Formalizing Human Ignorance: Collision-Resistant Hashing without the Keys

Phillip Rogaway

Department of Computer Science, University of California,
Davis, California 95616, USA, and
Department of Computer Science, Faculty of Science,
Chiang Mai University, Chiang Mai 50200, Thailand
rogaway@cs.ucdavis.edu

31 January 2007

Abstract. There is a foundational problem involving collision-resistant hash-functions: common constructions are keyless, but formal definitions are keyed. The discrepancy stems from the fact that a function $H\colon \{0,1\}^* \to \{0,1\}^n$ *always* admits an efficient collision-finding algorithm, it's just that us human beings might be unable to write the program down. We explain a simple way to sidestep this difficulty that avoids having to key our hash functions. The idea is to state theorems in a way that prescribes an explicitly-given reduction, normally a black-box one. We illustrate this approach using well-known examples involving digital signatures, pseudorandom functions, and the Merkle-Damgård construction.

Key words. Collision-free hash function, Collision-intractable hash function, Collision-resistant hash function, Cryptographic hash function, Provable security.

1 Introduction

FOUNDATIONS-OF-HASHING DILEMMA. In cryptographic practice, a collision-resistant hash-function (also called a collision-free or collision-intractable hash-function) maps arbitrary-length strings to fixed-length ones; it's an algorithm $H\colon \{0,1\}^* \to \{0,1\}^n$ for some fixed n (we momentarily assume a message space of $\{0,1\}^*$). But in cryptographic theory, a collision-resistant hash-function is always *keyed*; now $H\colon \mathcal{K} \times \{0,1\}^* \to \{0,1\}^n$ where each $K \in \mathcal{K}$ names a function $H_K(\cdot) = H(K, \cdot)$. (This formalization assumes a concrete-security framework.) In this case H can be thought of as a *collection* or *family* of hash functions $H = \{H_K\colon\ K \in \mathcal{K}\}$, each *key* (or *index*) $K \in \mathcal{K}$ naming one. (Note that we call K a key but it is not secret; it will be chosen from $\mathcal{K}$ and then made public.)

Why should theoretical treatments be keyed when practical constructions are not? The traditional answer is that a rigorous treatment of collision resistance for unkeyed hash-functions just doesn't work. At issue is the fact that for *any* function $H\colon \{0,1\}^* \to \{0,1\}^n$ there is *always* a simple and compact algorithm that outputs a collision: the algorithm that has one "hardwired in." That is, by the pigeonhole principle there must be distinct strings X and X' of length at most n such that $H(X) = H(X')$, and so there's a short and fast program that outputs such an X, X'. The difficulty, of course, is that us human beings might not *know* any such pair X, X', so no one can actually write the program down.

Because of the above, what is meant when someone says that a hash function $H\colon \{0,1\}^* \to \{0,1\}^n$ is collision resistant cannot be that there is no efficient adversary that outputs a collision in H. What is meant is that there is no efficient algorithm *known to man* that outputs a collision in H. But such a statement would seem to be unformalizable—outside the realm of mathematics. One cannot hope to construct a meaningful theory based on what humankind currently does or does not know. Regarding a hash function like SHA-1 [22] as a random element from a family of hash functions has been the traditional way out of this quandary.

Let us call the problem we've been discussing the *foundations-of-hashing dilemma.* The question is how to state definitions and theorems dealing with collision-resistant hashing in a way that makes sense mathematically, yet accurately reflects cryptographic practice. The treatment should respect our understanding that what makes a hash function collision resistant is *humanity's* inability to find a collision, not the computational complexity of printing one.

Our contributions. First, we bring the foundations-of-hashing dilemma out into the open. To the best of our knowledge, the problem has never received more than passing mention in any paper. Second, we resolve the dilemma. We claim that an answer has always been sitting right in front of us, that there's never been any real difficulty with providing a rigorous treatment of unkeyed collision-resistant hash-functions. Finally, we reformulate in a significantly new way three fundamental results dealing with collision-resistant hashing.

Suppose a protocol Π uses a collision-resistant hash-function H. Conventionally, a theorem would be given to capture the idea that the existence of an effective adversary A against Π implies the existence of an effective adversary C against H. But this won't work when we have an unkeyed $H\colon \{0,1\}^* \to \{0,1\}^n$ because such an adversary C will *always* exist. So, instead, the theorem statement will say that there is an explicitly given reduction: given an adversary A against Π there is a corresponding, explicitly-specified adversary C, as efficient as A, for finding collisions in H. So if someone knows how to break the higher-level protocol Π then they know how to find collisions in H; and if nobody can find collisions in H then nobody can break Π. In brief, our solution to the foundations-of-hashing dilemma is to recast results so as to assert the existence of an explicitly given reduction. We call this the *human-ignorance* (or *explicit-reduction*) approach.

We illustrate the approach with three well-known examples. The first is the hash-then-sign paradigm, where a signature scheme is constructed by hashing a message and then applying an "inner" signature to the result. Our second example is the construction of an arbitrary-input-length PRF by hashing and then applying a fixed-input-length PRF. Our third example is the Merkle-Damgård construction, where a collision-resistant compression-function is turned into a collision-resistant hash-function. In all cases we will give a simple theorem that captures the security of the construction despite the use of an unkeyed formalization for the underlying hash function.

We provide a concrete-security treatment for all the above. Giving our hash functions a security parameter and then looking at things asymptotically would only distance us, we feel, from widely-deployed, real-world hash-functions. That said, we will also point out that unkeyed hash-functions work fine in the asymptotic setting for the case of *uniform* adversaries. One eliminates keys but not the security parameter, making it the length of the hash-function's output.

