An Automata-Theoretic Interpretation of Iterated Hash Functions - Application to Multicollisions

Kimmo Halunen[1], Juha Kortelainen[2], and Tuomas Kortelainen[3]

[1] Oulu University Secure Programming Group, Department of Electrical and Information Engineering, University of Oulu
[2] Department of Information Processing Science, University of Oulu
[3] Mathematics Division, Department of Electrical and Information Engineering, University of Oulu

Abstract. In this paper we present a new method of constructing multicollision sets for iterated hash functions. Multicollisions have been studied quite actively since Joux published an attack against iterated hash functions, which works in $O(k2^{\frac{n}{2}})$ complexity for 2^k -multicollisions. Recently Kelsey & Schneier and Aumasson have published even faster multicollision attacks, which are based on fixed points of the compression function and some assumptions on the attacker's capabilities. Our method has complexity $O(2^{\frac{n}{2}})$ for arbitrarily large multicollision sets and does not have any additional assumptions on the compression function or attacker's capabilities. The drawback of our method is the extremely long messages generated by the algorithm in the worst case. Our method is based on automata theory and, given a compression function f, we construct a finite state transducer, which realizes the iterated hash function based on f. The method for generating multicollision sets is based on well known pumping properties of these automata.

1 Introduction

Hash functions are a vital part of modern information technology. These functions are needed especially in cryptography, where hash functions are used in digital signature schemes, pseudorandom number generation, digital timestamping and for message authentication.

Iterated hash functions proposed by Merkle [11] and Damgård [4] have been the method of constructing hash functions, which has been used in all technically feasible hash function proposals. The iterative structure makes it fairly easy to hash arbitrarily long messages and if the underlying compression function is collision free, also the whole hash function is collision free [4]. Unfortunately, most of the more widespread hash functions such as MD4, MD5 and SHA-1 have been found flawed, especially against collisions [17, 16, 9, 14, 5]. These collisions are found using some weakness of the underlying compression function and thus do not disprove the theorem of Damgård.

There are, however, some results, which show that the iterative structure does not provide an ideal hash function, i.e. a random oracle, which is a theoretical tool used in many proofs of security in cryptography. Most notable result is the multicollision attack by Joux [7], which shows that finding multicollisions for iterated hash functions is much easier than for an ideal hash function. With the help of this result, Joux also disproves some folklore assumptions about using different hash functions for the same message and catenating the results. Some of these results have been also proved for more general classes of hash functions by Nandi & Stinson [12] and Hoch & Shamir [6].

There are also other multicollision methods for example by Kelsey & Schneier [8] and by Aumasson [1]. These methods have better complexity than Joux's method, but they rely on the assumption that the compression function admits easily found *fixed points* and Aumasson's method restricts the use of the inital value of the hash function.

In this paper, we give an automata-theoretic interpretation of iterated hash functions. Suppose that a compression function $f : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n$ of block size m and length n (where $m > n$) is given. To implement an iterated hash function based on f, we construct a deterministic finite state transducer with approx. 2^{m+n} states, which

(i) takes as input any message $x = x_1 x_2 \cdots x_l$ such that l is a positive integer and x_i is a message block of length m over the binary alphabet for each $i \in \{1, 2, \ldots l\}$;

(ii) calculates the hash value $H_f(h_0, x)$ where h_0 is the initial value; and

(iii) outputs the hash value $H_f(h_0, x)$ on the output tape.

Modeling iterated hash functions in terms of transducers with a finite (albeit enormous) state set makes it possible to look at hashing as a computation, where cycles (parts of the computation starting and ending in the same state) are highly probable after certain number of iteration steps have been made. Thus, we are able to describe a fresh method for generating multicollision sets, which is based on cycles and pumping of input words. Arbitrarily large multicollision sets can be constructed with complexity $O(2^{\frac{n}{2}})$ for the iterated hash function based on $f : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n$, where f can be an arbitrary compression function, even a fixed input length random oracle (FIL-RO).

Our paper is organised in the following way. The second section introduces previous methods for generating multicollisions. The third and fourth sections give basic definitions of automata and hash functions. The fifth section shows how iterated hash functions and multicollisions can be modeled with finite state automata and transducers. In the sixth section, we introduce the multicollision generation method based on pumping of words. In the final sections, we discuss our results, draw conclusions from our research and give possible future research proposals.

2 Previous methods

In this section we present briefly the most significant previous results concerning multicollisions in iterated hash functions. First of all, a basic brute force multicollision attack against any hash function has the complexity $O(k!\frac{1}{k}2^{\frac{n(k-1)}{k}})$ for k colliding messages [15]. This was thought to be optimal also for iterated hash functions, until Joux's discovery. In the following we denote by f the compression function used in the iterated hash function.

