Cryptanalysis of a Hash Function Proposed at ICISC 2006

Willi Geiselmann[1] and Rainer Steinwandt[2]

[1] Institut für Algorithmen und Kognitive Systeme, Fakultät für Informatik, Universität Karlsruhe (TH), Am Fasanengarten 5, 76128 Karlsruhe, Germany, `geiselma@ira.uka.de`

[2] Department of Mathematical Sciences, Florida Atlantic University, 777 Glades Road, Boca Raton, FL 33431, USA, `rsteinwa@fau.edu`

Abstract. A simple method for constructing collisions for Shpilrain's polynomial-based hash function from ICISC 2006 is presented. The attack relies on elementary linear algebra and can be considered as practical: For the parameters suggested, we give a specific collision, computed by means of a computer algebra system.

Keywords: cryptanalysis, hash function

1 Introduction

In [Shp06] Shpilrain proposes a hash function H which builds on the Merkle-Damgård construction [Dam90,Mer90] and relies on computations in the quotient of a polynomial ring. In [Cha06] Chang reports that the underlying compression function is easy to invert and that a meet-in-the-middle attack enables a preimage attack on H. According to Chang's complexity estimate, for the specific parameters proposed in [Shp06] the computational effort for mounting such a preimage attack appears to be in the magnitude of 2^{80} operations.

The collision attack we describe below can be considered as practical—for the specific parameters proposed in [Shp06] we give a collision of two equal length bitstrings with about 10.2 KByte each. Shpilrain's proposed hash function H does not involve padding, but the collision given below remains valid if the usual Merkle-Damgård strengthening is applied to H.

2 The proposal from ICISC 2006

Let $p(x) \in \mathbb{F}_2[x]$ be a univariate polynomial of degree n over the finite field with two elements. Moreover, let α be the residue class of x in the quotient $R := \mathbb{F}_2[x]/(p(x))$, thus $p(\alpha) = 0$. We remark that [Shp06] writes "$R = \mathbb{F}_{2^n} = \mathbb{F}_2[x]/(p(x))$" which suggests $p(x)$ to be irreducible, but the specific polynomial $p(x)$ proposed is reducible.

2.1 General construction

To define the hash function H, two elements $h_0, h_1 \in R$ are fixed, and the hash value of an individual bit is defined as

$$\begin{aligned} H(0) &:= h_0 \\ H(1) &:= h_1 \end{aligned} \tag{1}$$

Next, a triple $(u_0, u_1, u_2) \in R^3$ is used to fix a binary operation $\circ$ on R:

$$\begin{aligned} \circ : \quad R^2 &\longrightarrow R \\ (r_1, r_2) &\longmapsto r_1 \circ r_2 := u_0 + r_1 \cdot r_2 + r_1^2 \cdot u_1 + r_2^2 \cdot u_2 \end{aligned} \tag{2}$$

To hash a bitstring M, the following procedure is used:

1. Going from left to right, the bitstring M is split into 32-bit blocks $M = B_1 \parallel B_2 \cdots \parallel B_\ell$, where the last block B_ℓ has less than 32 bit, if the length of M is not a multiple of 32. There is no padding.
2. The hash value of each single 32-bit block $B_i = B_{i,0} \parallel \cdots \parallel B_{i,31}$ is computed by applying the above operation $\circ$ one bit at a time, going from left to right:

$$H(B_i) := (\ldots((H(B_{i,0}) \circ H(B_{i,1})) \circ H(B_{i,2}))\ldots) \circ H(B_{i,31})$$

   (where the hash value $H(B_{i,j})$ of a single bit $B_{i,j}$ is given by (1)).
3. The hash value $H(M)$ of M is computed by applying the operation $\circ$ one block at a time, going from left to right:

$$H(M) := (\ldots((H(B_0) \circ H(B_1)) \circ H(B_2))\ldots) \circ H(B_\ell)$$

The value $H(M)$ is the output of the hash function for input M.

2.2 Suggested parameters

As specific parameter choice, [Shp06] suggests the following:

$$\begin{aligned} p(x) &:= x^{163} + x^7 + x^6 + x^5 + x^4 + x + 1 \\ h_0 &:= \alpha^7 + 1 \\ h_1 &:= \alpha^8 + 1 \\ (u_0, u_1, u_2) &:= (1, \alpha^2, \alpha) \end{aligned}$$

To demonstrate the practicality of the attack proposed below, in Section 3.3 we construct a specific collision for this parameter choice.

