idea is to use a message M such that $\varphi(M)$ is piecewise periodic. This reduces the message space, but when the code is quasi-cyclic, we will see that the hash will become periodic, and we can now work on only one period, instead of the whole hash. This step can be used as a kind of preprecessing for Wagner's attack or for the linearization attack. The second part of the attack is an algorithm to solve the remaining system of cyclic equations, which is more efficient than Wagner's attack or the linearization technique.

4.1 Quasi-Cyclic Codes and Rotations

We use $x \lll s$ to denote x rotated by s bits, and we say that x is s-periodic if $x = x \lll s$. Similarly, if x is broken into pieces of k bits $x = x_0||x_1||x_2...$, we define the k-piecewise rotation:

$$x \overset{k}{\lll} s = (x_0 \lll s)||(x_1 \lll s)||(x_2 \lll s)...$$

If $x \overset{k}{\lll} s = x$, we say that x is piecewise periodic. In this paper we will always use $k = r$.

Let us introduce a few definitions and properties of cyclic and quasi-cyclic matrices.

Definition 1. *The matrix $\mathcal{H}$ is cyclic if each row vector is rotated one element to the right relative to the previous row vector:*

$$\mathcal{H} = \begin{bmatrix} h_0 & h_1 & \dots & h_{n-2} & h_{n-1} \\ h_{n-1} & h_0 & h_1 & & h_{n-2} \\ \vdots & h_{n-1} & h_0 & \ddots & \vdots \\ h_2 & & \ddots & \ddots & h_1 \\ h_1 & h_2 & \dots & h_{n-1} & h_0 \end{bmatrix}$$

Property 1. If $\mathcal{H}$ is cyclic, we have:

$$\mathcal{H} \times (x \lll s) = (\mathcal{H} \times x) \lll s$$

Definition 2. *$\mathcal{H}$ is quasi-cyclic if $\mathcal{H} = (\mathcal{H}_0, \mathcal{H}_1, ...\mathcal{H}_{\sigma-1})$, and each $\mathcal{H}_i$ is cyclic.*

$$\mathcal{H} = \begin{bmatrix} f_0 & f_1 & \dots & f_{n-2} & f_{n-1} \\ f_{n-1} & f_0 & f_1 & & f_{n-2} \\ \vdots & f_{n-1} & f_0 & \ddots & \vdots \\ f_2 & & \ddots & \ddots & f_1 \\ f_1 & f_2 & \dots & f_{n-1} & f_0 \end{bmatrix} \begin{bmatrix} g_0 & g_1 & \dots & g_{n-2} & g_{n-1} \\ g_{n-1} & g_0 & g_1 & & g_{n-2} \\ \vdots & g_{n-1} & g_0 & \ddots & \vdots \\ g_2 & & \ddots & \ddots & g_1 \\ g_1 & g_2 & \dots & g_{n-1} & g_0 \end{bmatrix} \begin{bmatrix} h_0 & h_1 & \dots & h_{n-2} & h_{n-1} \\ h_{n-1} & h_0 & h_1 & & h_{n-2} \\ \vdots & h_{n-1} & h_0 & \ddots & \vdots \\ h_2 & & \ddots & \ddots & h_1 \\ h_1 & h_2 & \dots & h_{n-1} & h_0 \end{bmatrix} \dots$$

Property 2. If $\mathcal{H}$ is quasi-cyclic, we have:

$$\begin{aligned} \mathcal{H} \times (x \overset{r}{\lll} s) &= \sum_{i=0}^{\sigma-1} \mathcal{H}_i \times (x_i \lll s) \\ &= (\mathcal{H} \times x) \lll s \end{aligned}$$

Corollary 1. *If $\mathcal{H}$ is quasi-cyclic and x is piecewise periodic, then $\mathcal{H} \times x$ is periodic:*

$$(\mathcal{H} \times x) \lll s = \mathcal{H} \times (x \overset{r}{\lll} s) = \mathcal{H} \times x.$$

This simple remark will be the basis of our cyclic attack.

4.2 The Main Attack

The basic idea of our attack is very simple: let us choose M and M' such that $\varphi(M)$ and $\varphi(M')$ are piecewise periodic. Then we know that the output $\mathcal{H} \times \varphi(M)$ and $\mathcal{H} \times \varphi(M')$ are periodic, and we only have to collide on *one* period.

