Do Broken Hash Functions Affect the Security of Time-Stamping Schemes?

Ahto Buldas[1,2,3,⋆] and Sven Laur[4,⋆⋆]

[1] Cybernetica, Akadeemia tee 21, 12618 Tallinn, Estonia
[2] Tallinn University of Technology, Raja 15, 12618 Tallinn, Estonia
[3] University of Tartu, Liivi 2, 50409 Tartu, Estonia
Ahto.Buldas@ut.ee
[4] Helsinki University of Technology, Laboratory for Theoretical Computer Science, P.O. Box 5400, FI-02015 TKK, Finland
slaur@tcs.hut.fi

Abstract. We study the influence of collision-finding attacks on the security of time-stamping schemes. We distinguish between *client-side hash functions* used to shorten the documents before sending them to time-stamping servers and *server-side hash functions* used for establishing one way causal relations between time stamps. We derive necessary and sufficient conditions for client side hash functions and show by using explicit separation techniques that neither collision-resistance nor 2nd preimage resistance is necessary for secure time-stamping. Moreover, we show that server side hash functions can even be not one-way. Hence, it is impossible by using black-box techniques to transform collision-finders into wrappers that break the corresponding time-stamping schemes. Each such wrapper should analyze the structure of the hash function. However, these separations do not necessarily hold for more specific classes of hash functions. Considering this, we take a more detailed look at the structure of practical hash functions by studying the Merkle-Damgård (MD) hash functions. We show that attacks, which are able to find collisions for MD hash functions with respect to *randomly chosen initial states*, also violate the necessary security conditions for client-side hash functions. This does not contradict the black-box separations results because the MD structure is already a deviation from the black-box setting. As a practical consequence, MD5, SHA-0, and RIPEMD are no more recommended to use as *client-side hash functions* in time-stamping. However, there is still no evidence against using MD5 (or even MD4) as *server-side* hash functions.

1 Introduction

Cryptographic hash functions are intended for transforming a message X of an arbitrary length into a digest $h(X)$ of a fixed length, which, in a way, represents the original message. Hash functions have several applications, such as electronic signatures,

⋆ Partially supported by Estonian SF grant no. 5870, and by EU FP6-15964: “AEOLUS”.

⋆⋆ Partially supported by Finnish Academy of Sciences, and by Estonian Graduate School in Information and Communication Technologies.

© Springer-Verlag Berlin Heidelberg 2006

fast Message Authentic Codes (MACs), secure registries, time-stamping schemes, etc. Without any doubt, modern information technology needs hash functions as much as it needs stream and block ciphers. Therefore, the importance of research on hash function security can hardly be overestimated.

Unfortunately, the speed of developing suitable theoretical basis for hash function security cannot be compared to the expansion rate of hash function applications. Not much is known about suitable design criteria, nor about how to formalize the security requirements that originate from practical applications. A remarkable fact which characterizes the shortage of information in this field is that in many cases when theoreticians are looking for ways of modeling hash functions they just replace them with "random oracles".

Theoretical models of hash functions often deal with a limited number of "universal" security properties – collision-freedom, one-wayness, etc. –, which are possibly neither sufficient nor necessary in the context of particular practical applications. Recent success in finding collisions for practical hash functions (MD4,MD5, RIPEMD, SHA-0) by Wang et al [16, 17, 19] and later improvements [12, 18, 9, 10] raise an important question: For which practical implementations are the collisions a real threat? Modifications in software are always expensive and it would clearly not be economical to replace hash functions in all applications "just in case".

The problem addressed in this paper is to clarify and formalize the security properties of hash functions which are necessary and sufficient in the context of time-stamping schemes, and more general in secure registries. Considering the increasing use of electronic registries and databases, it is important to know to what extent and how their security depends on the security of hash functions:

- Which properties of hash functions would guarantee the security of time-stamping schemes?
- What kind of practical attacks (collisions, second preimages, etc.) are a suitable basis for replacing the hash functions in time-stamping schemes?

Just a few years after the birth of the first practical hash functions, it was pointed out that the specific security properties as well as their mutual relationships should deserve more attention. For example, Ross Anderson [1] listed several "freedom properties" (different from collision-freedom) arising from cryptographic constructions and applications. Rogaway and Shrimpton [13] presented an exhaustive study about "classical" security properties of hash functions and their mutual relationships. Hsiao and Reyzin [7] pointed out a fundamental difference between so-called *public-coin* hash functions and *secret-coin* hash functions by showing that the former cannot be constructed from the latter in a black-box way.

In the context of time-stamping, it has been shown [4] that the *chain-resistance* property, which is necessary in time-stamping schemes, is not implied by classical properties like collision-resistance or one-wayness. As a positive result, it was shown recently [5] that if time-stamping schemes have an additional audit functionality, then even the strongest reasonable (*universally composable*) security level is achievable if the hash functions used are universally one-way, which is a weaker property than collision resistance.

Time-stamping schemes use hash functions for two different goals: (1) to shorten the messages on the client side and (2) create one-way temporal (casual) relationships on the server side. Hence, it is natural to think that the client-side hash function and the server-side hash function have different security requirements. Thus far, the security proofs of time-stamping schemes [4, 5] assume the collision-resistance of client-side hash functions. Hence, it is important to study if we can replace collision-resistance on the client side with weaker requirements like *2nd preimage resistance* or *one-wayness*.

In this paper, we derive necessary and sufficient conditions for client side hash functions and show by using explicit separation techniques that neither collision-resistance nor 2nd preimage resistance is necessary for secure time-stamping. Moreover, we also show that server side hash functions can even be not one-way. More precisely, we prove that if secure hash-based time-stamping (as used in practical schemes like [15]) is possible at all, then we can replace client side hash functions with hash functions that are not 2nd preimage resistant and use server side hash functions, which are not one-way. In spite of using two "insecure" hash functions, we are able to achieve a new and rather strong security requirement for time-stamping schemes. Hence, it is impossible by using black-box techniques to transform collision-finders into wrappers that break the corresponding time-stamping schemes. Each such wrapper should analyze the structure of the hash function. Still, the results mentioned above do not necessarily apply to more specific classes of hash functions.

Considering the above, we will take a more detailed look at the structure of practical hash functions by studying the Merkle-Damgård (MD) style hash functions. We will show that the attacks which are able to find collisions to MD hash functions with respect to *randomly chosen initial state* also violate the necessary security conditions for client-side hash functions. This still does not mean that the recent attacks to MD hash functions render the practical hash functions insecure, because the attacks mostly consider the fixed (standard) initial state (IV) of the hash function. However, it is claimed by Klima [9, 10] that MD5 collisions can be find for random initial states, which (when true) would mean that MD5 cannot be used as a *client-side hash function* in time-stamping schemes. However, there are still no convincing arguments against using MD5 (or even MD4) as a *server-side hash function*.

This paper mainly focuses on the so called *hash-based time-stamping*, in which cryptographic (signature) keys are not used. However, the results about *client-side hash functions* also apply to the so-called *signature-based time stamps* [11] that consist of client-computed hash values, time values, and digital signatures of trusted servers.