Related work. The rigorous treatment of collision-resistant hash-functions begins with Damgård [7]. A concrete-security treatment of these objects is given by Bellare, Rogaway, and Shrimpton [3, 27]. Practical and widely-deployed cryptographic hash-functions were first developed by Rivest [26], and later constructions, like SHA-1 [22], have followed his approach. Bellare et al.'s [1, Theorem 4.2] is an early example of an explicitly constructive provable-security theorem-statement. Using a simulator to model what an adversary must know or be able to do is from Goldwasser, Micali, and Rackoff [14], while black-box reductions come from Goldreich, Krawczyk, and Oren [11, 12, 23].

Moving closer in, it is well-understood that one can rephrase provable-security results as assertions about explicitly given reductions, and several researchers have noted, with varying degrees of explicitness, that this can be used to make formal sense of unkeyed hash-functions. The most explicit work in this direction is from Stinson [29, 30]. Given a fixed hash-function $f\colon X \to Y$ and a subroutine *FindPreimage* for inverting f, Stinson constructs a collision-finding adversary C for f, and he gives theorems specifying the effectiveness of C in terms of the effectiveness of *FindPreimage*. Stinson explains [30, page 268] that *there seems to be no satisfactory formalization of the idea of collision resistance for a fixed hash function, because there always exists an algorithm that simply outputs a collision (if one exists). Alternatively, one can talk about infeasibility of problems in an asymptotic context, but this requires having an infinite "keyed" family of functions .... We prefer to instead study reductions among the different problems. We can study reductions without worrying about having to define terms such as "hard to solve" or "infeasible", even if the hash function under consideration is fixed.* Following Stinson [29], Laccetti and Schmid similarly state their related hash-function results in reduction-based terms [17].

More implicit examples of using reductions to deal with unkeyed collision-resistant hash-function can be found in Brown [5, see footnote 10] and Devanbu et al. [10], both of which prove the security of a protocol that employs an unkeyed hash-function by constructively transforming a successful adversary against it into a successful collision-finding one. Indeed using such a transformation to evidence a hash-function-based protocol's security goes back to Merkle [18, 20]. The option of speaking about reductions as a way of not having to key a hash function is also hinted at in recent work of Halevi and Krawczyk [16, footnote 5].

In summary, elements of our approach can be found scattered in the literature, but nowhere are the ideas carefully developed and explained. (Even Stinson sometimes falls back to the problematic language of "*if* ⟨something⟩ *then* ⟨there exists an efficient collision-finding algorithm⟩" [30, Theorem 4.1] and [29, Theorems 3.1 and 4.1].) In this current paper we attempt to clear up these ideas. We aim to illustrate and articulate the power and applicability of the human-ignorance approach.

2 Keyed Hash-Functions

We first give a *conventional* definition, in the concrete-security setting, for a (keyed) collision-resistant hash-function. Beginning with the syntax, a *keyed hash-function* is a pair of algorithms $(\mathcal{K}, H)$, the first probabilistic and the second deterministic. Algorithm $\mathcal{K}$, the *key-generation algorithm*, takes no input and produces a string K, the *key*. As a special case, $\mathcal{K}$ uniformly samples from a finite set, the *key space*, also denoted $\mathcal{K}$. Algorithm H takes as input a string K, the *key*, and a string X, the *message*, and it outputs a string of some fixed length n, *the output length*, or the distinguished value $\perp$. We write $H_K(X)$ for $H(K, X)$. We assume there is a set $\mathcal{X}$, the *message space*, such that $H_K(X) = \perp$ iff $X \notin \mathcal{X}$. We assume that $\mathcal{X}$ contains some string of length greater than n and that $X \in \mathcal{X}$ implies every string of length $|X|$ is in $\mathcal{X}$. We write a hash function as $H\colon \mathcal{K} \times \mathcal{X} \to \{0,1\}^n$ instead of saying "the keyed hash-function $(\mathcal{K}, H)$ with message space $\mathcal{X}$ and output length n." Hash functions and all other algorithms in this paper are given by code relative to some fixed and reasonable encoding.

We define hash functions as algorithms, not functions, to enable providing them as input to other algorithms and speaking of their computational complexity. But a hash function $H\colon \mathcal{K}\times\mathcal{X} \to \{0,1\}^n$ induces a function H from $\mathcal{K}\times\mathcal{X}$ to $\{0,1\}^n$, where $\mathcal{K}$ is now the support of the key-generation algorithm, and usually it is fine to regard the hash function as being this function.

To measure the collision-resistance of hash function $H\colon \mathcal{K} \times \mathcal{X} \to \{0,1\}^n$ let C (for *collision-finder*) be an adversary, meaning, in this case, an algorithm that takes in one string (the key) and outputs a pair of strings (the purported collision). We let the *advantage* of C in finding collisions in H be the real number

$$\mathbf{Adv}_H^{\mathrm{coll}}(C) \quad = \quad \Pr[K \xleftarrow{\$} \mathcal{K};\, (X, X') \xleftarrow{\$} C(K):\; X \neq X' \text{ and } H_K(X) = H_K(X')]$$

that measures the chance that C finds a collision in $H_K = H(K, \cdot)$ if a random key K is provided to it. Above and henceforth we assume that an adversary will never output a string outside the message space $\mathcal{X}$ of the hash function it is attacking (that is, $H_K(X) = H_K(X') = \perp$ never counts as a collision).

As usual, an advantage of 1 means that C does a great job (it always finds a collision) while an advantage of 0 means that C does a terrible job (it never finds a collision). Since we are in the concrete-security setting we do not define any absolute (yes-or-no) notion for H being coll-secure; instead, we regard a hash function H as good only to the extent that reasonable adversaries C can obtain only small advantage $\mathbf{Adv}_H^{\mathrm{coll}}(C)$. In order to obtain a useful theory, "reasonable" and "small" need never be defined.