The Joux's method [7] for generating 2^k-collisions has the complexity $O(k2^{\frac{n}{2}})$. In this method, one can begin with any initial value h_0 and one finds two message blocks x_1 and x_1' such that $f(h_0, x_1) = f(h_0, x_1') := h_1$. Now, continuing the same approach to h_1 and so on, one obtains a 2^k-collision after finding k collisions. In Joux's method there is no restriction on the initial values or the function f, which can be assumed to be a FIL-RO or any other compression function.

Kelsey and Schneier describe a multicollision finding algorithm in their paper [8]. This algorithm is used to generate a second preimage attack against iterated hash functions and it has the complexity $O(2^{\frac{n}{2}})$ for arbitrarily large multicollisions. However, there are some assumptions on the underlying function f and the memory required by the attacker. The function f is assumed to have easily found fixed points, i.e. such values of f denoted by h and message blocks m for which $f(h, m) = h$. Also the attacker is assumed to have $O(2^{\frac{n}{2}})$ memory available for storing the intermediate values.

Aumasson [1] has modified the above method in such a way that there is no need to assume any memory for the attacker and the complexity remains the same $O(2^{\frac{n}{2}})$ for arbitrarily large multicollisions. This method also assumes that fixed points are easily found for f and that the attacker can choose the initial value. After this, one only needs to find one fixed point collision and arbitrarily large multicollisions can be generated.

In information sciences in general and in information security in particular it is a common habit to visualize iterated structures with (directed) graphs, e.g., [2]. It is well known that finite digraphs represent an equivalent notion to finite state automata. However, when the state space of a system grows large, graphs loose their power of expression. Also when it is necessary to treat computations as rigorous mathematical objects, system description with automata substantially outweighs graph representation.

3 Basics

In this section we give the basic definitions needed in our presentation. Let $\mathbb{N} = \{0, 1, 2, \ldots\}$ be the set of all natural numbers and $\mathbb{N}_+ = \mathbb{N} \setminus \{0\}$. For each $l \in \mathbb{N}_+$, we define $\mathbb{N}_l$ to be the set of l first positive integers: $\mathbb{N}_l = \{1, 2, \ldots, l\}$. For each finite set S, let $|S|$ be the cardinality of S, i.e. the number of elements in S.

An *alphabet* is any finite, nonempty set of abstract symbols called *letters*. Let A be an alphabet. A *word* (over A) is any finite sequence of symbols in A. Thus, assuming that w is a word over A, we can write $w = x_1x_2\cdots x_n$, where $n \in \mathbb{N}$ and $x_i \in A$ for $i = 1, 2, \ldots, n$. Above n is the *length* $|w|$ of w. Notice that n may be equal to zero; then w is the *empty word*, denoted by ϵ, which contains no letters. By $|w|_a$ we mean the number of occurrences of the letter a in w. If $w \neq \epsilon$, define the *alphabet of* w by $alph(w) = \{a \in A \,|\, |w|_a > 0\}$. Let $alph(\epsilon) = \emptyset$. The word u is *a subword* of w if there exist $m \in \mathbb{N}$ and $x_0, u_1, x_1 \ldots, u_m, x_m \in A^*$ such that $w = x_0u_1x_1 \ldots u_mx_m$, where $u = u_1u_2\cdots u_m$. A subword u of w is a *factor* of w if $w = x_0ux_1$ for some $x_0, x_1 \in A^*$. A factor u of w is a *prefix* of w if $w = ux_1$ for some $x_1 \in A^*$. The *catenation* of two words u and v in A^* is the word uv obtained by writing u and v after one another.

For each $n \in \mathbb{N}$, denote by A^n the set of all words of length n over A. Let $A^* = \bigcup_{n=0}^{\infty} A^n$ be the set of all words and $A^+ = A^* \setminus \{\epsilon\} = \bigcup_{n=1}^{\infty} A^n$ the set of all nonempty words over A. Clearly catenation defines a binary operation $\cdot$ in A^*: $u \cdot v = uv$ for all $u, v \in A^*$. In algebraic terms $(A^*, \cdot)$ is a *free monoid* and $(A^+, \cdot)$ is a *free semigroup*.

Each subset of A^* is a *language* (over A).

Let A and B be alphabets. A mapping $h : A^* \to B^*$ is a (*monoid*) *morphism* if $h(uv) = h(u)h(v)$ for each $u, v \in A^*$ and $h(\epsilon) = \epsilon$.

A *finite state automaton* (abbrev. FSA) is a 5-tuple $\mathcal{A} = (S, A, \delta, s_0, F)$, where

S is a finite nonempty set (the *the inner states* of $\mathcal{A}$)
A is an alphabet (the *input alphabet) of* $\mathcal{A}$
δ is a mapping: $S \times A \to S$ (the *transition function* of $\mathcal{A}$)
$s_0 \in S$ (the *initial state* of $\mathcal{A}$)
$F \subseteq S$ (the *final states* of $\mathcal{A}$).