The paper is organized as follows. Section 2 provides the reader with necessary notation and definitions. Section 3 outlines the basics of secure hash-based time-stamping schemes. Section 4 introduces a new security requirement and derives sufficient conditions for the client side and the server side hash functions that together imply the new condition. In Section 5, we show that 2nd preimage resistance is not necessary for client side hash functions. Section 6 shows that server side hash functions are not necessarily one-way. In Section 7, we show that certain multi-collision attacks to MD hash functions violate the necessary condition for client side hash functions. Section 8 presents some open problems related to this work.

2 Notation and Definitions

By $x \leftarrow \mathcal{D}$ we mean that x is chosen randomly according to a distribution $\mathcal{D}$. If A is a probabilistic function or a Turing machine, then $x \leftarrow \mathsf{A}(y)$ means that x is chosen according to the output distribution of A on an input y. By $\mathcal{U}_n$ we denote the uniform distribution on $\{0,1\}^n$. If $\mathcal{D}_1, \ldots, \mathcal{D}_m$ are distributions and $F(x_1, \ldots, x_m)$ is a predicate, then $\mathsf{Pr}[x_1 \leftarrow \mathcal{D}_1, \ldots, x_m \leftarrow \mathcal{D}_m \colon F(x_1, \ldots, x_m)]$ denotes the probability that $F(x_1, \ldots, x_m)$ is true after the ordered assignment of $x_1, \ldots, x_m$. For functions $f, g \colon \mathbb{N} \to \mathbb{R}$, we write $f(k) = O(g(k))$ if there are $c, k_0 \in \mathbb{R}$, so that $f(k) \leq cg(k)$ $(\forall k > k_0)$. We write $f(k) = \omega(g(k))$ if $\lim_{k\to\infty} \frac{g(k)}{f(k)} = 0$. If $f(k) = k^{-\omega(1)}$, then f is *negligible*. A Turing machine M is *polynomial-time* (*poly-time*) if it runs in time $k^{O(1)}$, where k denotes the input size. Let FP be the class of all probabilistic functions $f \colon \{0,1\}^* \to \{0,1\}^*$ computable by a poly-time M.

A distribution family $\{\mathcal{D}_k\}_{k\in\mathbb{N}}$ is *poly-sampleable* if there is $\mathsf{D} \in \mathsf{FP}$ with output distribution $\mathsf{D}(1^k)$ equal to $\mathcal{D}_k$. A poly-sampleable distribution family $\{\mathcal{D}_k\}$ is *unpredictable* if $\mathsf{Pr}[x' \leftarrow \Pi(1^k), x \leftarrow \mathcal{D}_k \colon x = x'] = k^{-\omega(1)}$ for every predictor $\Pi \in \mathsf{FP}$. Two distribution families $\mathcal{D}^{(1)}$ and $\mathcal{D}^{(2)}$ are *indistinguishable* if for every distinguisher $\Delta \in \mathsf{FP}$: $|\ \mathsf{Pr}[x \leftarrow \mathcal{D}_k^{(1)} \colon \Delta(1^k, x) = 1] - \mathsf{Pr}[x \leftarrow \mathcal{D}_k^{(2)} \colon \Delta(1^k, x) = 1]\ | = k^{-\omega(1)}$.

Let $\{\mathfrak{F}_k\}_{k\in\mathbb{N}}$ be a distribution family such that every $h \leftarrow \mathfrak{F}_k$ is a (deterministic) function $h \colon \{0,1\}^\ell \to \{0,1\}^k$, where ℓ is polynomial in k. We say that $\{\mathfrak{F}_k\}$ is a *function distribution family*. For every $x, x' \in \{0,1\}^k$ let $C(x, x')$ denote the condition that (x, x') is a collision for h, i.e. $x \neq x'$ and $h(x) = h(x')$. By following the security notions in [13] we say that a randomly chosen $h \leftarrow \mathfrak{F}_k$ is:

- *Collision-Resistant* if $\forall \mathsf{A} \in \mathsf{FP} \colon\ \mathsf{Pr}[(x, x') \leftarrow \mathsf{A}(1^k, h) \colon C(x, x')] = k^{-\omega(1)}$.
- *Everywhere 2nd Preimage Resistant* (eSec) if $\forall \mathsf{A} \in \mathsf{FP}$:

$$\max_{x\in\{0,1\}^\ell} \mathsf{Pr}[x' \leftarrow \mathsf{A}(1^k, h) \colon C(x, x')] = k^{-\omega(1)}\ .$$

- *2nd Preimage Resistant* if $\forall \mathsf{A} \in \mathsf{FP} \colon\ \mathsf{Pr}[x \leftarrow \mathcal{U}_\ell, x' \leftarrow \mathsf{A}(x) \colon C(x, x')] = k^{-\omega(1)}$.
- *One-Way* if $\forall \mathsf{A} \in \mathsf{FP} \colon\ \mathsf{Pr}[x \leftarrow \mathcal{U}_\ell, x' \leftarrow \mathsf{A}(h(x)) \colon h(x') = h(x)] = k^{-\omega(1)}$.

If for every k there exists h_k so that $\mathsf{Pr}\,[h \leftarrow \mathfrak{F}_k \colon h = h_k] = 1$ then we have a fixed family of functions, i.e. for each k we have a single unkeyed hash function, e.g. SHA-1.

3 Security of Time-Stamping Schemes

In this paper, we focus on the security of *hash functions* used in time-stamping schemes. The other primitives supporting the time stamping schemes (like signature schemes or encryption schemes) are not studied in this paper. A time-stamping procedure consists of the following general steps:

- Client computes a hash $x = H(X)$ of a document X (where H is called a *client-side hash function*) and sends x to the Server.
- Server binds x with a time value t (a positive integer), either by using a digital signature or a hash-chain created by using another (server-side) hash function h.

For the self-consistency of this paper, we outline the basic facts about hash-chains and how they are used in time-stamping. In the definition of a hash-chain we use the following notation. We will follow the notation and definitions introduced in [4] except some technicalities which we change in order to make the definitions more usable for this work. By $\sqcup$ we mean the empty string. If $x = (x_1, x_2) \in \{0,1\}^{2k}$ and $x_1, x_2 \in \{0,1\}^k$ then by $y \in x$ we mean $y \in \{x_1, x_2\}$.

Definition 1 (Hash-Chain). *Let $h\colon \{0,1\}^{2k} \to \{0,1\}^k$ be a hash function.*[1] *By an h-chain from $x \in \{0,1\}^k$ to $r \in \{0,1\}^k$ we mean a (possibly empty) sequence $c = (c_1, \ldots, c_\ell)$ of pairs $c_i \in \{0,1\}^{2k}$, such that the following two conditions hold:*

(1) if $c = \sqcup$ then $x = r$; and
(2) if $c \neq \sqcup$ then $x \in c_1$, $r = h(c_\ell)$, and $h(c_i) \in c_{i+1}$ for every $i \in \{1, \ldots, \ell - 1\}$.

We denote by $F_h(x; c) = r$ the proposition that c is an h-chain from x to r. Note that $F_h(x; \sqcup) = x$ for every $x \in \{0,1\}^k$.