Trying to regard functions like SHA-1 as keyed. How can a real-world hash-function like SHA-1 be seen as fitting into the framework above? One possibility is that the intended key is the initial chaining vector; the constant K = 67452301 EFCDAB89 98BADCFE 10325476 C3D2E1F0 can be regarded as the key. In this case the key space is $\mathcal{K} = \{0,1\}^{160}$ and what NIST did in choosing SHA-1 was to randomly sample from this set. The problem with this viewpoint is that, first of all, NIST never indicated that they did any such thing. Indeed the constant K above does not "look" random (whatever that might mean), and it seems as though the specific constant should hardly matter: likely any method that would let one construct collisions in SHA-1 with respect to the actual K-value would work for other K-values, too.

A second way one might regard SHA-1 as keyed is to say that NIST, in designing SHA-1, considered some universe of hash functions $\{H_K :\ K \in \mathcal{K}\}$ and randomly selected this one hash function, SHA-1, from it. But, once again, NIST never indicated that they did any such thing; all we know is that they selected this *one* hash function. And it's not clear what $\mathcal{K}$ would even be in this case, or what H_K would be for "other" functions in the family.

Fundamentally, both explanations are unappealing. They make random sampling a crucial element to a definition when no random sampling ostensibly took place. They disregard the basic intuition about what SHA-1 is supposed to be: a fixed map that people shouldn't be able to find collisions in. And they distance the definition from the elegantly simple goal of the cryptanalyst: publish a collision for the (one) function specified by NIST.

3 Unkeyed Hash-Functions

An *unkeyed hash-function* is a deterministic algorithm H that takes as input a string X, the *message*, and outputs a string of some fixed length n, *the output length*, or the distinguished value $\perp$. The *message space* of H is the set $\mathcal{X} = \{X \in \{0,1\}^* :\ H(X) \neq \perp\}$. We assume that $\mathcal{X}$ contains some string of length greater than n and that $X \in \mathcal{X}$ implies every string of length $|X|$ is in $\mathcal{X}$. We will write a hash function as $H\colon \mathcal{X} \to \{0,1\}^n$, or simply H, instead of saying "the unkeyed hash-function H with message space $\mathcal{X}$ and output length n."

Let C be an adversary for attacking $H\colon \mathcal{X} \to \{0,1\}^n$, meaning an algorithm that, with no input, outputs a pair of strings X and X' in $\mathcal{X}$. We let the *advantage* of C in finding collisions in H be the real number

$$\mathbf{Adv}_H^{\mathrm{col}}(C) \quad = \quad \Pr[(X, X') \xleftarrow{\$} C :\; X \neq X' \text{ and } H(X) = H(X')]$$

that measures the chance that C finds a collision. Note the spelling of superscript *col* verses the earlier *coll* (the number of l's is the number of arguments to H).

Following the discussion in the Introduction, we observe that for any unkeyed hash-function H there is an efficient algorithm C (it runs in cn time and takes cn bits to write down, for some small c) for which $\mathbf{Adv}_H^{\mathrm{col}}(C) = 1$. We're not going to let that bother us.

4 Three Styles of Provable-Security Statements

PROVABLE-SECURITY FORMULATIONS. Let Π be a cryptographic protocol that employs a (keyed or unkeyed) hash function H. Imagine, for now, that H is the only cryptographic primitive that Π employs. To prove security of Π using a reduction-based approach and assuming the collision-resistance of H one would typically make a theorem statement that could be paraphrased like this:

> existential form (C0): If there's an effective algorithm A for attacking protocol Π then there's an effective algorithm C for finding collisions in H.

When cryptographic reductions were first introduced [13], theorems were stated with this kind of existential-only guarantee. To this day, people almost always state their provable-security results in such a manner.

Formalizing statement C0 works fine in the keyed setting but not in the unkeyed one, because, there, the conclusion vacuously holds. But in the unkeyed setting we can switch to a theorem statement that could be paraphrased as:

> code-constructive form (C1): If you know an effective algorithm A for attacking protocol Π then you know an effective algorithm C for finding collisions in H.

We are asserting the existence of a known "compiler" that turns A into C. Now your belief in the security of Π stems from the fact that if some human being can break Π then he can exhibit collisions in H. Statement C1 can be regarded as a constructive version of C0. Continuing on this trajectory, we could say that it's enough to have access to A's functionality, you don't actually need the code:

> blackbox-constructive form (C2): If you possess effective means A for attacking protocol Π then you possess effective means C for finding collisions in H.

Here, "possessing effective means" might mean owning a tamper-resistant device, or being able to run some big executable program, or it might even mean having a brain in your head that does some task well. Possessing effective means does not imply knowing the internal structure of those means; I might not know what happens within the tamper-resistant device, the big program, or in my own brain. Statement C2 is stronger than C1 because knowledge of an algorithm implies access to its functionality, but having access to an algorithm's functionality does not imply knowing how it works.

The main observation in this paper is that, in the concrete-security setting, it's easy to give provable-security results involving unkeyed hash-functions as long as you state your results in the code-constructive (C1) or blackbox-constructive (C2) format. In the asymptotic setting, all three formats work fine as long as you stick to uniform adversaries.

In high-level expositions, provable-security results are often summarized in what would appear to be a code-constructive or blackbox-constructive manner; people say things like "our result shows that if someone could break this signature scheme then he could factor large composite numbers."

But when we write out our theorem statements, it has been traditional to adopt the existential format. Usually the proof is constructive but the theorem statement is not.

FORMALIZING C1 AND C2. In the next section we'll formalize C1, in an example setting, by asking for an explicitly given algorithm $\mathcal{C}$ that, given the code for A (and also H and Π) provides us our collision finder C. We likewise formalize C2, in three example settings, by asking for an explicitly given algorithm C that, given black-box access to A (and also H and Π) is itself our collision finder.

When an algorithm C has black-box access to an algorithm F we write the latter as a subscript or superscript, C_F or C^F. (Oracles are conventionally written as superscripts, but writing all of an algorithm's oracles this way is less readable when the algorithm's job is, say, to distinguish one particular oracle from another.) We do not allow for C to see or control the internal coins of F; when C runs F, the latter's coins are random and externally provided. We do not object to C resetting F, so long as fresh (secret) coins are issued to it each time that it is run.