3 Finding collisions

As already indicated above, the notation "$R = \mathbb{F}_{2^n} = \mathbb{F}_2[x]/(p(x))$" in [Shp06] suggests the considered polynomial $p(x)$ to be irreducible. However, with a computer algebra system like Magma [BCP97] one easily checks that the proposed polynomial splits into four irreducible factors from $\mathbb{F}_2[x]$. Namely, for $p(x) = x^{163} + x^7 + x^6 + x^5 + x^4 + x + 1$ we have $p(x) = q_1(x) \cdot q_2(x) \cdot q_3(x) \cdot q_4(x)$, where

$$\begin{aligned}
q_1(x) &:= x^9 + x^7 + x^5 + x + 1\\
q_2(x) &:= x^{18} + x^{14} + x^{12} + x^{11} + x^6 + x^4 + 1\\
q_3(x) &:= x^{38} + x^{36} + x^{33} + x^{31} + x^{30} + x^{28} + x^{24} + x^{22} + x^{21} + x^{20} + x^{19}\\
&\quad + x^{17} + x^{16} + x^{12} + x^{10} + x^8 + x^7 + x^4 + x^3 + x^2 + 1\\
q_4(x) &:= x^{98} + x^{94} + x^{93} + x^{91} + x^{90} + x^{88} + x^{87} + x^{84} + x^{82} + x^{73} + x^{69}\\
&\quad + x^{68} + x^{67} + x^{65} + x^{64} + x^{61} + x^{58} + x^{55} + x^{54} + x^{53} + x^{46}\\
&\quad + x^{45} + x^{44} + x^{43} + x^{42} + x^{41} + x^{39} + x^{37} + x^{31} + x^{29} + x^{28}\\
&\quad + x^{26} + x^{25} + x^{24} + x^{20} + x^{18} + x^{17} + x^{14} + x^{13} + x^9 + x^8 + x^7\\
&\quad + x^6 + x^5 + x^3 + x^2 + 1
\end{aligned}$$

Thus, before discussing the core part of our attack, it is worth discussing briefly how to exploit such a factorization for a collision search.

3.1 Using the Chinese Remainder Theorem

According to the Chinese Remainder Theorem, any factorization of the polynomial $p(x)$ into coprime factors $q_1(x) \ldots, q_s(x)$ yields a decomposition of the ring $R = \mathbb{F}_2[x]/(p(x))$ into a direct product of rings $R_i := \mathbb{F}_2[x]/(q_i(x))$:

$$R \simeq R_1 \times \cdots \times R_s$$

As the hash function H composes the hash values of the individual 32-bit blocks with simple ring operations, it looks tempting to exploit this isomorphism of rings to perform the collision search "one R_i at a time". Suppose we have found two bitstrings M_1, M_2 whose lengths are multiples of 32 and which satisfy

$$H(M_1) \equiv H(M_2) \pmod{q_s(x)} \quad ,$$

i. e., we have a collision in the R_s-component. Owing to the Merkle-Damgård structure of H, we then have

$$H(M_1 \parallel T) \equiv H(M_2 \parallel T) \pmod{q_s(x)}$$

for arbitrary bitstrings T appended to M_1 and M_2. Thus, if we heuristically (though actually incorrectly) take the values $H(M_1 \parallel T)$ and $H(M_2 \parallel T)$ as being uniformly and independently distributed modulo $q_{s-1}(x)$, we would expect that within $\mathrm{O}(2^{\deg(q_{s-1}(x))})$ random attempts for T, we encounter a pair of messages $M_1 \parallel T_{s-1}$, $M_2 \parallel T_{s-1}$ whose hash values coincide in the $R_{s-1} \times R_s$-component of R. If the degree of q_{s-1} is small, this approach can be efficient enough. In our experiments we used the linear algebra technique described in the next section to reduce the computational effort for finding a matching T_{s-1}.