In fact we can even take further advantage of the two messages: let us choose M such that $\varphi(M)$ is piecewise s-periodic, and $M' = \varphi^{-1}(\varphi(M) \overset{r}{\lll} s/2)$. $\varphi(M) \oplus \varphi(M')$ is piecewise $s/2$-periodic, and so is $(\mathcal{H} \times \varphi(M)) \oplus (\mathcal{H} \times \varphi(M')) = \mathcal{H} \times (\varphi(M) \oplus \varphi(M'))$. Our collision search is now a search for M such that the first $s/2$ bits of $\mathcal{H} \times (\varphi(M) \oplus \varphi(M'))$ are zero.

In practice, the smallest period we can achieve with the regular encoder is n/w, so we will require that $n/2w$ is a divisor of r. We divide the encoded word into block of size r, and we choose the same m_i for all the chunk of the message that are used in the same block. The choice of a message M so that $\varphi(M)$ is piecewise n/w-periodic is equivalent to choice of n/r values $\mu_i \in \{0..n/w-1\}$ such that $m_i = \mu_{\lfloor i/\frac{rw}{n} \rfloor}$. Then:

$$F(M) = F(\mu_0, \mu_1, ..\mu_{n/w-1}) = \bigoplus_{i=0}^{n/r-1} \mathcal{H} \times \theta_i(\mu_i)$$

$$\theta_i(\mu_i) = \varphi_{i\frac{rw}{n}}(\mu_i) \oplus \varphi_{i\frac{rw}{n}+1}(\mu_i) \oplus \cdots \varphi_{(i+1)\frac{rw}{n}-1}(\mu_i)$$

M' can be constructed easily: due to the definition of the regular encoder, we just have to set $\mu'_i = \mu_i + n/2w \pmod{n/w}$. We have now reduced the collision search to the search of $\mu_0, ...\mu_{n/r-1}$ such that:

$$F(M) \oplus F(M') = \bigoplus_{i=0}^{n/r-1} \left(\mathcal{H} \times \theta_i(\mu_i) \oplus \mathcal{H} \times \theta_i(\mu_i + n/2w)\right) = 0.$$

but we know that $F(M) \oplus F(M')$ is $n/2w$ periodic, so we only need to cancel $n/2w$ bits.

This is an instance of the n/r-sum problem on $n/2w$ bit with lists of $n/2w$ elements, which can be solved with the same methods as the original one (which was an instance of the $2w$-sum problem on r bits, with lists of n/w elements):

- The linearization attack can be used if $n/r \geq n/2w$, which is equivalent to the condition on the original system: $r \leq 2w$. The complexity will drop from r^3 to $(n/2w)^3$.
- For Wagner's attack, let a_1 be the best a for the original problem and a_2 the best a for the new system. They have to satisfy:

$$\frac{2^a}{a+1} \leq \frac{2w}{r}\log(n/w) \qquad\qquad \frac{2^{a'}}{a'+1} \leq \frac{2w}{r}(\log(n/w) - 1)$$

  In most cases, we will be able the use the same a on the new system, and the complexity of the attack drops from $r2^a \cdot 2^{r/(a+1)}$ (which was used as a security parameter) to $r2^a \cdot 2^{\frac{n}{2w}/(a+1)}$. Since $n/2w$ is usually much smaller than r, this can already break all proposed parameters of FSB, many of them in practical time!

If the we are looking for collision with the same chaining value, instead of pseudo-collision, we will only have $n/r - n/s$ lists instead of n/r.

The next section will introduce a new way to solve this system, which is even more efficient.

4.3 A system of cyclic equations

The collision attack on IFSB has now been reduced to a collision attack on much smaller bit-strings. But the small bit-strings still have a strong structure: we have $\varphi_i(x+1) = \varphi_i(x) \lll 1$

because $\mathcal{H}$ is quasi-cyclic and similarly $\theta_i(x + 1) = \theta_i(x) \lll 1$. Therefore, each list actually contains every rotations of a single vector. We can write this as a system of equations: we are given p bit-strings of length 2^l $x_0, x_1, ...x_{p-1}$, and we look for $r_0, r_1, ...r_{p-1} \in \{0..2^l - 1\}$ such that

$$\bigoplus_{i=0}^{p-1} x_i \lll r_i = 0.$$

First of all, if the sum of all bits of the x_i is non-zero (*ie.* $\bigoplus_{k,i} x_k^{[i]} = 1$), there is no solution; in this case we will drop one of the x_i with an odd bit-sum, and set $\mu_i' = \mu_i$ for this particular i. We now assume that $\bigoplus_{k,i} x_k^{[i]} = 0$.