RESOURCE ACCOUNTING. Let F be an algorithm (possibly stateful, probabilistic, and itself oracle-querying). The algorithm F might be provided as an oracle to some other algorithm. Let $t_F(\ell)$ be the maximum amount of time (in a conventional, non-blackbox model) to compute F on strings that total ℓ or fewer bits (but count the empty string ε as having length 1). We simplify to t_F for an overall maximum. Let ℓ_F be the maximum of the total number of bits read or written by F (over F's input, output, oracle queries, and their responses) (but regard ε as having length 1). Let q_F be the maximum number of queries made by F before it halts (but no less than 2, to simplify theorem statements). We assume that all algorithms halt after some bounded amount of time. When an algorithm A calls out to an oracle for F, we charge to A the time to compute F (even though the internal computation of F seems, to the caller, unit time).

As an example of the above, for a keyless hash-function $H\colon \{0,1\}^* \to \{0,1\}^n$ we have that $t_H(\ell)$ is the maximal amount of time to compute H on any sequence of inputs $X_1, \ldots, X_q$ comprising ℓ total bits (where $X_i = \varepsilon$ counts as 1-bit). As a second example, for an adversary A attacking a signature scheme, the number ℓ_A includes the length of the public-key provided to A, the length of the signing queries that A asks, the length of the signatures A gets in response, and the length of A's forgery attempt. Since we insisted that A is bounded-time, if it is provided an overly-long input or oracle response, it should only read (and is only charged for) a bounded-length prefix.

5 Hash-then-Sign Signatures

The usual approach for digital signatures, going back to Rabin [24], is to sign a message by first hashing it and then calling an underlying signature scheme. The purpose of this *hash-then-sign* approach is two-fold. First, it extends the domain of the "inner" signature scheme from $\{0,1\}^n$ to $\{0,1\}^*$ (where the hash-function's output is n bits). Second, it may improve security by obscuring the algebraic structure of the inner signature scheme. We focus only on the first of these intents, establishing the folklore result that the hash-then-sign paradigm securely extends the domain of a signature scheme from $\{0,1\}^n$ to $\{0,1\}^*$. Our purpose is not only to prove this (admittedly simple) result, but also to illustrate the human-ignorance approach for dealing with collision-resistant hash-functions.

First we establish the notation, using concrete-security definitions. A *signature scheme* is a three-tuple of algorithms $\Pi = (\mathit{Gen}, \mathit{Sign}, \mathit{Verify})$. Algorithm Gen is a probabilistic algorithm that, with no input, outputs a pair of strings (PK, SK). (One could, alternatively, assume that Gen takes input of a security parameter k.) Algorithm Sign is a probabilistic algorithm that, on input (SK, X), outputs either a string $\sigma \xleftarrow{\$} \mathit{Sign}(SK, X)$ or the distinguished value $\perp$. We require the existence of a *message space* $\mathcal{X} \subseteq \{0,1\}^*$ such that, for any SK, we have that $\sigma \xleftarrow{\$} \mathit{Sign}(SK, X)$ is a string exactly when $X \in \mathcal{X}$. We insist that $\mathcal{X}$ contain all strings of a given length if it contains any string of that length. Algorithm *Verify* is a deterministic algorithm that, on input (PK, X, σ), outputs a bit. We require that if $(PK, SK) \xleftarrow{\$} \mathit{Gen}$ and $X \in \mathcal{X}$ and $\sigma \xleftarrow{\$} \mathit{Sign}(SK, X)$ then $\mathit{Verify}(PK, X, \sigma) = 1$. We sometimes write $\mathit{Sign}_{SK}(X)$ and $\mathit{Verify}_{PK}(X, \sigma)$ instead of $\mathit{Sign}(SK, X)$ and $\mathit{Verify}(PK, X, \sigma)$.

Let B be an adversary and $\Pi = (\mathit{Gen}, \mathit{Sign}, \mathit{Verify})$ a signature scheme. Define $\mathbf{Adv}_{\Pi}^{\mathrm{sig}}(B) = \Pr[(PK, SK) \xleftarrow{\$} \mathit{Gen} :\ B^{\mathit{Sign}_{SK}(\cdot)}(PK) \text{ forges}]$ where B is said to *forge* if it outputs a pair (X, σ) such that $\mathit{Verify}_{PK}(X, \sigma) = 1$ and B never asked a query X during its attack.

We now define the hash-then-sign construction. Let $H\colon \{0,1\}^* \to \{0,1\}^n$ be an unkeyed hash-function and let $\Pi = (\mathit{Gen}, \mathit{Sign}, \mathit{Verify})$ be a signature scheme with message space of at least $\{0,1\}^n$. Define from these primitives the signature scheme $\Pi^H = (\mathit{Gen}, \mathit{Sign}^H, \mathit{Verify}^H)$ by setting $\mathit{Sign}^H_{SK}(X) = \mathit{Sign}_{SK}(H(X))$ and $\mathit{Verify}^H_{PK}(X, \sigma) = \mathit{Verify}_{PK}(H(X), \sigma)$. The message space for Π^H is $\{0,1\}^*$.

We are now ready to state a first theorem that describes the security of the hash-then-sign paradigm.