Time-stamping involves Server, Publisher, and two procedures for *time-stamping* a bit-string and for *verifying* a time stamp. It is assumed that Publisher is write-once and receives items from Server in an authenticated manner. Time-stamping procedure is divided into rounds of equal duration. During each round, Server receives requests $x_1, \ldots, x_N \in \{0,1\}^k$ from the users. If the t-th round is over, Server computes a digest $r_t = T^h(x_1, \ldots, x_N) \in \{0,1\}^k$ by using a hash function $h\colon \{0,1\}^{2k} \to \{0,1\}^k$ and a tree-shaped hashing scheme T^h. After that, Server issues a hash chain c (*certificate*) for each request x, such that $F_h(x; c) = r_t$. In the scheme of Fig. 1, the time-certificate for x_2 is $((x_1, x_2), (y_1, z_1))$, where $y_1 = h(x_1, x_2)$. Certificate c of a request x is verified by obtaining a suitable r_t form Publisher and checking whether $F_h(x; c) = r_t$. Intuitively, this proves that x existed at time t when r_t was published.

Security condition for time-stamping [4] is inspired by the following simplistic attack-scenario with a malicious Server:

– Server computes $r \in \{0,1\}^k$ (not necessarily by using T^h) and publishes it.
– Alice, an inventor, creates a description $X_A \in \{0,1\}^*$ of her invention and (possibly) obtains a certificate for the hash $x_A = H(X_A)$ of the description.
– Some time later, the invention is disclosed to the public and Server tries to steal it by showing that the invention was known to Server long before Alice time-stamped it. He creates a slightly modified version X of X_A, i.e. changes invertor's name, modifies the creation time, and possibly rewords the document in a suitable way (to have a "desired" hash value).
– Finally, Server computes a hash $x = H(X)$, and back-dates x, by finding a certificate c, so that $F_h(x; c) = r$.

[1] Twice-compressing hash functions are sufficient in the server side, and strictly for this purpose it is not necessary to apply hash functions with long input length. When h is implemented by using a practical hash function like MD5, it is sufficient to use only one input block. This detail is very important for the conclusions of this work.

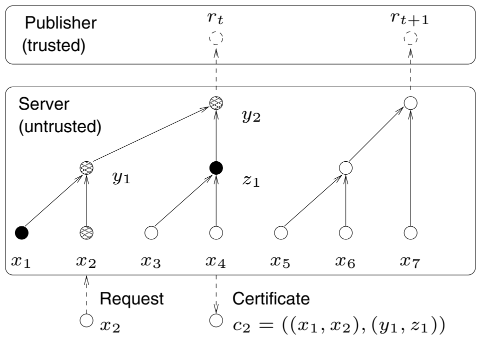

Fig. 1. Time-stamping by using a hash-function h

To formalize such a scenario, a two-staged adversary $\mathsf{A} = (\mathsf{A}_1, \mathsf{A}_2)$ is used. The first stage A_1 *computes* r (and an advice string a) after which the second stage A_2 on input a *new bit-string* $x \in \{0,1\}^k$ (modeled as an output of an unpredictable distribution $\mathcal{D}_k$) tries to find c, so that $F_h(x;c) = r$. The second stage can also use the advice string a if necessary. As h is the only cryptographic primitive used in the formal scenario, the security condition can be represented as a general requirement for a hash functions:

Definition 2 (Chain resistance – Chain**).** *A function distribution family* $\{\mathfrak{F}_k\}$ *of two-to-one hash functions* $h\colon \{0,1\}^{2k} \to \{0,1\}^k$ *is* chain resistant *iff for every unpredictable poly-sampleable distribution family* $\{\mathcal{D}_k\}_{k\in\mathbb{N}}$ *on* $\{0,1\}^k$:

$$\mathsf{Pr}[h\leftarrow\mathfrak{F}_k, (r,a)\leftarrow\mathsf{A}_1(1^k,h), x\leftarrow\mathcal{D}_k, c\leftarrow\mathsf{A}_2(x,a)\colon F_h(x,c) = r] = k^{-\omega(1)} \ . \quad (1)$$

Remark. In the definition above, a denotes *state information* stored by A_1 when computing the digest r. The reason why a is introduced is completely technical – we prefer *ordinary* Turing machines, which (unlike *interacting* machines) cannot save the state information between two calls. Informally, A_1 and A_2 are parts of a single adversary, and hence *all inputs and random coins of* A_1 *are available to* A_2.

To be more practical, we should take into account that lengthy documents are shortened by using another hash function $H\colon \{0,1\}^{\ell(k)} \to \{0,1\}^k$, which is not necessarily the same hash function as h, which is used by Server. Let $\{\mathfrak{F}_k^{\mathrm{c}}\}$ and $\{\mathfrak{F}_k^{\mathrm{s}}\}$ be the corresponding function distribution families producing functions of types $\{0,1\}^{\ell(k)} \to \{0,1\}^k$ and $\{0,1\}^{2k} \to \{0,1\}^k$ respectively.

Definition 3 (Secure (H,h)-time-stamping). *For every* $\mathsf{A} = (\mathsf{A}_1, \mathsf{A}_2) \in \mathsf{FP}$ *and for every unpredictable* $\mathcal{D}_k$ *on* $\{0,1\}^{\ell(k)}$ *the following probability is negligible:*

$$\mathsf{Pr}[H\leftarrow\mathfrak{F}_k^{\mathrm{c}}, h\leftarrow\mathfrak{F}_k^{\mathrm{s}}, (r,a)\leftarrow\mathsf{A}_1(1^k,H,h), X\leftarrow\mathcal{D}_k, c\leftarrow\mathsf{A}_2(X,a)\colon F_h(H(X),c)=r] \ . \quad (2)$$

This security definition may seem confusing for those who have got used to a ”folklore” belief that collision-resistance is essential for time-stamping. What if the inventor creates two colliding files, time-stamps one, and later tries to claim credits for the other? It is important here to notice that *this is not an attack in terms of time-stamping*! Indeed, both colliding files were created by the inventor approximately at the same time, and so there is nothing wrong in proving that the other file also existed at that time.

So far, security proofs exist only for time-stamping schemes which are "bounded" somehow. For example, if H and h are collision-resistant, then a (H, h)-time-stamping can be proven secure if the number of the allowed hash chain "shapes" is restricted to polynomial [4], or if there is an additional audit functionality included into the scheme [5]. It is also known [4] that the claim "h is collision-resistant $\Rightarrow$ h is chain-resistant" cannot be proven in a black-box way. One of the main objectives of this paper is to clarify whether collision-resistance of h (and of H) is necessary for secure time-stamping.

4 New Security Condition

There are several concerns related to the security condition (2). First, chain-resistance is a *necessary* property for h but it is not yet known whether it is *sufficient*, i.e. if H is collision resistant and h is chain-resistant, there are no known results for concluding that the time-stamping scheme (that uses H and h) is secure.

Another concern about (2) is that the adversary does not participate in the generation of X, i.e. X is picked independent of the adversary. This does not match with the informal description of the back-dating attack, where X was created by the adversary based on another document X_A and hence it is quite natural to assume that the adversary is able to "tune" the distribution $\mathcal{D}_k$ according to which the new document X is chosen. Based on these ideas, we give a new stronger security condition for (H, h)-time-stamping in which X is chosen by A_2. We still have to assume that X is unpredictable and hence we have to allow only those adversaries that produce unpredictable X. It is also important to require that A_2 adds "his own randomness" into X, i.e. X should be unpredictable even if the output and the random coins of A_1 are known.

We derive a necessary and sufficient security condition for the client side hash function H. Roughly saying, H must not destroy the computational entropy in a catastrophic way – unpredictable input distributions transform to unpredictable output distributions.