The domain of δ is enlarged to $S \times A^*$ in a natural way:

$$\begin{cases} \delta(s, \epsilon) = s & \text{for each } s \in S\ , \\ \delta(s, ax) = \delta(\delta(s, a), x) & \text{for each } s \in S, a \in A \text{ and } x \in A^*\ . \end{cases}$$

The language $L_{\mathcal{A}}$ *accepted* by the finite state automaton $\mathcal{A} = (S, A, \delta, s_0, F)$ is the set

$$L_{\mathcal{A}} \ = \ \{\, w \in A^* \,|\, \delta(s_0, w) = s \text{ for some } s \in F\} \ .$$

A *finite transducer* (abbrev. FTR) is a 7-tuple $\mathcal{M} = (S, A, \delta, s_0, F, B, \lambda)$, where

S, A, δ, s_0 ja F are as in a finite state automaton
B is an alphabet (the *output alphabet* of $\mathcal{M}$)
λ is a mapping: $S \times A \to B^*$ (the *output function* of $\mathcal{M}$).

The domain of λ is enlarged to $S \times A^*$ in the following manner:

$$\begin{cases} \lambda(s, \epsilon) = \epsilon & \text{for each } s \in S\ , \\ \lambda(s, ax) = \lambda(s, a)\lambda(\lambda(s, a), x) & \text{for each } s \in S, a \in A \text{ and } x \in A^*\ . \end{cases}$$

The *transducer mapping* $\tau_{\mathcal{M}}$ defined by the finite transducer $\mathcal{M}$ is a partial function from the set A^* into the set B^* such that $\tau_{\mathcal{M}}(w) = u$ if and only if $\delta(s_0, w) \in F$ and $\lambda(s_0, w) = u$.

4 Hash functions and collisions

In this section we give basic definitions of hash functions as presented in [4] and multicollisions as presented in [7, 12, 6]. By *block representation* of a message, we mean the division and padding of the message into blocks of equal size. We may certainly without loss of generality assume that all our messages are over the binary alphabet $\{0, 1\}$.

Definition 1. *A* hash function *(of length n, $n \in \mathbb{N}_+$) is a mapping $f : \{0,1\}^* \to \{0,1\}^n$.*

An ideal hash function $f : \{0,1\}^* \to \{0,1\}^n$ is a variable input length random oracle (VIL-RO): for each $x \in \{0,1\}^*$, the value $f(x) \in \{0,1\}^n$ is chosen uniformly at random.

Let $r \in \mathbb{N}_+$. An *r-multicollision* on the hash function $f : \{0,1\}^* \to \{0,1\}^n$ is a set $A \subseteq \{0,1\}^*$ such that $|A| = r$ and $f(x) = f(y)$ for all $x, y \in A$. A 2-multicollision (on f) is also called a collision on f.

An r-multicollision attack (algorithm) on a hash function f can loosely be characterized to be an algorithm that finds an r-multicollision on f with some nonnegligible probability. The *complexity* of the attack can be measured, for instance, with respect to the number of messages the hash values of which have to be determined to carry out the attack successfully.

In the following, we shall derive the concept of an iterated hash function. Remember that all messages are assumed to be in a block representation form.

Definition 2. *A* compression function *(of block size m and length n) is a mapping $f : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n$ where $m, n \in \mathbb{N}_+, m > n$.*

Let $m, n \in \mathbb{N}_+, m > n$ and the compression function $f : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n$ be given.

Define the functions $f^i : \{0,1\} \times (\{0,1\}^m)^i \to \{0,1\}^n$, $i \in \mathbb{N}_+$, inductively as follows. Let $f^1 = f$. For each $i \in \mathbb{N}_+$

$$f^{i+1}(x, y_1 y_2 \cdots y_{i+1}) = f(f^i(x, y_1 y_2 \cdots y_i), y_{i+1}) \quad ,$$

where $x \in \{0,1\}^n$ and $y_1, y_2, \ldots, y_{i+1} \in \{0,1\}^m$. Let finally $f^+ : \{0,1\}^n \times (\{0,1\}^m)^+ \to \{0,1\}^n$ be the mapping for which $f^+(x, y) = f^i(x, y)$ for all $x \in \{0,1\}^n, i \in \mathbb{N}_+$, and $y \in (\{0,1\}^m)^i$.

The *iterated hash function* $H_f : \{0,1\}^n \times (\{0,1\}^m)^+ \to \{0,1\}^n$ (based on f) is defined as follows: Given the initial value $h_0 \in \{0,1\}^n$, and the message $x \in (\{0,1\}^m)^+$, let $H_f(h_0, x) = f^+(h_0, x)$. Now, the complexity of any attack against an iterated hash function is measured as the number of calls to the underlying compression function and not just as the number of messages for which the hash value has to be calculated.

5 Hash functions as finite state automata

Next, we construct a finite state transducer $\mathcal{M}_f$, which implements the iterated hash function based on the compression function $f : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n$ with block size m and length n ($m, n \in \mathbb{N}_+$, $m > n$), and with initial value $h_0 \in \{0,1\}^n$.