Now assume we have found a matching "tail" T_{s-1} and that the length of T_{s-1} is a multiple of 32. Then we can apply the same reasoning as before to extend the collision

$$H(M_1 \parallel T_{s-1}) \equiv H(M_2 \parallel T_{s-1}) \pmod{q_{s-1}(x) \cdot q_s(x)}$$

from $R_{s-1} \times R_s$ to $R_{s-2} \times R_{s-1} \times R_s$: Analogously as before, now we test bitstrings T_{s-2} until

$$H(M_1 \parallel T_{s-1} \parallel T_{s-2}) \equiv H(M_2 \parallel T_{s-1} \parallel T_{s-2}) \pmod{q_{s-2} \cdot q_{s-1} \cdot q_s}$$

holds. In this way, we can process the components $R_s, R_{s-1}, \ldots, R_1$ one by one, starting from a collision in a single component.

Example 1. For the specific parameters from Section 2.2 we have $s = 4$, and the degrees of $q_1(x)$, $q_2(x)$ and $q_3(x)$ are rather small—namely 9, 18 and 38. Thus, once we know a pair of messages colliding in the larger R_4-component (of size 2^{98}), deriving a full collision that is valid in R should be straightforward. Indeed, in our actual computations this worked as expected.

3.2 Using linear algebra

In view of the above discussion, the parameter choice in [Shp06] does not seem to offer an adequate security level, and constructing a collision in the component R_4 (of size 2^{98}) seems to be the most time-consuming task for mounting such an attack. In this section we show that such a collision can be found easily, without implementing a full birthday attack in R_4.

Remark 1. We describe the attack for an irreducible polynomial $p(x)$ of degree n, i.e., for $R \simeq \mathbb{F}_{2^n}$. For the specific parameter set from Section 2.2, this linear algebra based part is exploited for R_4 and R_3 only, but the attack technique as such does not rely on the described short-cut via the Chinese Remainder Theorem. In particular, simply imposing

$p(x)$ to be irreducible of degree 163 does not appear to be an adequate countermeasure to rule out the attack.

Let $R' \subseteq R$ be the image of H when being restricted to messages whose length is a multiple of 32 (i. e., we have no incomplete last blocks). To each 32-bit block B, we can assign the following map ϕ_B, which captures the update of H's internal state when appending B to a message whose length is a multiple of 32.

$$\begin{aligned} \phi_B : R' &\longrightarrow R' \\ h &\longmapsto h \circ H(B) \end{aligned}$$

The map ϕ_B, is affine in the sense that it splits into the sum of the $\mathbb{F}_2$-linear map $h \mapsto h \cdot H(B) + h^2 \cdot u_1$ and the constant shift $H(B)^2 \cdot u_2 + u_0$. If we consider a sequence of blocks $B_1, \dots, B_t$, then the composition

$$\phi_{B_1\|B_2\|\cdots\|B_t} := \phi_{B_t}(\phi_{B_{t-1}}(\dots \phi_{B_1}(h))\dots)$$

computes the hash value obtained by appending $B_1 \| B_2 \| \cdots \| B_t$ to a preimage of $h \in R'$. As each of the ϕ_{B_i} is affine in the sense just described, the same holds for $\phi_{B_1\|B_2\|\cdots\|B_t}$—with the constant shift depending on $B_1, \dots, B_t$. The linear part of $\phi_{B_1\|B_2\|\cdots\|B_t}$ is just the functional composition of the linear parts of the ϕ_{B_i}s.

Once we know a sequence of 32-bit blocks $B_1, \dots, B_t$ and two different values $h_1, h_2 \in R'$ with

$$\phi_{B_1\|B_2\|\cdots\|B_t}(h_1) = \phi_{B_1\|B_2\|\cdots\|B_t}(h_2) \quad ,$$

or equivalently

$$\phi_{B_1\|B_2\|\cdots\|B_t}(h_1) + \phi_{B_1\|B_2\|\cdots\|B_t}(h_2) = 0 \quad , \tag{3}$$

we have a collision for H—provided we know preimages of h_1 and h_2. As the left-hand side of Equation (3) is $\mathbb{F}_2$-linear in $h_1 + h_2$—the constant shifts cancel out in the summation—we can rewrite (3) in the form

$$\overline{(h_1 + h_2)} \cdot \mathcal{M}_{B_1\|B_2\|\cdots\|B_t} = 0 \quad .$$

Here $\mathcal{M}_{B_1\|B_2\|\cdots\|B_t}$ is an $n \times n$ matrix over $\mathbb{F}_2$, and $\overline{(h_1 + h_2)} \in \mathbb{F}_2^n$ is comprised of the coefficients of $h_1 + h_2$ when being expressed in the appropriate $\mathbb{F}_2$-vector space basis. Now, if we can find $B_1, \dots B_t$ such that $\mathcal{M}_{B_1\|B_2\|\cdots\|B_t}$ is of low rank (i. e., has a large kernel) we can simply try to choose messages $M_1 \neq M_2$ at random until the sum of their hash values $(H(M_1) + H(M_2))$ yields a vector $\overline{(H(M_1) + H(M_2))}$ in the kernel of $\mathcal{M}_{B_1\|B_2\|\cdots\|B_t}$.