We prove that the new condition is not weaker than (2). We also propose a new stronger condition for h – *Strong Chain-Resistance* (sChain), which is sufficient for secure time-stamping. We prove that if H is unpredictability-preserving and h is strongly chain-resistant, then we have a secure (H, h)-time-stamping scheme in terms of (2).

4.1 New Security Definition

Let FPU be the class of all two-staged probabilistic poly-time adversaries $(\mathsf{A}_1, \mathsf{A}_2)$, such that the first output component is unpredictable, even if the output of A_1 is known to the predictor, i.e. for every poly-time predictor Π:

$$\mathsf{Pr}[(r, a) \leftarrow \mathsf{A}_1(1^k), x' \leftarrow \Pi(r, a), (x, c) \leftarrow \mathsf{A}_2(a) \colon x' = x] = k^{-\omega(1)} \enspace .$$

Note that as the additional inputs (r, a) of Π are generated by a uniform machine $\mathsf{A}_1(1^k)$ this definition does not imply unpredictability in the *non-uniform* model. Note also that is is reasonable to assume that the advice string a contains all internal random coins of

A_1 because concealing these coins by A_1 certainly would not make any attacks easier. Moreover, as the main role of Π is to measure the capability of A_1 to predict the future, then for this measure to be adequate Π has to know the random coins of A_1.

Definition 4 (Secure (H,h)-time-stamping). *A (H,h)-time-stamping scheme is secure if for every $(\mathsf{A}_1,\mathsf{A}_2)\in\mathsf{FPU}$ the next probability is negligible:*

$$\mathsf{Pr}[H\leftarrow\mathfrak{F}^{\mathrm{c}}_k, h\leftarrow\mathfrak{F}^{\mathrm{s}}_k, (r,a)\leftarrow\mathsf{A}_1(1^k, H,h), (X,c)\leftarrow\mathsf{A}_2(a)\colon F_h(H(X);c)=r]\ . \qquad (3)$$

It is easy to see that (3) implies the old condition (2). Indeed, if $(\mathsf{A}_1,\mathsf{A}_2)\in\mathsf{FP}$ breaks (H,h)-time-stamping in terms of (2) with success $\delta(k)$, then define $\mathsf{A}'_2(a)$ that picks $x\leftarrow\mathcal{D}_k$, computes $c\leftarrow\mathsf{A}_2(x,a)$, and outputs (x,c). By definition, $(\mathsf{A}_1,\mathsf{A}'_2)\in\mathsf{FPU}$ breaks (H,h)-time-stamping in terms of (3) with success $\delta(k)$.

Remark. It is insufficient to assume that X is unpredictable without advice, because then the condition (3) would be not achievable. Indeed, let A_1 be an adversary who generates X at random and outputs $(H(X),X)$ (where H is the client-side hash function) and let $\mathsf{A}_2(1^k,a)$ be an adversary who always outputs $(a,||)$. For such an adversary

$$\mathsf{Pr}[H\leftarrow\mathfrak{F}^{\mathrm{c}}_k, h\leftarrow\mathfrak{F}^{\mathrm{s}}_k, (r,a)\leftarrow\mathsf{A}_1(1^k,H,h), (x,c)\leftarrow\mathsf{A}_2(a)\colon F_h(H(x),c)=r]=1.$$

4.2 Necessary and Sufficient Requirements for *H*

Finding collisions for H does not mean that the time-stamping scheme is insecure according to our definitions. A single collision is not sufficient to produce probability distribution with high uncertainty. In a way, one single collision allows one to backdate a single document that is known before the digest is produced, leaving the majority of temporal dependencies intact. It turns out that the following entropy-preservation property is necessary and sufficient for the client-side hash function H.

Definition 5 (Unpredictability preservation – uPre). *A function distribution family $\{\mathfrak{F}_k\}$ is* unpredictability preserving*, if for every unpredictable poly-sampleable distribution family $\{\mathcal{D}_k\}$ and for every predictor $\Pi\in\mathsf{FP}$:*

$$\mathsf{Pr}[H\leftarrow\mathfrak{F}_k, y\leftarrow\Pi(1^k,H), x\leftarrow\mathcal{D}_k\colon y=H(x)]=k^{-\omega(1)}\ .$$

A fixed $H\colon\{0,1\}^{\ell(k)}\to\{0,1\}^k$ is uPre iff it converts unpredictable poly-sampleable distributions $\mathcal{D}_k$ to unpredictable output distributions $H(\mathcal{D}_k)$.

Remark. Poly-sampleability of $\mathcal{D}_k$ is crucial, because if $H_k\colon\{0,1\}^{\ell(k)}\to\{0,1\}^k$ and $\ell(k)=k+\omega(\log k)$, then there exists a family $\mathcal{D}_k$ with Rényi entropy $\mathsf{H}_2[\mathcal{D}_k]=\omega(\log k)$, such that $\mathsf{H}_2[H(\mathcal{D}_k)]=0$. Indeed, $\exists y\in\{0,1\}^k$ for which $\mid H^{-1}(y)\mid=(2^{k+\omega(\log k)})/2^k=k^{\omega(1)}$. Define $\mathcal{D}_k$ as the uniform distribution on $H^{-1}(y)$.

Theorem 1. *Unpredictability preservation is a necessary requirement for H: in every secure (H,h)-time-stamping scheme, the client-side hash function H is* uPre.

Proof. Let $\mathcal{D}_k$ be unpredictable and $\mathsf{\Pi}$ be a predictor for $H(\mathcal{D}_k)$ with success probability $\pi(k) = \mathsf{Pr}[H \leftarrow \mathfrak{F}_k^{\mathrm{c}}, y \leftarrow \mathsf{\Pi}(1^k, H), x \leftarrow \mathcal{D}_k \colon H(x) = y]$. Define $\mathsf{A}_1(1^k, H, h) \equiv \mathsf{\Pi}(1^k, H)$ and A_2 which on input x outputs $(x, \lfloor\rfloor)$. As $F_h(H(x); \lfloor\rfloor) = H(x) = y$ whenever $\mathsf{\Pi}$ is successful, the success of $(\mathsf{A}_1, \mathsf{A}_2)$ in terms of (2) is $\pi(k)$. Hence, $\pi(k)$ must be negligible and H is uPre. □

Definition 6 (Strong chain-resistance – sChain**).** *A function distribution family* $\{\mathfrak{F}_k\}$ *is strongly chain-resistant, if for every* $(\mathsf{A}_1, \mathsf{A}_2) \in \mathsf{FPU}$*:*

$$\varepsilon(k) = \mathsf{Pr}[h \leftarrow \mathfrak{F}_k, (r, a) \leftarrow \mathsf{A}_1(1^k, h), (x, c) \leftarrow \mathsf{A}_2(a) \colon F_h(x; c) = r] = k^{-\omega(1)} \enspace .$$

Theorem 2. *For secure* (H, h)*-time-stamping in terms of (3) it is sufficient that h-is* sChain*,* H *is* uPre *and the distribution* $H \leftarrow \mathfrak{F}_k^{\mathrm{c}}$ *is poly-sampleable.*