Theorem 1 [hash-then-sign, unkeyed, concrete, C1-form] *There exist algorithms $\mathcal{B}$ and $\mathcal{C}$, explicitly given in the proof of this theorem, such that for any unkeyed hash-function $H\colon \{0,1\}^* \to \{0,1\}^n$, signature scheme $\Pi = (\mathit{Gen}, \mathit{Sign}, \mathit{Verify})$ with message space at least $\{0,1\}^n$, and adversary A, adversaries $B = \mathcal{B}(\langle A, H\rangle)$ and $C = \mathcal{C}(\langle A, H, \Pi\rangle)$ satisfy*

$$\mathbf{Adv}_{\Pi}^{\mathrm{sig}}(B) + \mathbf{Adv}_{H}^{\mathrm{col}}(C) \quad \geq \quad \mathbf{Adv}_{\Pi^H}^{\mathrm{sig}}(A)\,.$$

Adversary B runs in time at most $t_A + t_H(\ell_A) + t_{\mathit{Sign}}(nq_A) + c(\ell_A + nq_A)$ and asks at most q_A queries entailing at most $\ell_A + n$ bits. Adversary C runs in time at most $t_A + t_{\mathit{Gen}} + t_H(\ell_A) + t_{\mathit{Sign}}(nq_A{+}n) + c(\ell_A + nq_A)\lg(q_A)$. Functions $\mathcal{B}$ and $\mathcal{C}$ run in time c times the length of their input. The value c is an absolute constant implicit in the proof of this theorem. $\diamondsuit$

The theorem says that if you know the code for A, H, and Π then you know the code for B and C. You know that code because it's given by *reduction functions* $\mathcal{B}$ and $\mathcal{C}$. These reduction functions are explicitly specified in the proof of the theorem. Reduction function $\mathcal{B}$ takes in an encoding of A and H and outputs the code for adversary B. Reduction function $\mathcal{C}$ takes in an encoding of A, H, and $\Pi = (\mathit{Gen}, \mathit{Sign}, \mathit{Verify})$ and outputs the code for adversary C.

One might argue that we don't really care that B is constructively given—we might have demanded only that it exist whenever A does. But it seems simpler and more natural to demand that both adversaries B and C be constructively given when we are demanding that one adversary be. Besides, it is nicer to conclude *you know a good algorithm to break* Π than to conclude *there exists a good algorithm to break* Π; it would, in fact, be an unsatisfying proof that actually gave rise to a nonconstructive attack on the inner signature scheme Π.

Theorem 1 does not capture statement C2 because access to the functionality of adversary A might be more limited than possessing its code. To capture the intent of statement C2, we can strengthen our theorem as follows:

Theorem 2 [hash-then-sign, unkeyed, concrete, C2-form] *There exist adversaries B and C, explicitly given in the proof of this theorem, such that for any unkeyed hash-function $H\colon \{0,1\}^* \to \{0,1\}^n$, signature scheme $\Pi = (\mathit{Gen}, \mathit{Sign}, \mathit{Verify})$ with message space at least $\{0,1\}^n$, and adversary A, we have that*

$$\mathbf{Adv}_{\Pi}^{\mathrm{sig}}(B_{A,H}) + \mathbf{Adv}_{H}^{\mathrm{col}}(C_{A,H,\Pi}) \quad \geq \quad \mathbf{Adv}_{\Pi^H}^{\mathrm{sig}}(A)\,. \tag{1}$$

Adversary B runs in time at most $t_A + t_H(\ell_A) + t_{Sign}(nq_A) + c(\ell_A + nq_A)$ and asks at most q_A queries entailing at most $\ell_A + n$ bits. Adversary C runs in time at most $t_A + t_{Gen} + t_H(\ell_A) + t_{Sign}(nq_A+n) + c(\ell_A + nq_A)\lg(q_A)$. The value c is an absolute constant implicit in the proof of this theorem. $\diamondsuit$

The theorem asserts the existence of an explicitly known forging adversary B (for attacking Π) and an explicitly known collision-finding adversary C (for attacking H), at least one of which must do well if the original adversary A does well (in attacking Π^H). Algorithm C employs A, as well as H, *Gen*, *Sign*, and *Verify*, in a black-box manner. (Writing $\Pi = (Gen, Sign, Verify)$ as a subscript to C means giving each component algorithm as an oracle.) We may not care that the dependency on H, *Gen*, *Sign*, and *Verify* is black-box, for there is no question there about having access to the code, but it seems simpler to demand that all dependencies be black-box when we require one to be. As with Theorem 1, the final set of lines in Theorem 2 explain that the time and communications complexity of algorithms B and C is insignificantly more than that of A.

Proof of of Theorem 2 and then Theorem 1: In the following exposition, computations of A, H, *Gen*, *Sign*, and *Verify* are done via oracle queries.

Construct collision-finding adversary $C_{A,H,\Pi}$ as follows. First it calls *Gen* to determine an output $(PK, SK) \xleftarrow{\$} Gen$. Then it calls adversary A on input PK. When A makes its i^{th} query, X_i, a request to sign the string X_i, algorithm C calls H to compute $x_i = H(X_i)$, it calls *Sign* on input x_i to compute $\sigma_i \xleftarrow{\$} Sign_{SK}(x_i)$, and it returns σ_i in answer to A's query. When A halts with output (X_*, σ_*) algorithm C invokes H to compute $x_* = H(X_*)$. If x_* is equal to x_i for some prior i, and $X_* \neq X_i$, then algorithm C outputs the collision (x_i, x_*) and halts. Otherwise, algorithm C fails; it outputs an arbitrary pair of strings. The reader can check that C has the claimed time complexity. The log-term accounts for using a binary search tree, say, to lookup if x_* is equal to some prior x_i.

Construct forging-adversary $B^{Sign}_{A,H}(PK)$ as follows. Algorithm B, which is provided a string PK, runs black-box adversary A on input of PK. When A makes its i^{th} query, X_i, a request for a signature of X_i, algorithm B uses its oracle H to compute $x_i = H(X_i)$. It then uses its *Sign*-oracle to compute $\sigma_i \leftarrow Sign(x_i)$. It returns σ_i in answer to the adversary A. When A halts with output (X_*, σ_*) algorithm B uses its H-oracle to compute $x_* = H(X_*)$. Algorithm B halts with output (x_*, σ_*). The reader can check that B has the claimed time and communications complexity. (The t_{Sign} term is because of our convention to consistently charge algorithms for their oracle calls.)