Remark 2. It is worth noting that there is no particular requirement on the messages M_1, M_2. This seems a useful feature when aiming at meaningful collisions: Suppose we have a message/file format of interest, where it is possible to append "garbage" at the end of a valid message (up to some fixed end-of-message delimiter).

Then we could fix two meaningful messages M_1', M_2' which we want to collide and choose our candidates as $M_1 := M_1' \parallel N_1$, $M_2 := M_2' \parallel N_2$ with random bitstrings N_1, N_2. The final colliding messages then had the form

$$M_1 = M_1' \parallel N_1 \parallel B_1 \parallel \cdots \parallel B_t \parallel E$$
$$M_2 = M_2' \parallel N_2 \parallel B_1 \parallel \cdots \parallel B_t \parallel E$$

where E can be a message-independent (possibly empty) end-of-message delimiter.

Expediting the computation of a kernel element In our experiments with the parameters from Section 2.2, finding a small, say $\approx$16, number t of blocks $B_1, \ldots, B_t$ such that $\mathcal{M}_{B_1\|B_2\|\cdots\|B_t}$ has a rank defect of $\approx t$ required no particular effort. Already a trivial enumeration of some 32-bit blocks B_1 quickly yields a candidate where choosing all t blocks equal to B_1 results in a matrix $\mathcal{M}_{B_1\|B_1\|\cdots\|B_1}$ with rank defect t. For larger rank defects, however, the heuristics we used required a significantly larger number of blocks (see below). Aiming at collisions of moderate length, it seems worthwhile to improve the simple guessing strategy for finding kernel elements:

Suppose our $n \times n$ matrix $\mathcal{M}_{B_1\|B_2\|\cdots\|B_t}$ over $\mathbb{F}_2$ has rank defect d. Taking the candidate vectors $\overline{(H(M_1) + H(M_2))}$ for independently and uniformly at random chosen elements from $\mathbb{F}_2^n$, we could expect that after $\mathrm{O}(2^{n-d})$ attempts a kernel vector is found. If we do not mandate M_1 and M_2 to have a particular form, we can easily improve on this as follows:

1. Using a computer algebra system, we can easily find a vector space basis of the (d-dimensional) kernel $\ker(\mathcal{M}_{B_1\|B_2\|\cdots\|B_t})$ of $\mathcal{M}_{B_1\|B_2\|\cdots\|B_t}$.
2. Using a birthday attack we search for messages M_1, M_2 such that the projections of $\overline{(H(M_1))}$, $\overline{(H(M_2))}$ on $\ker(\mathcal{M}_{B_1\|B_2\|\cdots\|B_t})$ coincide. In other words we want $\overline{(H(M_1))}$ and $\overline{(H(M_2))}$ to be in the same residue class of $\mathbb{F}_2^n / \ker(\mathcal{M}_{B_1\|B_2\|\cdots\|B_t})$. Then

$$\overline{(H(M_1))} + \overline{(H(M_2))} = \overline{(H(M_1) + H(M_2))} \in \ker(\mathcal{M}_{B_1\|B_2\|\cdots\|B_t})$$

as desired.

Taking the $\overline{(H(M_i))}$ for independently and uniformly at random chosen elements from $\mathbb{F}_2^n$, we expect to find the desired messages M_1 and M_2 after $\mathrm{O}(2^{(n-d)/2})$ attempts.

Example 2. For Shpilrain's specific parameter proposal (see Section 2.2), in the largest component obtained from the Chinese Remainder Theorem, we have $n = 98$. Here we used a matrix with a rank defect of $d = 42$, constructed from $t = 2882$ blocks B_i.