We can use Wagner's algorithm to solve this system, but we will present a more efficient solution when $n/2w$ is a power of two (wich is always the case to ease the implementation). To describe our algorithm, we will use a set of function π_i which folds 2^l-bit strings into 2^i-bit strings: $\pi_i(x)$ cuts x into chunks of 2^i bits, and xors them together (π_l is the identity function). We also use $\pi_i^L(x)$ which is the left part of $\pi_i(x)$, and $\pi_i^R(x)$ is the right part (therefore $\pi_{i-1}(x) = \pi_i^L(x) \oplus \pi_i^R(x)$). The algorithm is described by Algorithm 1; it uses linear algebra in a similar way as the attack of section 2.3.

Algorithm 1 Cyclic system solver

1: **for all** r_k **do**
2: $\quad r_k \leftarrow 0$
3: **for** $1 \leq i < l$ **do**
4: $\quad \Delta \leftarrow [\pi_i(x_k \lll r_k)]_{k=0}^{p-1}$
5: $\quad$ Set u as the solution to $\Delta \times u = \pi_{i+1}^L(\oplus x_k \lll r_k)$
6: $\quad$ **for all** r_k **do**
7: $\quad\quad r_k \leftarrow r_k + u^{[k]} 2^i$

Proof. The proof of Algorithm 1 uses the fact that after iteration i we have $\pi_{i+1}(\bigoplus x_k \lll r_k) = 0$. If r_k are the values at the *beginning* of iteration i, we have $\pi_i(\bigoplus x_k \lll r_k) = 0$ and:

$$\begin{aligned}
L &= \pi_{i+1}^L\left(\bigoplus x_k \lll (r_k + u^{[k]} 2^i)\right) \\
&= \pi_{i+1}^L\left(\bigoplus x_k \lll r_k \oplus \bigoplus_{u^{[k]}=1} \left(x_k \lll r_k \oplus x_k \lll (r_k + 2^i)\right)\right) \\
&= \pi_{i+1}^L\left(\bigoplus x_k \lll r_k\right) \oplus \bigoplus_{u^{[k]}=1} \pi_{i+1}^L(x_k \lll r_k) \oplus \pi_{i+1}^R(x_k \lll r_k) \\
&= \pi_{i+1}^L\left(\bigoplus x_k \lll r_k\right) \oplus \bigoplus_{u^{[k]}=1} \pi_i(x_k \lll r_k) \\
&= \pi_{i+1}^L\left(\bigoplus x_k \lll r_k\right) \oplus \Delta \times u = 0 \qquad \text{(By construction of } u\text{)}
\end{aligned}$$

$$
\begin{aligned}
R &= \pi_{i+1}^{R}\left(\bigoplus x_k \lll (r_k + u^{[k]}2^i)\right) \\
&= \pi_{i+1}^{L}\left(\bigoplus x_k \lll (r_k + u^{[k]}2^i)\right) \oplus \pi_i\left(\bigoplus x_k \lll (r_k + u^{[k]}2^i)\right) \\
&= 0 \oplus \pi_i\left(\bigoplus x_k \lll r_k\right) \qquad \text{(Because } \pi_i(x \lll 2^i) = \pi_i(x)\text{)} \\
&= 0
\end{aligned}
$$

Therefore $\pi_{i+1}(\bigoplus x_k \lll r_k) = L||R = 0$ at the *end* of the iteration (with the new r_k). After the last iteration, this reads $\bigoplus x_k \lll r_k = 0$. □

Complexity Analysis. The complexity of the attack is very low: the only computational intensive step is the linear algebra. Using a simple Gaussian elimination, we can solve the equation $\Delta \times u = c$ where Δ has 2^i rows and p columns in time $p2^{2i}$. The time of the full algorithm is therefore $\sum_{i=1}^{l-1} p2^{2i} = \frac{4^l-4}{3}p$.