We must show (1). Let a be the probability that A, in carrying out its attack in the experiment defining $\mathbf{Adv}^{\text{sig}}_{\Pi^H}(A)$, outputs a valid forgery (X_*, σ_*) where $H(X_*) = H(X_i)$ for some i. Let b be the probability that A, in carrying out its attack in the experiment defining $\mathbf{Adv}^{\text{sig}}_{\Pi^H}(A)$, outputs a valid forgery (X_*, σ_*) where $H(X_*) \neq H(X_i)$ for all i. Then $a + b = \mathbf{Adv}^{\text{sig}}_{\Pi^H}(A)$. We also have that $\mathbf{Adv}^{\text{col}}_{H}(C_{A,H,\Pi}) \geq a$ and $\mathbf{Adv}^{\text{sig}}_{\Pi}(B_{A,H}) \geq b$, establishing Theorem 2.

As for Theorem 1, the reduction functions $\mathcal{B}$ and $\mathcal{C}$ are what is spelled out in the definition of B and C, above, except that computation by code replaces oracle invocations. (One can now see why we have selected our earlier conventions about how to charge-out oracle calls: it is convenient that it has no impact on the running time if one imagines calling an oracle for H, say, verses running that code oneself.) It is a simple, linear-time algorithm that takes in A and H (which are code) and outputs B (which is also code), or that produces C from A, H and each component of Π. ∎

For the remainder of our examples we will use the stronger, black-box style of theorem statement corresponding to Statement C2 and Theorem 2.

6 Hash-then-PRF

As a second example of using our framework we consider a symmetric-key analog of hash-then-sign, where now we aim to extend the domain of a pseudorandom function (PRF) from $\{0,1\}^n$ to $\{0,1\}^*$. The algorithm, which we consider to be folklore, is to hash the message X and then apply a PRF, setting $F_K^H(X) = F_K(H(X))$ where $H\colon \{0,1\}^* \to \{0,1\}^n$ is the hash function and $F\colon \mathcal{K} \times \{0,1\}^n \to \{0,1\}^m$ is the PRF. A special case of this construction is using a hash function $H\colon \{0,1\}^* \to \{0,1\}^n$ and an n-bit blockcipher to make an arbitrary-input-length message authentication code (MAC). A second special-case is using a hash function $H\colon \{0,1\}^* \to \{0,1\}^{2n}$ and the two-fold CBC MAC of an n-bit blockcipher to make an arbitrary-input-length MAC.

First the definitions, following works like [2]. An (m-bit output) pseudorandom function (PRF) is an algorithm $F\colon \mathcal{K} \times \mathcal{X} \to \{0,1\}^m$ where $\mathcal{K}$ and $\mathcal{X}$ are sets of strings. We assume that there is an algorithm associated to F, which we also call $\mathcal{K}$, that outputs a random element of $\mathcal{K}$. For $\mathcal{X}, \mathcal{Y} \subseteq \{0,1\}^*$ and $\mathcal{Y}$ finite, let $\mathrm{Func}(\mathcal{X}, \mathcal{Y})$ be the set of all functions from $\mathcal{X}$ to $\mathcal{Y}$. We endow this set with a distribution by saying that, for each $x \in \mathcal{X}$, the value of $f(x)$ is uniformly sampled from $\mathcal{Y}$. For a PRF $F\colon \mathcal{K} \times \mathcal{X} \to \{0,1\}^m$ let $\mathbf{Adv}_F^{\mathrm{prf}}(B) = \Pr[K \stackrel{\$}{\leftarrow} \mathcal{K} :\ B^{F_K(\cdot)} \Rightarrow 1] - \Pr[f \stackrel{\$}{\leftarrow} \mathrm{Func}(\mathcal{X}, \{0,1\}^m) :\ B^{f(\cdot)} \Rightarrow 1]$. The following quantifies the security of the hash-then-PRF construction F^H.

Theorem 3 [hash-then-PRF, unkeyed, concrete, C2-form] *There exist adversaries B and C, explicitly given in the proof of this theorem, such that for any unkeyed hash-function $H\colon \{0,1\}^* \to \{0,1\}^n$, pseudorandom function $F\colon \mathcal{K} \times \{0,1\}^n \to \{0,1\}^m$, and adversary A,*

$$\mathbf{Adv}_F^{\mathrm{prf}}(B_{A,H}) + \mathbf{Adv}_H^{\mathrm{col}}(C_{A,H,F}) \quad \geq \quad \mathbf{Adv}_{F^H}^{\mathrm{prf}}(A)\,. \tag{2}$$

Adversary B runs in time at most $t_A + t_H(\ell_A) + t_F(nq_A) + c(\ell_A + nq_A + mq_A)$ and asks at most q_A queries entailing at most ℓ_A bits. Adversary C runs in time at most $t_A + t_H(\ell_A) + c(\ell_A + nq_A + mq_A + t_{\mathcal{K}})\lg(q_A)$. The value c is an absolute constant implicit in the proof of this theorem. $\diamondsuit$

Proof: Construct collision-finding algorithm $C_{A,H,F}$ as follows. The algorithm runs adversary A, which is given by an oracle. When A makes its i^{th} oracle query, X_i, algorithm C uses its H oracle to compute $x_i = H(X_i)$ and then, if $x_i \neq x_j$ for all $j < i$, adversary C returns a random $y_i \stackrel{\$}{\leftarrow} \{0,1\}^m$ in response to A's query. If $x_i = x_j$ for some $j < i$, adversary C returns $y_i = y_j$. When A finally halts, outputting a bit a, algorithm C ignores a and looks to see if there were distinct queries X_i and X_j made by A such that $x_i = x_j$. If there is such a pair, algorithm C outputs an arbitrary such pair (X_i, X_j) and halts. Otherwise, algorithm C *fails* and outputs an arbitrary pair of strings. The time of C is at most that which is stated in the theorem. Note that C does not actually depend on F beyond employing the values n and m.