Finding a low rank matrix By construction, we have

$$\mathcal{M}_{B_1\|B_2\|\cdots\|B_t} = \mathcal{M}_{B_1} \cdot \mathcal{M}_{B_2} \cdot \cdots \cdot \mathcal{M}_{B_{t-1}} \cdot \mathcal{M}_{B_t},$$

with $\mathcal{M}_{B_i}$ being the $n \times n$ matrix over $\mathbb{F}_2$ representing the linear part of ϕ_{B_i}. Thus, the task of finding a matrix $\mathcal{M}_{B_1\|B_2\|\cdots\|B_t}$ of low rank reduces to finding 32-bit blocks B_i such that we can form products of the respective matrices $\mathcal{M}_{B_i}$ with the product having low rank. Also, from a practical perspective it seems desirable that the number t of blocks is not too large, so that the resulting collision fits into, say, a few KByte.

In our experiments with the parameter set from Section 2.2, simple heuristics turned out to yield adequate blocks $B_1, \ldots, B_t$, and we did not attempt a thorough theoretical analysis or optimization of the task:

- For small values of t, say $t \approx 16$, already by just enumerating some 32-bit blocks B_i we quickly obtain candidates such that t identical blocks B_i yield a matrix $\mathcal{M}_{B_i\|B_i\|\cdots\|B_i}$ with rank defect t.
- Knowing a product $\mathcal{M}_{B_1} \cdot \cdots \cdot \mathcal{M}_{B_{t'}}$ of low rank, one can try to exhaust 32-bit blocks $B_{t'+1}$ until multiplying $\mathcal{M}_{B_1} \cdot \cdots \cdot \mathcal{M}_{B_{t'}}$ with $\mathcal{M}_{B_{t'+1}}$ reduces the rank further. Experimentally, this worked nicely for up to around $t' \approx 20$ blocks.
- If we have found a small number of matrix products $\mathcal{P}_1, \ldots, \mathcal{P}_v$ with a certain rank defect, we can try to form short products of these $\mathcal{P}_i$s and hope that the multiplication reduces the rank.
  This procedure can be applied repeatedly and in our experiments worked quite nicely. The main drawback is that each $\mathcal{P}_i$ can already be derived from a number of 32-bit blocks B_j: if we form a product
  $$\mathcal{P}' := \mathcal{P}_1 \cdot \cdots \cdot \mathcal{P}_{n_1}$$
  of n_1 matrices $\mathcal{P}_i$ where each $\mathcal{P}_i$ is a product of n_2 matrices $\mathcal{M}_{B_j}$, then $\mathcal{P}'$ corresponds to $n_1 \cdot n_2$ 32-bit blocks B_j.

The next section shows that the above attack can be considered as practical: We use it to derive a collision for the parameter choice proposed in [Shp06] (see Section 2.2).

3.3 A collision for the proposed parameters

As already mentioned, the specific polynomial $p(x)$ suggested by Shpilrain in [Shp06] splits into a product $p(x) = q_1(x) \cdot q_2(x) \cdot q_3(x) \cdot q_4(x)$ as specified in Section 3. Therefore we made use of the Chinese Remainder Theorem as discussed in Section 3.1.

A collision in $\mathbb{F}_{2^{98}}$ To construct a collision in $\mathbb{F}_2[x]/(q_4(x)) \simeq \mathbb{F}_{2^{98}}$ we applied the techniques from the previous section: Using a sequence

$$T_4' := B_1 \parallel \cdots \parallel B_{2882}$$

of 2882 suitably chosen 32-bit blocks, we derived a matrix $\mathcal{M}_{B_1\parallel\cdots\parallel B_{2882}}$ of rank 56, i. e., with rank defect $d = 98 - 56 = 42$. To specify T_4', we define the following bitstrings (to be read from left to right, line by line):

$A_1 :=$ '003FF003 06B80000 06B20000 06B20000 06BA0000 06B0C000
06BA6800 06B4F400 06B6F400 06B52A00 06BB9600 06B9DC80
06BD1180 06B6AB20 06BEF3B0 06B2B470 06BDCAF0 06B11ACC
06B90F3C 06B3B432 06B49CCA 06BB6E03' $(22 \cdot 32 \text{ bit})$

$A_2 :=$ '003FF003 06A80000 06AA0000 06A50000 06A84000 06A24000
06A22000 06A16800 06AE8400 06AE1C00 06ADAE00 06A9D500
06A3B780 06AC29C0 06AD93C0 06A7E260 06A874C2 06A85DCA
06A7A3B9 06ABAF95 06A84DFD' $(21 \cdot 32 \text{ bit})$