The success probability of the algorithm is related to the probability $C(n,p)$ that a random system $\Delta \times u = c$ is consistent, when Δ has n rows and p columns. See Appendix A for an analysis of C. If all the systems are independent, identically distributed, the success probability can be expressed as:

$$P(n,p) = \prod_{i=1}^{\log(n)-1} C(2^i, p).$$

Actually, the systems are not independent and identically distributed because the random bits the x_k's are aligned in a particular manned by the r_k of the previous step; in particular we have an obvious relation at each step: $\sum \Delta_i = 0$. We will assume that this is the only difference between the algorithm and the resolution of independent systems, which means the probability of success is $P(2^l, p-1)$. However, to solve a cyclic system, we first have to make sure that $\bigoplus_{k,i} x_k^{[i]} = 0$; this means that for one system out of two we have to run the algorithm with $p-1$ bit-strings. In the end, the expected success probability is

$$\Pi(2^l, p) = \frac{P(2^l, p-1) + P(2^l, p-2)}{2}$$

and the expected running time is:

$$\Pi(2^l, p)^{-1} \cdot \frac{4^l - 4}{3}p.$$

To check the hypothesis of independence, we ran the cyclic solver on random systems, and compared the success rate to the theoretical value. Figure 2 show that we are very close to the actual success probability.

When we use the cyclic solver to break IFSB, we will have $l = \log(n/2w)$ and $p = n/r$ for pseudo-collision, $p = n/r - n/s$ for collisions. We will analyse the pseudo-collision case in more detail:

1. If $p \geq 2^{l-1}$, *ie* $r \leq 4w$: we will use p a little larger than 2^{l-1} so that $\Pi(2^l, p)$ is almost 1. In this case, the running time of the full algorithm of the algorithm is essentially:
$$T = \frac{4^l-4}{3}p < 2(n/4w)^3.$$
2. If $p < 2^{l-1}$, *ie* $r > 4w$, we cannot have a good probability of success and we have to repeat the cyclic solver with a randomized system. When we want to find collisions in a given hash function, we have only one particular system, but we can still apply a random rotation to

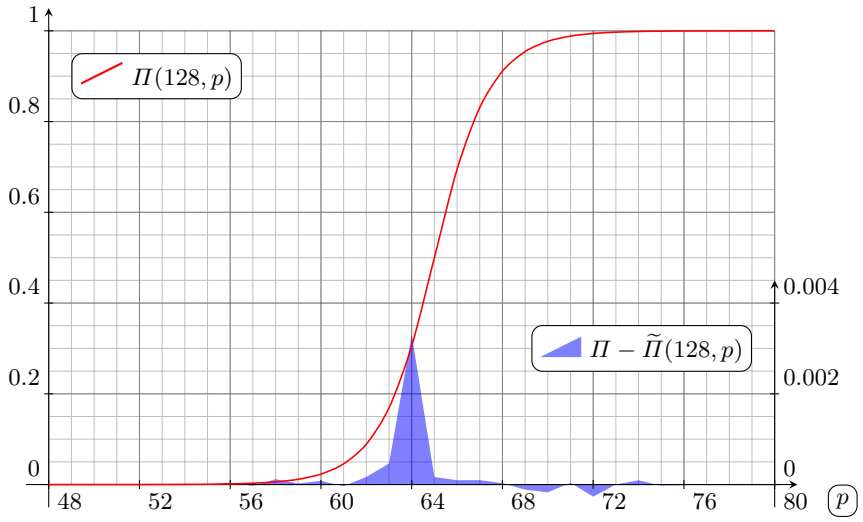

Fig. 2. Theoretical success probability Π versus experimental success probability $\widetilde{\Pi}$. Since the two curves would be on top of each over, we draw the difference between them on a different scale. $\widetilde{\Pi}$ is measured by running the algorithm 2^{20} times with randon systems of the given size.

each word. This will not give an independent system, but since our algorithm only find some special solutions to the cyclic system (the lower bits of the r_k's are mostly zeros) this randomization could sufficient. Experimentally, with $l = 6$ and $p = 53$ (used for a collision attack against line 4 of Table 1) it works very well.
In this case the running time of the full algorithm of the algorithm is essentially:

$$T = 2^{p-2^{l-1}\frac{4^l-4}{3}}\, p < 2^{\frac{n(r-4w)}{4wr}} \cdot 2(n/4w)^3.$$

Example. To illustrate the attack, let us consider the most interesting setting of the table 1 of [5], designed for 128-bit security. We have:

$$r = 1024 \qquad w = 1024 \qquad s = 8192 \qquad n/w = 256$$

For a collision attack, we can build a cyclic to build a system with $l = n/2w = 128$ and $p = n/r - n/s = 224$. The main operation to solve this system will be linear algebra on a matrix of size 64... this only costs 2^{18} elementary operations (xors on one bit). Since the compression function requires $rw = 2^{20}$ elementary operations, our attack costs less than one call to the compression function!