Construct distinguishing algorithm $B_{A,H}^f$ as follows. It begins by running algorithm A, which is given by an oracle. When A makes its i^{th} query, X_i, algorithm B computes $x_i = H(X_i)$ and then asks its f oracle x_i, obtaining return value $y_i = f(x_i)$. Algorithm B returns y_i to A. When A finally halts, outputting a bit a, algorithm B halts without output a. The resources of B are as given by the theorem statement.

We have that $\mathbf{Adv}_{F^H}^{\mathrm{prf}}(A) - \mathbf{Adv}_F^{\mathrm{prf}}(B_{A,H}) = \Pr[A^{F_K^H} \Rightarrow 1] - \Pr[A^R \Rightarrow 1] - \Pr[B_{A,H}^{F_K} \Rightarrow 1] + \Pr[B_{A,H}^{\rho} \Rightarrow 1]$ where $\rho \stackrel{\$}{\leftarrow} \mathrm{Func}(n,m)$ and $R \stackrel{\$}{\leftarrow} \mathrm{Func}(\{0,1\}^*, m)$ and $K \stackrel{\$}{\leftarrow} \mathcal{K}$. Now, from our definition of B, the first and third addend are equal, $\Pr[A^{F_K^H} \Rightarrow 1] = \Pr[B_{A,H}^{F_K} \Rightarrow 1]$, and so $\mathbf{Adv}_{F^H}^{\mathrm{prf}}(A) - \mathbf{Adv}_F^{\mathrm{prf}}(B_{A,H}) = \Pr[B_{A,H}^{\rho} \Rightarrow 1] - \Pr[A^R \Rightarrow 1]$.

Let C be the event that, during B's attack, there are distinct queries X_i and X_j made by B such that $H(X_i) = H(X_j)$. Let $c = \Pr[\mathsf{C}]$ where the probability is taken over B's oracle being a random function $\rho \xleftarrow{\$} \text{Func}(n, m)$. Observe that, from C's definition, $c = \mathbf{Adv}_H^{\text{col}}(C_{A,H,F})$. Now note that $\Pr[B_{A,H}^{\rho} \Rightarrow 1] - \Pr[A^R \Rightarrow 1] \le c$ because in the second experiment a random m-bit value is returned for each new X_i and in the first experiment a random m-bit value is returned for each new X_i *except* when $x_i = H(X_i)$ is identical to a prior $x_j = H(X_j)$. This establishes Equation (2). ▮

A result similar to Theorem 3, but for MACs instead of PRFs, can easily be established. That is, if $H\colon \{0,1\}^* \to \{0,1\}^n$ is an unkeyed hash-function and MAC: $\{0,1\}^n \to \{0,1\}^m$ is a good MAC [2] then MAC^H is a good MAC. Here, as before, MAC^H is defined by $\text{MAC}_K^H(M) = \text{MAC}_K(H(M))$. The weaker assumption (F is a good MAC instead of a good PRF) suffices to get the weaker conclusion (F^H is a good MAC).

7 Merkle-Damgård without the Keys

We adapt the Merkle-Damgård paradigm [8, 19] to the unkeyed hash-function setting. To get a message space of $\{0,1\}^*$ and keep things simple we adopt the length-annotation technique known as Merkle-Damgård *strengthening*. There are actually several variants of this; for a fixed-key, reduction-based treatment, up until the actual theorem statement, for a different version of Merkle-Damgård strengthening, see Stinson [29, Section 4].

First we define the mechanism. Let $H\colon \{0,1\}^{b+n} \to \{0,1\}^n$ be an unkeyed hash-function, called a *compression function*, and define from it the unkeyed hash-function $H^*\colon \{0,1\}^* \to \{0,1\}^n$ as follows. On input $X \in \{0,1\}^*$, algorithm H^* partitions $\mathsf{pad}(X) = X \parallel 0^p \parallel [|X|]_b$ into b-bit strings $X_1 \cdots X_m$ where $p \ge 0$ is the least nonnegative number such that $|X| + p$ is a multiple of b and where $[|X|]_b$ is $|X| \bmod 2^b$ encoded as a b-bit binary number. Then, letting $Y_0 = 0^n$, say, define $Y_i = H(X_i \parallel Y_{i-1})$ for each $i \in [1\,..\,m]$ and let $H^*(X)$ return Y_m. Note that $\mathbf{Adv}_H^{\text{col}}(C) = \Pr[(X, X') \xleftarrow{\$} C\colon\ X \neq X' \text{ and } H(X) = H(X')]$ where C must output $X, X' \in \{0,1\}^{b+n}$. We now show that if H is a collision-resistant compression-function then H^* is a collision-resistant hash-function.

Theorem 4 [Merkle-Damgård, unkeyed, concrete, C2-form] *Fix positive numbers b and n. There exists an adversary C, explicitly given in the proof of this theorem, such that for any unkeyed hash-function $H\colon \{0,1\}^{b+n} \to \{0,1\}^n$ and any adversary A that outputs a pair of strings each of length less than 2^b,*

$$\mathbf{Adv}_H^{\text{col}}(C_{A,H}) \quad \ge \quad \mathbf{Adv}_{H^*}^{\text{col}}(A)\,. \tag{3}$$

Adversary C runs in time at most $t_A + (\ell_A/b + 4)t_H + c(\ell_A + b + n)$. The value c is an absolute constant implicit in the proof of this theorem. $\diamondsuit$