Informally, $\mathcal{M}_f$ works as follows. It reads the input from left to right one symbol (0, 1 or β) at a time. Suppose that the word $x_1x_2\cdots x_l\beta$, where $l \in \mathbb{N}_+$ and $x_i \in \{0,1\}^m$ for $i = 1, 2, \ldots, l$, is on the input tape. Then the transducer, starting the reading from the first symbol of x_1 in the state (h_0, ϵ), is after reading $x_1x_2\cdots x_i$ in the state $(f^i(h_0, x_1x_2\cdots x_i), \epsilon)$, $i = 1, 2, \ldots, l$. The transducer has in each step written the empty word ϵ on the output tape. Suppose now that $\mathcal{M}_f$ has read $x_1x_2\cdots x_l$, written ϵ on the output tape and lies in the state $(f^l(h_0, x_1x_2\cdots x_l), \epsilon)$. It then (i) reads the symbol β; (ii) changes its inner state to s_{acc} (an accepting state); and (ii) writes $H_f(h_0, x_1x_2\cdots x_l)$ $(= f^l(h_0, x_1x_2\cdots x_l))$ on the output tape. If the input is not of the aforementioned form $x_1x_2\cdots x_l\beta$, the transducer (at some stage) goes to a rejecting state s_{rej} and outputs only the empty word ϵ on the output tape.

More rigorously, let

$$\mathcal{M}_f \;=\; (S, A, \Delta, \delta, s_0, \lambda, F)$$

be a finite state transducer in which

(i) $S = \{0,1\}^n \times (\cup_{i=0}^{m-1}\{0,1\}^i) \cup \{s_{acc}, s_{rej}\}$, where s_{acc} and s_{rej} are new symbols;
(ii) $A = \{0, 1, \beta\}$, where β is a new symbol;
(iii) $\Delta = \{0, 1\}$;
(iv) δ is defined as follows: (1) $\delta((x,y),a) = (x, ya)$ for all $(x,y) \in S$, $|y| < m-1$, and $a \in \{0,1\}$; (2) $\delta((x,y),a) = (f(x,ya),\epsilon)$ for all $(x,y) \in S$, $|y| = m-1$, and $a \in \{0,1\}$; (3) $\delta((x,y),\beta) = s_{rej}$ for all $(x,y) \in S$, $y \neq \epsilon$; (4) $\delta((x,\epsilon),\beta) = s_{acc}$ for all $x \in \{0,1\}^n$; and (5) $\delta(s_{acc},a) = \delta(s_{rej},a) = s_{rej}$ for all $a \in A$;
(v) $s_0 = (h_0, \epsilon)$;
(vi) λ is defined as follows: (1) $\lambda((x,y),a) = \epsilon$ for all $(x,y) \in S$ and $a \in \{0,1\}$; (2) $\lambda((x,y),\beta) = \epsilon$ for all $(x,y) \in S$, $y \neq \epsilon$; (3) $\lambda((x,\epsilon),\beta) = x$ for all $x \in \{0,1\}^n$; and (4) $\lambda(s_{acc},a) = \lambda(s_{rej},a) = \epsilon$ for all $a \in A$; and
(vii) $F = \{s_{acc}\}$

One is easily convinced that the above finite state transducer $\mathcal{M}_f$ implements the informal description we gave above. Note that the number of states of the FTR (the underlying automaton, respectively) is huge. In this sense, it is not implementable in practise; the state space of the transducer cannot be presented in any efficient format in the memory of a real computer. However, the (abstract) transducer formulation allows the exact examination of the cycle structure of computations. Local cycles, which induce mutually independent pumpings in input words are characteristic to finite state automata and transducers. In the

next section we show how to utilise this property to construct arbitrarily large multicollision sets for iterated hash functions. More complicated devices (for instance two-way pushdown automata and transducers) can be applied to implement generalized iterated hash functions, for instance those introduced in [12] and [6].

6 Multicollisions by pumping

Let $\mathcal{M}_f = (S, A, \Delta, \delta, s_0, \lambda, F)$ be the finite state transducer that implements the traditional iterated hash function $H_f : \{0,1\}^n \times (\{0,1\}^m)^+ \to \{0,1\}^n$. Choose two (arbitrary) message blocks $x_1, x_2 \in \{0,1\}^m$ such that $x_1 \neq x_2$. For instance, one can make the choice $x_1 = 00\cdots0$ (meaning that x_1 consists of m zeros only) and $x_2 = 11\cdots1$ (i.e., x_2 consists of m ones only).