Example 2. An other interesting set of parameter that we can break are the recommended parameters for memory constrained environments. This function was believed to provide a security of 2^{80}, but we can break it *by hand*! Since we have $n/4w = 2$ the resolution of the cyclic system is almost trivial and the most expensive step of the attack is the construction of the system...

Table 1 gives an overview of the various parameters of IFSB and the complexity of the linearization attack and the cyclic attack (for pseudo-collisions and collisions). The first part of the table presents the parameters used for performance evaluation in [5]. These parameters use r and n/w are powers of two, which allow our new attack to be used. We can see that it has a

lower complexity than the linear attack, especially for the 80 bit security parameters, which have $r > 2w$. The next parts of the table show the parameters recommended by [5]. The recommended parameter set for standard applications has a prime r, which make our attack unusable, but the parameter set for memory constrained environments use a power of two. In the last part of the table, we show the parameters of FSB. They have not been proposed for use with IFSB, but we feel that it would be a natural move to prevent the attacks which need a low r/w ratio.

Table 1. Comparison of the linearization attack and the cyclic attack on the various parameter sets proposed in [5]. The complexity is given in elementary operations; one compression function costs rw elementary operations (typically 2^{20}).

						Linear		Cyclic				
r	w	n	s	n/w	secu.	psd.	coll	psd.	coll	$n/4w$	n/r	n/s
512	512	131072	4096	256	64	2^{27}	2^{27}	2^{19}	2^{19}	64	256	32
512	450	230400	4050	512		2^{27}	2^{27}	2^{22}	2^{22}	128	450	32.2
1024	2^{17}	2^{25}	2^{20}	256		2^{30}	2^{30}	2^{19}	2^{19}	64	2^{15}	32
512	170	43520	1360	256	80	2^{100}	-	2^{19}	2^{30}	64	85	32
512	144	73728	1296	512		-	-	2^{22}	2^{63}	128	144	56.9
1024	1024	262144	8192	256	128	2^{30}	2^{30}	2^{19}	2^{19}	64	256	32
1024	904	462848	8136	512		2^{30}	2^{30}	2^{22}	2^{22}	128	452	56.5
1024	816	835584	8160	1024		2^{30}	2^{30}	2^{25}	2^{25}	256	816	102.4
Recommended parameters for standard applications:												
1061	1024	262144	8192	256	128	2^{30}	2^{30}	-	-			
Recommended parameters for memory constrained environments:												
512	320	2560	1280	8	80	2^{27}	2^{61}	2^{11}	2^{11}	2	5	2
Parameters of FSB [1]. Not proposed for IFSB, but could be a way to repair it:												
480	170	43520	1360	256	80	2^{85}	-	2^{19}	2^{25}	64	90.7	32
400	85	21760	680	256		-	-	2^{29}	2^{61}	64	54.4	32
320	42	10752	336	256		-	-	2^{50}	-	64	33.6	32

4.4 About the optimal encoder

The optimal encoder allows to create messages such that $\varphi(M)$ is concentrated on one cyclic block of $\mathcal{H}$. Since (almost) any word of weight w is in the range of φ, if $w \geq r$ we can even choose a piecewise 1-periodic $\varphi(M)$! In this case, we have a very easy pseudo-collision attack:

1. Consider the messages $M_k = \varphi^{-1}\left(0^{kr}1^r0^{n-(k+1)r}1^{w-r}\right)$
2. We have $F(M_k) = \mathcal{H} \times \varphi(M_k) = s_k \oplus t$, where:
   $s_k = \mathcal{H}_k \times 1^r$ is 1-periodic
   $t = \mathcal{H} \times 0^{n-w+r}1^{w-r}$
3. Since s_k is 1-periodic, it can only take two values: 0^r and 1^r: we have at least one collision between M_0, M_1 and M_2.

If the optimal encoder is combined with a quasi-cyclic matrix of non prime length, it is easy to build periodic messages with a very small period. Because of this, we strongly discourage the use of the optimal encoder with quasi-cyclic codes.

Acknowledgement

Part of this work is supported by the Commission of the European Communities through the IST program under contract IST-2002-507932 ECRYPT, and by the French government through the Saphir RNRT project.