$A_3 :=$ '003FF003 06CC0000 06C20000 06CA8000 06CF8000 06C84000
06C64000 06C17000 06CF4800 06C98400 06CB2900 06CE8080
06C79080 06C95080 06C2A948 06CBCE28 06C00214 06CC572C
06C70021' $(19 \cdot 32 \text{ bit})$

$A_4 :=$ '003FF003 06F80000 06F80000 06F80000 06F88000 06F98000
06FBA000 06F13000 06F04800 06FE9C00 06F32E00 06FEEE00
06FA9180 06F4CDC0 06F88EB0 06F0BEF0 06FE26A8 06FB3B78'
$(18 \cdot 32 \text{ bit})$

$A_5 :=$ '003FF003 00000000 00010000 00038000 0009C000 000CE000
00065000 0007D000 00033000 000D1400 00033C00 000D0900
00008080 000CD020 000A9FA0 0009EEF0 000BDE0C 000A944C
00031A4A 0007A5FE 001F97E7 004081C9 006AC9DC 008039BD
01C1E775 031A68F0 0E217B84' $(27 \cdot 32 \text{ bit})$

At this each hexadecimal digit represents a sequence of 4 bits (`'0'` – `'0000'`, `'1'` – `'0001'`,..., `'E'` – `'1110'`, `'F'` – `'1111'`). Using $A_1, \ldots, A_5$ as building blocks, we define eight more bitstrings:

$$
\begin{aligned}
A_6 &:= A_5 \parallel A_5 \parallel A_4 \parallel A_5 \parallel A_5 \parallel A_5 \parallel A_5 \parallel A_3 \parallel A_5 \quad \text{(226 blocks)}\\
A_7 &:= A_5 \parallel A_5 \parallel A_4 \parallel A_5 \parallel A_5 \parallel A_5 \parallel A_3 \parallel A_5 \quad \text{(199 blocks)}\\
A_8 &:= A_5 \parallel A_5 \parallel A_4 \parallel A_5 \parallel A_5 \parallel A_5 \parallel A_3 \parallel A_5 \parallel A_5 \quad \text{(226 blocks)}\\
A_9 &:= A_5 \parallel A_5 \parallel A_4 \parallel A_5 \parallel A_2 \parallel A_1 \parallel A_5 \parallel A_5 \quad \text{(196 blocks)}\\
A_{10} &:= A_5 \parallel A_5 \parallel A_5 \parallel A_5 \parallel A_3 \parallel A_5 \parallel A_3 \parallel A_3 \parallel A_5 \parallel A_5 \quad \text{(246 blocks)}\\
A_{11} &:= A_5 \parallel A_3 \parallel A_3 \parallel A_5 \parallel A_5 \parallel A_5 \parallel A_5 \parallel A_3 \parallel A_3 \parallel A_5 \parallel A_5\\
&\quad \text{(265 blocks)}\\
A_{12} &:= A_5 \parallel A_5 \parallel A_5 \parallel A_5 \parallel A_3 \parallel A_3 \parallel A_2 \parallel A_5 \parallel A_5 \parallel A_5 \parallel A_5\\
&\quad \text{(275 blocks)}\\
A_{13} &:= A_5 \parallel A_4 \parallel A_5 \parallel A_5 \parallel A_5 \parallel A_3 \parallel A_3 \parallel A_5 \parallel A_5 \quad \text{(218 blocks)}
\end{aligned}
$$

In terms of $A_6, \ldots, A_{13}$, the bitstring T'_4 can be described as follows:

$$T'_4 := A_{11} \parallel A_{12} \parallel A_7 \parallel A_{11} \parallel A_{13} \parallel A_{10} \parallel A_9 \parallel A_6 \parallel A_{11} \parallel A_{12} \parallel A_8 \parallel A_6$$

Next, with a birthday attack as described we found two 32-bit blocks

$$M_1 := \texttt{'2B99EF46'} \text{ and } M_2 := \texttt{'02B6CF84'}$$

with $\overline{(H(M_1) + H(M_2))}$ being in the kernel of $\mathcal{M}_{B_1 \parallel \cdots \parallel B_{2882}}$. Consequently we obtain

$$H(M_1 \parallel T'_4) \equiv H(M_2 \parallel T'_4) \pmod{q_4(x)} \quad . \tag{4}$$