Generate a sequence $h_0, h_1, h_2 \ldots$ of compression function values and, respectively, a sequence $(h_0, \epsilon), (h_1, \epsilon), (h_2, \epsilon) \ldots$ of states of the finite state transducer by the rule $h_i = f(h_{i-1}, x_1)$ for $i = 1, 2, 3, \ldots$. Now, since f is designed to be a FIL-RO we may assume that the compression function values $h_0, h_1, h_2 \ldots$ are randomly and evenly distributed over the set $\{0,1\}^n$. By the birthday paradox, there exists, with a good probability, a collision among the first $2^{\frac{n}{2}}$ values $h_0, h_1, h_2 \ldots, h_{2^{\frac{n}{2}}}$. Let $r_1, r_2 \in \{1, 2, \ldots 2^{\frac{n}{2}}\}$, $r_1 < r_2$, be such that $h_{r_1} = h_{r_2}$. Now, the two messages $u_1 = x_1^{r_1}$ and $u_2 = x_1^{r_2}$ induce the same hash value $f^{r_1}(h_0, x_1^{r_1}) = f^{r_2}(h_0, x_1^{r_2}) = h_{r_1}$. The respective finite transducer $\mathcal{M}_f$, after reading the prefix $x_1^{r_1}$ (the prefix $x_1^{r_1}x_1^{r_2-r_1}$, resp.) of any message $x_1^{r_1}x_1^{r_2-r_1}w$, where $w \in (\{0,1\}^m)^+$, is in the state (h_{r_1}, ϵ). This means that the transducer contains a cycle: for each $k \in \mathbb{N}_+$, if $\mathcal{M}_f$ is in the state (h_{r_1}, ϵ) and reads $(x_1^{r_2-r_1})^k$, it returns to the state (h_{r_1}, ϵ). This means that for each $w \in (\{0,1\}^m)^+$, all messages (words) in the set (language) $D_w = x_1^{r_1}(x_1^{r_2-r_1})^*w$ have the same hash value, i.e., $f^+(h_0, x_1^{r_1}(x_1^{r_2-r_1})^i w) = f^+(h_0, x_1^{r_1}(x_1^{r_2-r_1})^j w)$ for all $i, j \in \mathbb{N}$.

Now, we wish to construct a multicollision set for the hash function H_f. Obviously D_w is such an infinite collision set. However, since $r_1 < r_2$, for each pair z_1, z_2 of distinct messages in D_w, the lengths of z_1 and z_2 are not equal, i.e., $|z_1| \neq |z_2|$. To overcome this obstacle, we proceed as follows. We repeat the procedure by starting with the hash value $h'_0 = f(h_{r_2}, x_2)$ and generating a sequence of compression function values $h'_0, h'_1, h'_2 \ldots$ (and a sequence of states $(h'_0, \epsilon), (h'_1, \epsilon), (h'_2, \epsilon) \ldots$ of $\mathcal{M}_f$) such that $h'_i = f(h'_{i-1}, x_2)$ for $i = 1, 2, 3, \ldots$. Again, since f is a FIL-RO, by the birthday paradox, there is with a significant probability, a collision among the values $h'_0, h'_1, h'_2 \ldots, h'_{2^{n/2}}$. Let $s_1, s_2 \in \{1, 2, \ldots 2^{\frac{n}{2}}\}$, $s_1 < s_2$, be such that $h'_{s_1} = h'_{s_2}$. Thus $f^{s_1}(h'_0, x_2^{s_1}) = f^{s_2}(h'_0, x_2^{s_2}) = h'_{s_1}$ and, more generally, all messages in any language $D'_w = x_1(x_1^{r_2-r_1})^* x_2(x_2^{s_2-s_1})^* w$, where $w \in (\{0,1\}^m)^+$ have the same hash value.

From automata-theoretic point of view, we have found a second cycle in the finite state transducer $\mathcal{M}_f$. The transducer, when having as input any word in the language D'_w, after reading the input, is in the state $f^+(h_0, x_1^{r_1}x_2x_2^{s_1}w)$ and writes this value on its output tape. As was pointed out before, after reading

$x_1^{r_1}$ and any power $(x_1^{r_2-r_1})^i$ of the word $x_1^{r_2-r_1}$ $\mathcal{M}_f$ is in the state $f(h_0, x_1^{r_1})$. Respectively, $\mathcal{M}_f$ after reading $x_1^{r_1}x_2x_2^{s_1}$ and any power $(x_1^{s_2-s_1})^j$ of the word $x_2^{s_2-s_1}$ is in the state $f(h_0, x_1^{r_1}x_2x_1^{s_1})$. For each $w \in (\{0,1\}^m)^+$, the language D'_w is a subset of the language accepted by the underlying finite state automaton of $\mathcal{M}_f$.