References

1. Augot, D., Finiasz, M., Sendrier, N.: A family of fast syndrome based cryptographic hash functions. In Dawson, E., Vaudenay, S., eds.: Mycrypt. Volume 3715 of Lecture Notes in Computer Science., Springer (2005) 64–83
2. Bellare, M., Micciancio, D.: A new paradigm for collision-free hashing: Incrementality at reduced cost. In: EUROCRYPT. (1997) 163–192
3. Coron, J.S., Joux, A.: Cryptanalysis of a provably secure cryptographic hash function. Cryptology ePrint Archive, Report 2004/013 (2004) `http://eprint.iacr.org/`.
4. Finch, S.R. In: Mathematical Constants. Cambridge University Press (2003) 354–361
5. Finiasz, M., Gaborit, P., Sendrier, N.: Improved fast syndrome based cryptographic hash functions. In Rijmen, V., ed.: ECRYPT Hash Workshop 2007. (2007)
6. Saarinen, M.J.O.: Linearization attacks against syndrome based hashes. Cryptology ePrint Archive, Report 2007/295 (2007) `http://eprint.iacr.org/`.
7. Wagner, D.: A generalized birthday problem. In Yung, M., ed.: CRYPTO. Volume 2442 of Lecture Notes in Computer Science., Springer (2002) 288–303
8. Wang, X., Yin, Y.L., Yu, H.: Finding Collisions in the Full SHA-1. In Shoup, V., ed.: CRYPTO. Volume 3621 of Lecture Notes in Computer Science., Springer (2005) 17–36
9. Wang, X., Yu, H.: How to Break MD5 and Other Hash Functions. In Cramer, R., ed.: EUROCRYPT. Volume 3494 of Lecture Notes in Computer Science., Springer (2005) 19–35

A Probability of Solving a Random Linear System

In this appendix we study the probability that the equation $\Delta \times x = c$ has at least one solution in x, given a random $n \times p$ binary matrix Δ, and a random vector c. We call this probability $C(n,p)$.

$$\begin{aligned}
C(n,p) &= \Pr_{\Delta,c}\left[\exists x : \Delta \times x = c\right] \\
&= \Pr_{\Delta,c}\left[c \in \operatorname{Im}\Delta\right] \\
&= \sum_{r=0}^{n} \Pr_{c}\left[c \in \operatorname{Im}\Delta \middle| \operatorname{rank}(\Delta) = r\right] \cdot \Pr_{\Delta}\left[\operatorname{rank}(\Delta) = r\right] \\
&= \sum_{r=0}^{n} 2^{r-n} \cdot \Pr_{\Delta}\left[\operatorname{rank}(\Delta) = r\right] \\
&= 2^{-np} \sum_{r=0}^{n} 2^{r-n} \cdot \prod_{i=0}^{p-r-1} \frac{2^p - 2^i}{2^{p-r} - 2^i} \prod_{i=0}^{r-1} 2^n - 2^i
\end{aligned}$$

The case $p > n$. The following lower bound is true for all p, but is mostly useful in the case $p > n$, and very tight when $p \gg n$:

$$\begin{aligned}
C(n,p) &\geq \Pr_{\Delta}\left[\operatorname{rank}(\Delta) = n\right] = 1 - \Pr_{\Delta}\left[\operatorname{rank}(\Delta) < n\right] \\
&\geq 1 - \sum_{H \text{hyperplan}} \Pr_{\Delta}\left[\operatorname{Im}\Delta \subset H\right] \\
&\geq 1 - 2^n \frac{2^{(n-1)p}}{2^{np}} = 1 - 2^{n-p}
\end{aligned}$$

It shows that we just have to choose p a little bigger than n to get a very high probability of success.

The case $p < n$. When $p < n$, we have

$$C(n,p) \geq 2^{p-n} \cdot \Pr_{\Delta}\left[\operatorname{rank}(\Delta) = p\right] = 2^{p-n} \prod_{i=0}^{p-1} 1 - 2^{i-n}$$

When $p \ll n$ The quantity $Q(n,p) = \prod_{i=0}^{p-1} 1 - 2^{i-n}$ is very close to one and the bound is very tight, but we can derive a lower bound it as long as $p < n$:

$$Q(n,p) \geq \prod_{k=1}^{\infty} 1 - 2^{-k} = 0.288788... \qquad \text{(see [4])}$$

This allows us to say that the probability of success of the algorithm when $p < n$ is about 2^{p-n}.