Proof. Let $(\mathsf{A}_1, \mathsf{A}_2) \in \mathsf{FPU}$ an adversary with success

$$\varepsilon(k) = \mathsf{Pr}[H \leftarrow \mathfrak{F}_k^{\mathrm{c}}, h \leftarrow \mathfrak{F}_k^{\mathrm{s}}, (r, a) \leftarrow \mathsf{A}_1(1^k, H, h), (X, c) \leftarrow \mathsf{A}_2(a) \colon F_h(H(X); c) = r] \enspace .$$

Define $\mathsf{A}_1'(1^k, h)$ that picks $H \leftarrow \mathfrak{F}_k^{\mathrm{c}}$, computes $(r, a) \leftarrow \mathsf{A}_1(1^k, H, h)$ and outputs (r, a'), where $a' = (a, H)$. Define $\mathsf{A}_2'(a')$ that parses a' to obtain a and H, calls $(X, c) \leftarrow \mathsf{A}_2(a)$ and outputs $(H(X), c)$. We have $(\mathsf{A}_1', \mathsf{A}_2') \in \mathsf{FPU}$, because H is uPre. Obviously, $(\mathsf{A}_1', \mathsf{A}_2')$ breaks h in terms of sChain with success $\varepsilon(k)$. □

5 Unpredictability Preservation vs 2nd Preimage Resistance

It is known that every collision-resistant function is uPre [5]. However, it turns out that 2nd preimage resistance does not imply uPre and *vice versa*, which means that client-side hash functions need not be 2nd preimage resistant.

Theorem 3. *If* uPre *hash functions exist (i.e. if secure time-stamping with client side hashing is possible at all), then there are hash functions which are* uPre *but not 2nd preimage resistant.*

Proof. Let $H \colon \{0,1\}^{\ell(k)} \to \{0,1\}^k$ (chosen randomly from $\mathfrak{F}_k$) be uPre. Define $H'(X') = H(X' \,\mathsf{or}\, 1)$ for every $X' \in \{0,1\}^{\ell(k)}$, where or denotes the logical bitwise OR-operation. Let $\mathfrak{F}_k'$ denote the distribution of H'. Obviously, H' is not 2nd preimage resistant. To show that H' is uPre, let us assume that $\mathcal{D}_k$ is an unpredictable distribution and $\mathsf{\Pi}$ is a poly-time predictor for $H'(\mathcal{D}_k)$. As the distribution $\mathcal{D}_k' = (\mathcal{D}_k \,\mathsf{or}\, 1)$ is also unpredictable, the success probability of $\mathsf{\Pi}$ is

$$\begin{aligned}\pi(k) &= \mathsf{Pr}[H' \leftarrow \mathfrak{F}_k', y \leftarrow \mathsf{\Pi}(1^k, H'), X' \leftarrow \mathcal{D}_k \colon H'(X') = y] \\ &= \mathsf{Pr}[H \leftarrow \mathfrak{F}_k, y \leftarrow \mathsf{\Pi}'(1^k, H), X \leftarrow \mathcal{D}_k' \colon H(X) = y] = k^{-\omega(1)} \enspace ,\end{aligned}$$

because H is uPre. Here $\mathsf{\Pi}'(1^k, H)$ just transforms H to H' and returns $\mathsf{\Pi}(1^k, H')$. □

On the other hand, it turns out that 2nd preimage resistance does not imply uPre and is thereby also insufficient for client side hash functions. Recall that collision-resistance was sufficient on the client side (but still not on the server side [4]).

Theorem 4. *If there are hash functions which are 2nd preimage resistant, then there are hash functions that are 2nd preimage resistant but not* uPre.

Proof. Let $H\colon \{0,1\}^{\ell(k)} \to \{0,1\}^k$ be 2nd preimage resistant and $\ell(k) = k+\omega(\log k)$. We construct a function $H'\colon \{0,1\}^{\ell'(k)} \to \{0,1\}^k$ which is 2nd preimage resistant but not uPre. Let $\ell'(k) = \ell(k-1)$ for all $k > 1$, and for every $X \in \{0,1\}^{\ell'(k)}$:

$$H'_k(X) = \begin{cases} 0^k & \text{if } X = 0^{k-1}\|X_1 \text{ for an } X_1 \in \{0,1\}^{\ell(k-1)-k+1} \\ 1\|H_{k-1}(X) & \text{otherwise.} \end{cases}$$

Define $\mathcal{D}$ on $\{0,1\}^{\ell'(k)}$, so that $\mathcal{D}_k = 0^{k-1}\|\mathcal{U}_{\ell(k-1)-k+1}$. $\mathcal{D}$ is unpredictable because it has Rényi entropy $\mathsf{H}_2(\mathcal{D}_k) = \ell(k-1)-k+1 = \omega(\log k)$. As the output distribution $H'(\mathcal{D})$ has no entropy at all, we conclude that H' is not uPre. At the same time, H' is 2nd preimage resistant because the probability that the first $k-1$ bits of a uniformly chosen $X \leftarrow \mathcal{U}_{\ell(k)}$ are all zeroes is $2^{-(k-1)}$, which is negligibly small. □

It is interesting to note that if in the everywhere second preimage-resistance (eSec) condition the adversary is prevented from abusing a small set of pre-computed existential collisions (which do not affect the security of time-stamping schemes) then we obtain a weaker condition weSec which turns out to be equivalent to uPre. This shows that eSec is a sufficient (but not necessary) condition for client-side hash functions. In this weaker requirement, the class of adversaries is restricted by requiring that the second pre-image X' produced by an adversary is distributed according to a high-entropy distribution. Though the following theorem holds for a fixed family H, it is possible to generalize the definition and the proof to arbitrary function distribution families.

Theorem 5. *For fixed families $H = \{H_k\}$,* uPre *is equivalent to the following* weak everywhere 2nd preimage resistance *(*weSec*) condition: For every poly-sampleable unpredictable distribution family $\mathcal{A}_k$ on $\{0,1\}^{\ell(k)}$:*

$$\max_{X\in\{0,1\}^{\ell(k)}} \mathsf{Pr}[X' \leftarrow \mathcal{A}_k\colon\ X'{\neq}X, H(X'){=}H(X)] = k^{-\omega(1)} \ .$$

Proof. $\mathsf{weSec} \implies \mathsf{uPre}$**:** Let $\mathcal{D}_k$ be unpredictable and Π be a predictor for $H(\mathcal{D}_k)$ with success $\pi(k) = \mathsf{Pr}[y \leftarrow \Pi(1^k), X' \leftarrow \mathcal{D}_k\colon y = H(X')] \neq k^{-\omega(1)}$. Hence, there is $y \in \{0,1\}^k$ such that $\mathsf{Pr}[X' \leftarrow \mathcal{D}_k\colon y = H(X')] \geq \pi(k)$ and we have

$$\max_{X\in\{0,1\}^{\ell(k)}} \mathsf{Pr}[X' \leftarrow \mathcal{D}_k\colon H(X'){=}H(X)] \geq \pi(k) \neq k^{-\omega(1)} \ .$$

As $\mathop{\mathsf{Pr}}\limits_{X'\leftarrow\mathcal{D}_k}[H(X'){=}H(X)] = \mathop{\mathsf{Pr}}\limits_{X'\leftarrow\mathcal{D}_k}[X'{=}X] + \mathop{\mathsf{Pr}}\limits_{X'\leftarrow\mathcal{D}_k}[X'{\neq}X, H(X'){=}H(X)]$ and the first probability in the sum is negligible (because $\mathcal{D}_k$ is unpredictable), the second one must be non-negligible and hence $\mathcal{D}_k$ breaks H in the sense of weSec.