We wish to generate, for each integer $r \geq 2$, an r-collision set of equal length messages for the hash function H_f. Let $r \in \mathbb{N}_+$, $r \geq 2$, and $w \in \{0,1\}^m$. Define

$$C_{r,w} = \{x_1^{r_1}(x_1^{r_2-r_1})^{(s_2-s_1)i}x_2x_1^{s_1}(x_1^{s_2-s_1})^{(r_2-r_1)(r-1-i)}w \mid i \in \{0,1,\ldots,r-1\}\} \quad .$$

We claim that $C_{r,w}$ is an r-collision set of equal length messages for H_f. Clearly, by the above facts, all messages in $C_{r,w}$ have the same hash value. Since

$$\begin{aligned} &|x_1^{r_1}(x_1^{r_2-r_1})^{(s_2-s_1)i}\ x_2\ x_1^{s_1}(x_1^{s_2-s_1})^{(r_2-r_1)(r-1-i)}w| \\ &\qquad = [r_1 + s_1 + 1 + (r-1)(r_2-r_1)(s_2-s_1)]m + |w| \end{aligned}$$

for all $i \in \{0,1,\ldots,r-1\}$, all messages in $C_{r,w}$ are of equal length. Now, if any two messages in $C_{r,w}$ were the same, we would have a contradiction with our assumption that $x_1 \neq x_2$. Thus $|C_{r,w}| = r$ and $C_{r,w}$ is an r-multicollision set of equal length messages for H_f.

Remark 1. Clearly, to find $C_{r,w}$, alltogether $O(2^{\frac{n}{2}})$ applications of f are needed, so the complexity of the respective multicollision attack is $O(2^{\frac{n}{2}})$. It should, however, be noted that despite the complexity of the attack, the messages in the set $C_{r,w}$ may be very long; for each $u \in C_{r,w}$ we have the approximation $|u| \leq r \cdot m \cdot 2^n + 1$.

Remark 2. The messages in the set $C_{r,w}$ may be long, but their true (information theory or Kolmogorov) complexity is quite small. Even if x_1 and x_2 are chosen to be complex messages of length m, to present and store the multicollision set $C_{r,w}$, very little resources is required. In fact, one only needs to present and store the words x_1 and x_2 and the numbers r_1, r_2, s_1, and s_2. Only approximately $2m + \log_2 r_1 + \log_2 r_2 + \log_2 s_1 + \log_2 s_2 \leq 2m + 2n$ bits are needed. Also the possibility to choose arbitrarily the messages x_1 and x_2 increases the flexibility of the collision set and the number of alternatives to create it.

Remark 3. In the construction above, the factor $(r_2 - r_1)(s_2 - s_1)$ appearing in the power of the word x_1 can certainly be replaced by the least common multiplier $lcm(r_2 - r_1, s_2 - s_1)$ of the numbers $r_2 - r_1$ and $s_2 - s_1$. In average, the length of the words in $C_{r,w}$ can so be shortened.

As we have seen above, the pumping construction creates infinitely large multicollisions, while the work we have to do remains constant. The basic construction requires $2^{n/2+1}$ calls of the compression function f. On the downside, the generated messages are extremely long, namely for a 2^k-collision $r_1 + s_1 + 1 + (2^k - 1)(r_2 - r_1)(s_2 - s_1)$ message blocks. It is reasonable to assume this to be about $2^{n/2-1} + 2^{n/2-1} + 1 + (2^k - 1)2^{n/2-1}2^{n/2-1}$ message blocks, which brings

us to the messages with size class of 2^n blocks. This is of course huge, when we compare it to the attack by Joux, which gives us 2^k-collision, with messages of size k blocks.

The downside of the Joux attack is that we have to search for k 2-collisions to create a 2^k-collision. It is possible to combine these two attacks, while retaining many properties from both of them.

We start by calculating $f(h_0, x_1^{2^{n/2-1}}) = h_{2^{n/2-1}}$, where x_1 can be any message block. Next we search message blocks m_1 and m_2 such that $m_1 \neq m_2$, $f(h_{2^{n/2-1}}, m_1) = f(h_{2^{n/2-1}}, m_2) = h_{2^{n/2-1}+1}$. We continue by searching x_2 that satisfies $f(h_{2^{n/2-1}+1}, x_2) = f(h_0, x_1^{r_1})$ for some $0 \leq r_1 \leq 2^{n/2-1}$. Short combinatoric evaluation shows us that finding such x_2 is a bit easier, than finding a 2-collision. Let us name $M_{r_1} = x_1^{r_1}$, $M_1 = x_1^{2^{n/2-1}-r_1} m_1 x_2$, $M_2 = x_1^{2^{n/2-1}-r_1} m_2 x_2$.

After this we can create as large collision as we like by simply adding cycles M_1 or M_2 at the end of the messages. We can create 4-collision from messages $M_{r_1} M_1 M_1$, $M_{r_1} M_1 M_2$, $M_{r_1} M_2 M_1$, $M_{r_1} M_2 M_2$ and so on.

We have to do a bit more work than would be necessary for the normal pumping construction. Basically, we have to call the compression function $2.5 \cdot 2^{n/2}$ times. On the other hand, the length of the messages for 2^k-collision is $r_1 + k(2^{n/2-1} - r_1) + 2$ blocks, which gives us messages with size class of $2^{n/2}$ blocks.