Pruning $\boldsymbol{T'_4}$ Inspecting $M_1 \parallel T'_4$ and $M_2 \parallel T'_4$ more closely, it turns out that (4) remains valid, if we remove the last 300 blocks from T'_4. We write T_4 for the bitstring of length $2582 \cdot 32 = 2882 \cdot 32 - 300 \cdot 32$ resulting from pruning T'_4 accordingly. In particular, we have

$$H(M_1 \parallel T_4) \equiv H(M_2 \parallel T_4) \pmod{q_4(x)} \quad . \tag{5}$$

Applying the Chinese Remainder Theorem Next, we want to identify bitstrings T_3, T_2, T_1 such that

$$H(M_1 \parallel T_4 \parallel \cdots \parallel T_i) \equiv H(M_2 \parallel T_4 \parallel \cdots \parallel T_i) \pmod{q_1(x) \cdot \cdots \cdot q_i(x)}$$

holds for $1 \le i \le 4$.

The polynomial $q_3(x)$ is of degree 38. To extend the "$\mathbb{F}_{2^{98}}$-collision" in (5) accordingly, the linear algebra approach from before can be reused: First, we identify a short bitstring

$T_3' :=$ '003FF003 06300000 06320000 063E8000 06394000 0638C000 0639A000 063C6000 0633D000 063A3400 063DBA00 0633BC80 06395B80 0637AC40 0635AF10 0636CB38 063CF824 063EEE8C' (18 blocks)

which, when "hashing modulo $q_3(x)$", corresponds to a matrix $\mathcal{M}_{T_3'}$ of low rank. Then we enumerate short bitstrings, until a candidate

$$T_3'' := \text{'00171999'}$$

is found such that $H(M_1 \parallel T_4 \parallel T_3'') + H(M_2 \parallel T_4 \parallel T_3'') \pmod{q_3(x)}$ yields a vector in the kernel of $\mathcal{M}_{T_3'}$. Defining T_3 as $T_3 := T_3'' \parallel T_3'$, we have

$$H(M_1 \parallel T_4 \parallel T_3) \equiv H(M_2 \parallel T_4 \parallel T_3) \pmod{q_4(x) \cdot q_3(x)} \qquad (6)$$

as desired. Extending the collision in (6) to the complete quotient ring $\mathbb{F}_2[x]/(q_1(x) \cdot q_2(x) \cdot q_3(x) \cdot q_4(x))$ turns out to be straightforward: Appending one more 32-bit block

$$T_2 := \text{'0008D718'}$$

already yields the desired collision

$$H(M_1 \parallel T_4 \parallel T_3 \parallel T_2) = H(M_2 \parallel T_4 \parallel T_3 \parallel T_2) \quad .$$

Thus, we have found two different bitstrings of size $2603 \cdot 32$ bit (i. e., ≈10.2 KByte), both of which hash to the same value.

For computing this collision we used the computer algebra system Magma [BCP97] on a number of different hardware platforms. We estimate our computational effort to be in the magnitude of one CPU day on a standard PC with about 8 GByte RAM.

4 Conclusion

As explained in the above discussion and demonstrated through a specific collision, the hash function proposed in [Shp06] does not offer strong collision resistance. Consequently, for applications that rely on collision resistance, the use of this hash function does not seem to be advisable.

Acknowledgments

We would like to thank Markus Grassl, Viktória Ildikó Villányi and Kenneth Matheis for interesting discussions.

References

[BCP97] Wieb Bosma, John J. Cannon, and Catherine Playoust. The Magma Algebra System I: The User Language. *Journal of Symbolic Computation*, 24:235–265, 1997.

[Cha06] Donghoon Chang. Preimage Attack on Hashing with Polynomials proposed at ICISC'06. Cryptology ePrint Archive: Report 2006/411, 2006. Available at `http://eprint.iacr.org/2006/411`.

[Dam90] Ivan 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, 1990.

[Mer90] Ralph C. Merkle. A Certified Digital Signature. In G. Brassard, editor, *Advances in Cryptology – CRYPTO '89*, volume 435 of *Lecture Notes in Computer Science*, pages 218–238. Springer, 1990.

[Shp06] Vladimir Shpilrain. Hashing with Polynomials. In M.S. Rhee and B. Lee, editors, *Proceedings of ICISC 2006*, volume 4296 of *Lecture Notes in Computer Science*, pages 22–28. Springer, 2006.