$\mathsf{uPre} \implies \mathsf{weSec}$: Let $\mathcal{A}_k$ be a unpredictable distribution on $\{0,1\}^{\ell(k)}$ and let $X \in \{0,1\}^{\ell(k)}$ be a bit-string such that $\delta(k) = \mathop{\mathsf{Pr}}\limits_{X'\leftarrow\mathcal{A}_k}[X'{\neq}X, H(X'){=}H(X)] \neq k^{-\omega(1)}$. Therefore, $\mathop{\mathsf{Pr}}\limits_{X'\leftarrow\mathcal{A}_k}[H(X'){=}H(X)] \geq \delta(k) \neq k^{-\omega(1)}$ and $H(\mathcal{A}_k)$ predicts itself with success $\pi(k) = \mathsf{Pr}[X' \leftarrow \mathcal{A}_k, X'' \leftarrow \mathcal{A}_k\colon H(X''){=}H(X')] \geq \delta^2(k) \neq k^{-\omega(1)}$. □

6 Strong Chain-Resistance vs One-Wayness

In this section, we show that the server side hash function h is not necessarily one-way.

Theorem 6. *For every secure (H, h)-time-stamping scheme, there is a secure (H, h')-time-stamping scheme, where h' is not one-way (and hence not collision-resistant and not 2nd preimage resistant).*

Proof. Define h' that behaves like h, except that $h'(x, x) = x$ for every $x \in \{0, 1\}^k$. The new function h' is clearly not one-way. To show that h' is strongly chain-resistant, let $\mathsf{A}_1 \in \mathsf{FP}$ and $\mathsf{A}_2 \in \mathsf{FPU}$ be an adversaries for h' with success

$$\varepsilon(k) = \mathsf{Pr}[(r, a) \leftarrow \mathsf{A}_1(1^k), (X, c) \leftarrow \mathsf{A}_2(a) \colon F_{h'}(H(X); c) = r] \neq k^{-\omega(1)} \ .$$

Define a new A_2' that calls $(x, c) \leftarrow \mathsf{A}_2$ and outputs (x, c'), where c' is produced from c by deleting all elements c_i of the form (y, y). It is easy to verify that $F_h(H(X); c') = F_{h'}(H(X); c) = r$ (which is true even if c' is empty) and hence $(\mathsf{A}_1, \mathsf{A}_2')$ breaks the (H, h)-time-stamping scheme. A contradiction. □

Note that the proof also shows that strong chain resistant functions are not necessarily one-way functions, i.e. the chain resistance property is quite separated from other standard requirements for hash functions. Recall that there are no black-box proofs [4] for showing that collision-resistance implies chain resistance.

7 Implications to Practical Iterated Hash Functions

In this section, we will study what kind of collision-finding attacks to practical (client side) hash functions would make them insecure for time-stamping. We use the fact that most of the practical hash functions use the Merkle-Damgård construction, which (in order to compute hash for long messages) iterates a fixed compression function f.

Definition 7 (Merkle-Damgård Hash). *Let $f_k : \mathcal{S}_k \times \mathcal{M}_k \rightarrow \mathcal{S}_k$ be a family of poly-time compression functions and $g_k : \mathcal{S}_k \rightarrow \mathcal{T}_k$ be a family of poly-time output functions. Let the state update function $F_k : \mathcal{S}_k \times \mathcal{M}_k^* \rightarrow \mathcal{S}_k$ be defined by $F_k(s, x_1, \ldots, x_r) = f_k(\cdots f_k(s, x_1), \ldots, x_r)$. Then $h_k : \mathcal{S}_k \times \mathcal{M}_k^* \rightarrow \mathcal{T}_k$, defined by $h_k(s, x) = g(F_k(s, x))$, is a family of iterative (Merkle-Damgård) hash functions.*

Definition 8 (Collision-resistance w.r.t random initial state). *A family $\{h_k\}$ of MD hash functions is* collision resistant (w.r.t. to random initial state) *if for every $\mathsf{A} \in \mathsf{FP}$:*

$$\mathsf{Pr}\left[s \leftarrow \mathcal{S}_k, (x_0, x_1) \leftarrow \mathsf{A}(1^k, s, h_k) : x_0 \neq x_1,\ h_k(s, x_0) = h_k(s, x_1)\right] = k^{-\omega(1)} \ .$$

The internal state of $\{h_k\}$ is said to be collision resistant w.r.t. random initial state *if the state update function family $\{F_k\}$ is collision-resistant w.r.t. random initial state.*

Definition 9 (Collision-resistance w.r.t fixed initial state s_0). *A family of MD hash functions $\{h_k\}$ is collision resistant (w.r.t. to a fixed initial state s_0) if for every $\mathsf{A} \in \mathsf{FP}$:*

$$\mathsf{Pr}\left[(x_0, x_1) \leftarrow \mathsf{A}(1^k, f) : x_0 \neq x_1,\ h_k(s_0, x_0) = h_k(s_0, x_1)\right] = k^{-\omega(1)} \ .$$

The internal state of $\{h_k\}$ is said to be collision resistant w.r.t. fixed initial state s_0 *if the state update function family $\{F_k\}$ is collision-resistant w.r.t. fixed initial state.*

7.1 Discussion on Practical Hash Functions

In practical MD-hash functions the initial state s_0 (so called *Initial Value – IV*) is fixed by standards and is not chosen randomly. In order to formally define the collision-resistance of such functions, we have to assume that the compression function f is chosen randomly in accordance to a distribution $\mathfrak{F}$. Otherwise, an adversary can abuse a single existential collision which always exists because hash functions compress data.

It is important to distinguish between two kinds of collision-finding attacks: (1) attacks that find collisions for a fixed (standard) initial state, or more general, for a limited number of “weak” initial states, and (2) attacks that find collisions for random initial states (i.e. for a non-negligible fraction of initial states). In some sense these two types of attacks are incomparable in strength. For example, if the standard initial value s_0 is weak but still almost all other values are strong, then there are attacks of the first type but no attacks of the second type. If in turn the standard s_0 is strong and a non-negligible fraction of other states are weak, then there exist attacks of the second type but no attacks of the first type. However, these cases are ruled out by the following heuristic assumptions about the design of practical hash functions:

– *Reasonable choice of the standard IV:* Widely used hash functions are designed by specialists with good experience. Hence, it is reasonable to believe that *the choice of standard IV is at least as good as a random choice*. Hence, the situation where the standard IV is weak but almost all other IV-s are strong is extremely unlikely.
– *Reasonably efficient encoding of the internal state:* It is reasonable to believe that hash functions are designed quite efficiently, i.e. there is no considerable amount of redundancy in the initial state. Hence, it is also unlikely that the standard IV is strong but still a non-negligible fraction of other IV-s are weak. This is because the output of the compression function (in case of random inputs) is intuitively viewed as a random value, which would mean that weak initial states will eventually occur. (See the Computational Uniformity assumption below)

Therefore, it is reasonable to believe that efficient collision finders w.r.t. fixed IV imply the existence of efficient collision-finders w.r.t. random IV. Still, this does not mean that we *know* how to find collisions for random IV, though the heuristic assumptions above suggest that such attacks exist. The latest attacks against MD5 by Wang [16, 17] and by Klima [10] are claimed to be able to find collisions for arbitrary IV.