It is worth noticing that this attack also retains the basic structure of the attack by Joux, meaning that messages, which form the collision, can be changed by simply changing a single message block to another with the same desired properties. This is essential in certain theoretical attacks against more general classes of iterated hash functions such as those presented in [12] and [6] and means that the above method can be directly applied with them. From these attacks especially [6] also uses extremely long messages.

6.1 Practical implications

Suppose for a while that the compression function f is not a FIL-RO and that the respective FTR contains an unsafe set of states, i.e., for certain (known) inputs the FTR enters into a part of the state space containing only a restricted number of states and stays there. Then certainly the lengths of the multicollision messages in the previous attack can be considerably shortened. In the light of fixed point vulnerabilities in many of the modern hash functions, it is quite probable that subsets of this type exist in many hash functions and thus our method could have practical implications in addition to its theoretical value.

These sets of weak states can be thought of as an extension of the fixed points mentioned earlier. A set of weak states is comprised of a set of states $W \subset S$ and a set of message blocks $M \subset \{0,1\}^m$ such that $f(m, w) \in W$ for all $m \in M$ and $w \in W$. A weaker assumption would be that for all $w \in W$ there exists $m \in M$ such that $f(m, w) \in W$. Now our method could be applied within these sets.

In the case of the stronger assumption, our method could be used to find two cycles within the set W. One merely chooses the two message blocks x_1

and x_2 from the set M. In this way finding the two cycles (and thus arbitrarily large multicollision sets) would take $O(\sqrt{|W|})$ compression function calculations. With the weaker assumption, our method can still be applied, but now in each step one has to go through message blocks from M until one obtains a state in W and make sure, that the two cycles are obtained with two distinct sequences of message blocks. Thus, in the worst case, finding the two cycles requires $O(|M|\sqrt{|W|})$ compression function calculations. If $|M|$ is considerably smaller than $|W|$, this would be of the same complexity than with the stronger assumption.

The length of the messages in both cases would be in the class $O(|W|)$ message blocks, unless one uses the least common multiplier in the exponent as mentioned in the earlier remark. This could reduce the length of the messages even further. In any case, this kind of application of our method could produce very practical ways to construct arbitrarily large multicollision sets for iterated hash functions.

In a way, finding a cycle in the FTR with our method corresponds to finding a set of weak states with $|W|$ in the size class of $O(2^{\frac{n}{2}})$ and $|M| = 1$. Of course much smaller sets could be found in real life iterated hash functions.

7 Discussion

As presented in the previous section, the pumping attack generates arbitrarily large multicollisions for iterated hash functions with time complexity $O(2^{\frac{n}{2}})$. It is clear that these messages contain an enormous amount of message blocks, but the presentation of these blocks is very simple as pointed out in the previous remarks. As [13] demonstrates, cycles in periodic functions can be found with negligible memory. Using the algorithm described in [13], our method can also be used with negligible memory.

In comparison to Joux's method, our method has better time complexity and does not have any additional assumptions on the attacker's capabilities or the underlying compression function f. However, the messages generated with our method come with one property, which is not present with messages generated with Joux's method. Given a multicollision set generated with our method, the verification that these messages form a multicollision for the given hash function is a cumbersome task, because the verifier has to calculate many hash values for extremely long messages. Thus, the verifier needs to do as much work as the attacker to be assured of the correctness of the multicollision set. Messages generated by Joux's method are very short and thus the verification is extremely easy and the work done by the verifier is negligible.

When comparing our results with Kelsey & Schneier or Aumasson, the asymptotic complexity is the same. On the other hand, our method does not suffer from the limitations of these two methods, and can be applied to any compression function, even a FIL-RO and with negligible memory. The problem of lengthy messages is of course present in our method as the other two methods generate much shorter messages.

If one considers practical attacks against iterated hash functions such as MD5, SHA-1 etc., Aumasson's method for finding multicollisions seems to be optimal, if fixed points can be found for these hash functions. Our method has very little practical value, if one assumes the compression function to be FIL-RO, because of the size of the messages. However, as a theoretical result, our method points out the limits of Merkle-Damgård type iterative structure against multicollision attacks.

We would like to point out that our method can be applied in situations, where the compression function is not a FIL-RO. Also, as mentioned in the end of the previous section, our method could be applicable, when single fixed points are not easy to find for the compression function, but there is a comparatively small set of vulnerable states within which our method can be applied. This way the length of the messages would not be an issue and finding multicollisions could be even faster than in the case of fixed points.

Furthermore, in the light of our new results and the previous results of Joux, we can see the following result. For iterated hash functions, finding arbitrarily many preimages, second preimages or collisions has the same complexity as finding a single preimage, second preimage or collision, respectively, even when the underlying compression function is a random oracle. The claims about preimages follow from the fact that any large enough collision set can be made into a preimage set by appending one suitable message block to all of the messages. This idea has already been used by Joux in his paper, where this fact is implicitly present. Thus, finding arbitrarily large preimage sets requires only one preimage attack in addition to the collision attack. The claim about collisions follows from the complexity of the pumping method demonstrated in this paper.

8 Conclusion

In this paper we have demonstrated a very natural way to study the theoretical properties of iterated hash functions. We have also demonstrated a new method of finding multicollisions for iterated hash functions. Our method is not depended on the compression function having any special properties, e.g. easily found fixed points, but works even when the compression function is a fixed input length random oracle. Furthermore, our method works with any initial value of the hash function. The drawback of our method is that the messages generating the multicollision set are extremely long and the verification of the multicollision requires as much work as the generation of the multicollision in the worst case.

However, our result could be applied in practice when single fixed points are hard to find but comparatively small sets of vulnerable states can be found. It would be an interesting research problem to try and find such sets of states and apply our method in this scenario. In these cases the length of the messages generated by our method would not pose a problem.

Future research into the automata-theoretic aspects of iterated hash functions could lead into more new results on the theoretical limits of iterated hash functions. Even the more complex methods of iteration such as HAIFA [3] and

wide-pipe hashing [10] can be easily presented as automata. This could reveal some new properties of these methods. Furthermore, one can model even more general classes of iterated hash functions such as those presented in [12] and [6] and thus gain more insight on these constructs.

It is also worth mentioning that these results are not too tightly bound only to hash functions, but these properties of multicollisions are more related to the iterative structure of the hash functions. Thus, similar ideas may be useful in studying other methods, which use iteration.

References

1. J.-P. Aumasson. Faster multicollisions. In D. R. Chowdhury, V. Rijmen, and A. Das, editors, *INDOCRYPT*, volume 5365 of *Lecture Notes in Computer Science*, pages 67–77. Springer, 2008.
2. M. Bellare, K. Pietrzak, and P. Rogaway. Improved security analyses for CBC MACs. In V. Shoup, editor, *CRYPTO*, volume 3621 of *Lecture Notes in Computer Science*, pages 527–545. Springer, 2005.
3. E. Biham and O. Dunkelman. A framework for iterative hash functions - haifa. Cryptology ePrint Archive, Report 2007/278, 2007. `http://eprint.iacr.org/`.
4. 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, 20–24 Aug. 1989.
5. H. Dobbertin. Cryptanalysis of MD4. *Journal of Cryptology: the journal of the International Association for Cryptologic Research*, 11(4):253–271, Fall 1998.
6. J. J. Hoch and A. Shamir. Breaking the ICE - finding multicollisions in iterated concatenated and expanded (ICE) hash functions. In M. J. B. Robshaw, editor, *FSE*, volume 4047 of *Lecture Notes in Computer Science*, pages 179–194. Springer, 2006.
7. A. Joux. Multicollisions in iterated hash functions. application to cascaded constructions. In M. K. Franklin, editor, *CRYPTO*, volume 3152 of *Lecture Notes in Computer Science*, pages 306–316. Springer, 2004.
8. J. Kelsey and B. Schneier. Second preimages on n-bit hash functions for much less than 2^n work. In R. Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 474–490. Springer Berlin / Heidelberg, 2005.
9. V. Klima. Finding MD5 collisions on a notebook PC using multi-message modifications. http://eprint.iacr.org/2005/102.pdf, Apr. 2005.
10. S. Lucks. Design principles for iterated hash functions. Cryptology ePrint Archive, Report 2004/253, 2004. `http://eprint.iacr.org/`.
11. R. 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, Santa Barbara, CA, USA, 1990. International Association for Cryptologic Research, Springer-Verlag, Berlin Germany.
12. M. Nandi and D. R. Stinson. Multicollision attacks on some generalized sequential hash functions. *IEEE Transactions on Information Theory*, 53(2):759–767, 2007.
13. R. Sedgewick, T. G. Szymanski, and A. C. Yao. The complexity of finding cycles in periodic functions. *SICOMP: SIAM Journal on Computing*, 11:376–390, 1982.
14. M. Stevens. Fast collision attack on MD5. http://eprint.iacr.org/2006/104.pdf, Mar. 2006.

15. K. Suzuki, D. Tonien, K. Kurosawa, and K. Toyota. Birthday paradox for multi-collisions. *IEICE Transactions*, 91-A(1):39–45, 2008.
16. X. Wang, Y. L. Yin, and H. Yu. Finding collisions in the full SHA-1. In V. Shoup, editor, *CRYPTO*, volume 3621 of *Lecture Notes in Computer Science*, pages 17–36. Springer, 2005.
17. X. Wang and H. Yu. How to break MD5 and other hash functions. In R. Cramer, editor, *EUROCRYPT*, volume 3494 of *Lecture Notes in Computer Science*, pages 19–35. Springer, 2005.