We show that *collision-finding attacks w.r.t. random IV are sufficient to render the client-side hash function H insecure for time-stamping*, i.e. H is no more uPre. This means that MD5 and MD4 are probably insecure as *client-side hash functions* in time-stamping. However, as we show later, this still does not mean that MD5 (or even MD4) are insecure as *server-side hash functions*.

The next property of MD hash functions (*Computational Uniformity*) is not an explicit design goal, but is often implicitly assumed in heuristic discussions about hash functions. Indeed, it has been shown [3] that hash functions must be almost regular to withstand birthday attacks. This suggests that some kind of statistical uniformity must hold for secure hash functions and hence the computational indistinguishability from uniform distribution is not a so far-fetched assumption.

Definition 10 (Computational uniformity). *Let ℓ be a polynomial and $U_{\ell(k)}$ denote uniform distribution on $\mathcal{M}_k^{\ell(k)}$. We say that iterative hash function family $\{h_k\}$ is computationally uniform w.r.t. length restriction ℓ, if $h_k(s, U_{\ell(k)})$ is computationally indistinguishable from uniform distribution on $\mathcal{T}_k$ for any $s \in \mathcal{S}_k$.*

7.2 Collisions of MD-Hash Functions Affect uPre

In the following, we will prove two results. First, if a collision finder has non-negligible success probability for every initial state, then the iterative hash function violates the uPre property. The second result states that the average-case and worst-case complexities for collision finding are roughly the same, if we assume *computational uniformity* from the compression function. Thus, it is quite likely that uPre implies collision resistance w.r.t. random initial value for all practical iterative hash functions.

Theorem 7. *Let $\{h_k\}$ be a fixed family of iterative hash functions. Then unpredictability preservation implies negligible worst-case success probability for all collision finders of $\{F_k\}$, i.e. for every $\mathsf{A} \in \mathsf{FP}$:*

$$\min_{s_0 \in \mathcal{S}_k} \mathsf{Pr}\left[(x_0, x_1) \leftarrow \mathsf{A}(s_0) : x_0 \neq x_1,\ F_k(s_0, x_0) = F_k(s_0, x_1)\right] = k^{-\omega(1)} \ .$$

Proof. For the sake of contradiction, assume that there exists an algorithm A that the worst-case success probability is larger than k^{-c} for infinitely many indices. Then running A sufficiently many times (polynomial in k) assures that we fail with negligible probability. Denote this algorithm by A'. Then we start A' on s_0 and get a collision pair (x_1^0, x_1^1) such that $s_1 = F_k(s_0, x_1^0) = F_k(s_0, x_1^1)$. Similarly, we can find the following collisions $s_i = F_k(s_{i-1}, x_i^0) = F_k(s_{i-1}, x_i^1)$, $i = 1, \ldots, k$. The total failure probability is still negligible. Now, for any $b \in \{0,1\}^k$, the corresponding hash value $h_k(x_1^{b_1} \ldots x_k^{b_k})$ is the same. The distribution $\mathcal{D} = \{x_1^{b_1} \ldots x_k^{b_k} : b \in \{0,1\}^k\}$ is poly-sampleable and has min-entropy k, but $H(\mathcal{D})$ has no min-entropy. A contradiction. □

Theorem 8. *Let $\{F_k\}$ be a fixed family of computationally uniform compression functions. Then the negligible worst-case success probability for all collision finders of $\{F_k\}$ implies collision resistance w.r.t. random initial state.*

Proof. Since $\{F_k\}$ is computationally uniform for a polynomial $\ell(k)$, we know that $F_k(s, U_{\ell(k)})$ must be computationally indistinguishable from the uniform distribution on $\mathcal{S}_k$. The latter implies that the success probability of any collision finder A that works on the initial state $s = F_k(s_0, x), x \leftarrow U_{\ell(k)}$ can differ from the average case probability

$$\mathsf{Pr}\left[s \leftarrow \mathcal{S}_k, (x_0, x_1) \leftarrow \mathsf{A}(s) : x_0 \neq x_1,\ F_k(s, x_0) = F_k(s, x_1)\right]$$

by a negligible amount. Otherwise, we convert A to an efficient distinguisher that outputs 1 if a collision was found, and 0 otherwise. Hence, if $\{F_k\}$ is not collision resistant (w.r.t. random IV), the worst-case success is not negligible for all collision finders. □

Having an adversary that finds collisions for random IV, it is possible to construct a poly-sampleable high-entropy distribution $\mathcal{D}$ and launch the next back-dating attack:

1. Given 1^k as input, A_1 computes a list $a = [(x_1^0, x_1^1), (x_2^0, x_2^1), \ldots, (x_k^0, x_k^1)]$ of colliding pairs like in Theorem 7, computes $d = H(x_1^0 x_2^0 \ldots x_k^0)$ and outputs (d, a).
2. Given (d, a) as input, A_2 picks $b_1, \ldots, b_k \leftarrow \{0, 1\}$ and outputs $(x_1^{b_1} x_2^{b_2} \ldots x_k^{b_k}, \lfloor\rfloor)$.

The adversary $(\mathsf{A}_1, \mathsf{A}_2)$ has success probability 1 in terms of Definition 4, which means that the time-stamping scheme is insecure. Note however that this still does not mean one is able to back-date *meaningful documents* in practice.

7.3 MD-Hash Functions at the Server Side

If the server side hash function $h\colon \{0,1\}^{2k} \to \{0,1\}^k$ is implemented by using a practical MD hash function, then it is sufficient to apply the compression function f only once: $h(x_1, x_2) = f(IV, x_1 \| x_2 \| \mathsf{Padding})$, where IV denotes the standard initial value. In the proof of Theorem 7 we needed multiple applications of f to construct the high-entropy distribution $\mathcal{D}$ that was mapped to a single output value. Hence, Theorem 7 does not have practical implications for server-side hash functions.

To break h as a server-side hash function (i.e. to back-date "new" hash values), we should be able to find collisions for f, if one of the arguments x_1 or x_2 is randomly fixed, i.e. an attacker A is successful if for randomly chosen $x_1 \leftarrow \{0,1\}^k$ it is able to find a pair $x_2 \neq x_2'$ such that $f(IV, x_1 \| x_2 \| \mathsf{Padding}) = f(IV, x_1 \| x_2' \| \mathsf{Padding})$.

To our knowledge, no such attacks have been presented to MD5 or even to MD4, which means that there are no rational reasons not to use MD5 as the server-side hash function in a time-stamping scheme.

7.4 Separation of Collision Resistance and Computational Uniformity

The proof above may raise the following concern. We assumed that the hash function is broken in terms of collisions but still the compression function is computationally uniform. Hence, if collision-resistance is implied by computational uniformity, then the proof above does not make any sense. We will show that this is not the case.

Theorem 9. *There exist Merkle-Damgård hash functions that are not collision-resistant w.r.t. random initial state but have computationally uniform compression functions.*

Proof. Let $\mathcal{M}_k = \{0,1\}^{p(k)}$ and $\mathcal{S}_k = \{0,1\}^k$, where $p(k) > k$. Define the compression function $f_k\colon \mathcal{S}_k \times \mathcal{M}_k \to \mathcal{S}_k$, so that $f_k(s, x) = x_{\{1,\ldots,k\}}$, i.e. $f_k(s, x)$, independent of s, returns the first k bits of x. Obviously, the corresponding MD-hash function h_k and its internal state F_k are not collision-resistant w.r.t. random initial state, but the compression function is regular, which implies computational uniformity. □

Just for interest, we will also prove a dual separation result, which shows that computational uniformity does not follow from collision-resistance (w.r.t. random initial state) and hence it is not an ultimate design criterion for collision-free hash functions.

Theorem 10. *If there exist collision-resistant Merkle-Damgård (MD) hash functions, then there exist collision-resistant MD-hash functions in which the compression function is not computationally uniform.*

Proof. Let $f_k\colon \{0,1\}^k \times \{0,1\}^{p(k)} \to \{0,1\}^k$ be a compression function, so that the corresponding MD hash function h_k is collision-resistant w.r.t. random initial state. Define a new compression function $f'_k\colon \{0,1\}^{k+1} \times \{0,1\}^{p(k)} \to \{0,1\}^{k+1}$, so that $f'_k(b\|s,x) = 1\|f_k(s,x)$. The new compression function is collision-resistant, because every collision for h' w.r.t. initial state $b\|s$ implies a collision for h w.r.t. initial state s. However, h' is not computationally uniform, because the first output bit of F'_k is 1 with probability 1, whereas in the case of uniform distributions this probability is $\frac{1}{2}$. □

8 Conclusions and Open Questions

Collision-resistance is unnecessary if the hash-functions in time-stamping schemes are viewed as black-box functions, i.e. without considering particular design elements it is impossible to prove that collision-resistance is necessary for secure time-stamping. This also means that not every collision-finding attack is dangerous for time-stamping.

Still, we proved that for an important and wide class of practical hash functions (MD hash functions) certain multi-collision attacks also violate uPre, which we proved is a necessary and sufficient condition for client-side hash functions in time-stamping schemes (both the hash-based and for the signature based ones). We proved that uPre implies *collision resistance w.r.t. random initial state* whenever the state function is *computationally uniform*, which is a natural (though, not ultimate) design criterion for practical MD hash-functions. Heuristic arguments show that if the standard IV of a practical hash function turns out to be weak, then probably also a randomly chosen IV is weak. Still, in order to draw conclusions on the (in)security of time-stamping it is important to check whether the collision-finding attacks work in the case of random IV.

We also proved that in hash-based time-stamping, the server side hash functions may even be not one-way. Twice-compressing hash functions $h\colon \{0,1\}^{2k} \to \{0,1\}^k$ in the server side can be implemented with practical MD hash functions (like MD4, MD5, SHA-1, etc.) by calling the compression function f only once. Although we proved that the chain-resistance condition implies uPre, we cannot apply Theorem 7 because to construct a high-entropy input distribution $\mathcal{D}$ (with no output entropy) in the proof, we used multiple calls to f. So, it needs further research, whether there are efficient attacks that are able to find preimages for the *compression functions* of practical hash functions (MD4, MD5, SHA-1, etc.) in case a considerable number of input bits are (randomly) fixed. Only such attacks would be dangerous for *server-side* hash functions.

Considering very black scenarios it would be interesting to study whether secure time-stamping is possible in case *no hash function is collision-free*, i.e. if all the known practical hash functions have collisions or if one proves that the collision-resistance is not achievable. Recent results suggest that the former situation could be very likely. We conjecture that even in such a situation, secure time-stamping is still possible. Analogous to the result by Simon [14], this can probably be proven via oracle separation by constructing an oracle that provides access to a universal collision-finder but relative to which secure time-stamping schemes still exist.

References

1. Ross Anderson. The classification of hash functions. In *Proc. of the Fourth IMA Conference on Cryptography and Coding*, pp. 83–93, 1993.
2. Dave Bayer, Stuart Haber, and W.-Scott Stornetta. Improving the efficiency and reliability of digital time-stamping. In *Sequences II: Methods in Communication, Security, and Computer Science*, pp.329-334, Springer-Verlag, New York 1993.
3. Mihir Bellare and Tadayoshi Kohno. Hash Function Balance and Its Impact on Birthday Attacks. In *Advances in Cryptology – EUROCRYPT 2004*, *LNCS 3027*, pp. 401–418. 2004.
4. Ahto Buldas and Märt Saarepera. On Provably Secure Time-Stamping Schemes. In *Advances in Cryptology – Asiacrypt 2004*, *LNCS 3329*, pp. 500–514. 2004.
5. Ahto Buldas, Peeter Laud, Märt Saarepera, and Jan Willemson. Universally Composable Time-Stamping Schemes with Audit. In *Information Security Conference – ISC 2005*, LNCS 3650, pp.359–373. 2005. (IACR ePrint Archive 2005/198, 2005).
6. Stuart Haber and W.-Scott Stornetta. Secure Names for Bit-Strings. In *ACM Conference on Computer and Communications Security*, pp. 28–35, 1997.
7. Chun-Yuan Hsiao and Leonid Reyzin. Finding Collisions on a Public Road, or Do Secure Hash Functions Need Secret Coins? In *Advances in Cryptology – Crypto 2004*, *LNCS 3152*, pp. 92–105. 2004.
8. Antoine Joux. Multicollisions in Iterated Hash Functions. Application to Cascaded Constructions. In *Advances in Cryptology – CRYPTO 2004*, *LNCS 3152*, pp. 306–316, 2004.
9. Vlastimil Klima. Finding MD5 Collisions – a Toy For a Notebook. *Cryptology ePrint Archive*, Report 2005/075.
10. Vlastimil Klima. Finding MD5 Collisions on a Notebook PC Using Multi-message Modifications. *Cryptology ePrint Archive*, Report 2005/102.
11. RFC 3161: Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP).
12. Vincent Rijmen and Elisabeth Oswald. Update on SHA-1. In *Topics in Cryptology - CT-RSA 2005*, *LNCS 3376*, pp. 58–71. 2005.
13. Phillip Rogaway and Thomas Shrimpton. Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance. In *Fast Software Encryption – FSE 2004*. 2004.
14. Daniel Simon. Finding Collisions on a One-Way Street: Can Secure Hash Functions Be Based on General Assumptions? In *Advances in Cryptology – Eurocrypt 1998*, *LNCS 1403*, pp. 334–345. 1998.
15. Homepage of Surety: `www.surety.com`
16. Xiaoyun Wang, Xuejia Lai, Dengguo Feng, Hui Chen, and Xiuyuan Yu. Cryptanalysis of the Hash Functions MD4 and RIPEMD. In *Advances in Cryptology – Eurocrypt 2005*, *LNCS 3494*, pp. 1–18, 2005.
17. Xiaoyun Wang and Hongbo Yu. How to Break MD5 and Other Hash Functions. In *Advances in Cryptology – Eurocrypt 2005*, *LNCS 3494*, pp. 19–35, 2005.
18. Xiaoyun Wang, Yiqun Lisa Yin, Hongbo Yu. Finding Collisions in the Full SHA-1. *Advances in Cryptology – CRYPTO 2005*, *LNCS 3621*, pp. 17-36, 2005.
19. Xiaoyun Wang, Hongbo Yu, Yiqun Lisa Yin. Efficient Collision Search Attacks on SHA-0. In *Advances in Cryptology – CRYPTO 2005*, *LNCS 3621*, pp.1-16, 2005.