Proof: Construct the collision-finding adversary $C_{A,H}$ as follows. It runs the adversary A, which requires no inputs and halts with and output X, X', each string having fewer than 2^b bits. Swap X and X', if necessary, so that X is at least as long as X'. Adversary C then computes $X_1 \cdots X_m = \mathsf{pad}(X)$ and $X'_1 \cdots X'_{m'} = \mathsf{pad}(X')$ where each X_i and X'_j is b-bits long. Using its H-oracle, adversary C computes Y_i-values by way of $Y_0 = 0^n$ and, for each $i \in [1\,..\,m]$, $Y_i = H(X_i \parallel Y_{i-1})$. It similarly computes Y'_j-values, defining $Y'_0 = 0^n$ and $Y'_j = H(X'_j \parallel Y'_{j-1})$ for each $j \in [1..m']$. Now if $X = X'$ or $Y_m \neq Y'_{m'}$ then adversary C fails, outputting an arbitrary pair of strings. Otherwise, adversary C computes the largest value $i \in [1\,..\,m]$ such that $Y_i = Y'_{i-\Delta}$ but $X_i \parallel Y_{i-1} \neq$

$X'_{i-\Delta} \parallel Y'_{i-1-\Delta}$ where $\Delta = m - m'$. (We prove in a moment that such an i exists.) Adversary C outputs the pair of strings $(X_i \parallel Y_{i-1},\ X'_{i-\Delta} \parallel Y'_{i-1-\Delta})$, which collide under H.

We must show that this value of i, above, is well defined. To do so, distinguish two cases in which the adversary might succeed in finding a collision. For the first case, $|X| \neq |X'|$. In this case the definition of pad (together with the requirement that $|X|, |X'| < 2^b$) ensures that $X_m \neq X'_m$ and so we will have $i = m$ as the index for a collision. In the second case, $|X| = |X'|$ and so, in particular, $m = m'$ and $\Delta = 0$. Because $X \neq X'$ there is a largest value $j \in [1 \mathinner{..} m]$ such that $X_j \neq X'_j$. It must be the case that $Y_j = Y'_j$ because the messages X and X', being identical on later blocks, would otherwise yield $Y_m = Y'_m$. But $X_j \neq X'_j$ and $Y_j = Y'_j$ and so $j = i$ satisfies the definition above.

We have shown that whenever A outputs a collision of H^*, adversary $C_{A,H}$ outputs a collision of H. The running time of $C_{A,H}$ is as claimed (the $+4$ accounts for 0-padding and length annotation in the scheme), so we are done. ▮

8 Asymptotic Treatment of Unkeyed Hash Functions

Definition. The traditional treatment of cryptographic hash-functions [7] is asymptotic. In this section we show that as long as one is willing to ask for security only against *uniform* adversaries, we don't need the keys in the asymptotic formalization of collision-resistant hash-functions either.

An *asymptotic-and-unkeyed hash-function* is a deterministic, polynomial-time algorithm H that takes as input an integer n, the *output length*, encoded in unary, and a string X, the *message*. It outputs either a string of length n or the distinguished value $\perp$. When we say that H is polynomial-time we mean that it is polynomial-time in its first input. We write H_n for the induced function $H(1^n, \cdot)$. Define the *message space* of H_n as $\mathcal{X}_n = \{X \in \{0,1\}^* :\ H_n(X) \neq \perp\}$ and that of H as the indexed family of sets $\langle X_n :\ n \in \mathbb{N}\rangle$. We assume $X \in \mathcal{X}_n$ implies every string of length $|X|$ is in $\mathcal{X}_n$, and we assume that $\mathcal{X}_n$ contains a string of length exceeding n.

Let C be an adversary for attacking asymptotic-and-unkeyed hash-function H, meaning that C is an algorithm (*not* a family of circuits; we are in the uniform setting) that, on input 1^n, outputs a pair of strings $X, X' \in \mathcal{X}_n$. We let the *advantage* of C in finding collisions in H be the function (of n) defined by

$$\mathbf{Adv}_H^{\mathrm{col}}(C, n) \quad = \quad \Pr[(X, X') \stackrel{\$}{\leftarrow} C(1^n) :\ X \neq X' \text{ and } H_n(X) = H_n(X')]$$

measuring, for each n, the probability that $C(1^n)$ finds a collision in H_n. We say that H is *collision-resistant* if for every polynomial-time adversary C, the function $\mathbf{Adv}_H^{\mathrm{col}}(C, n)$ is negligible. As usual, function $\epsilon(n)$ is *negligible* if for all $c > 0$ there exists an N such that $\epsilon(n) < n^{-c}$ for all $n \geq N$.

An asymptotic treatment of hash-then-sign. With a definition in hand it is easy to give an asymptotic counterpart for hash-then-sign, say. The existential (C0-style) statement would say that if Π is a secure signature scheme with message space $\langle \{0,1\}^n :\ n \in \mathbb{N}\rangle$ and H is a collision-resistant asymptotic-and-unkeyed hash-function with message space $\langle X_n \rangle$ then Π^H, the hash-then-sign construction using H and Π, is a secure signature scheme with message space $\langle X_n \rangle$. Details follow, beginning with the requisite definitions.

Now in the asymptotic setting [15], a *signature scheme* is a three-tuple of algorithms $\Pi = (\mathit{Gen}, \mathit{Sign}, \mathit{Verify})$. Algorithm Gen is a probabilistic polynomial-time (PPT) algorithm that, on input 1^n, outputs a pair of strings $(\mathit{PK}, \mathit{SK})$. Algorithm Sign is a PPT algorithm that, on input (SK, X), outputs either a string $\sigma \stackrel{\$}{\leftarrow} \mathit{Sign}(\mathit{SK}, X)$ or the distinguished value $\perp$. For each $n \in \mathbb{N}$ we require the existence of a *message spaces* $\mathcal{X}_n \subseteq \{0,1\}^*$ such that, for any SK that may be output by $Gen(1^n)$, we have that $\sigma \xleftarrow{\$} Sign(SK, X)$ is a string exactly when $X \in \mathcal{X}_n$. We insist that $\mathcal{X}_n$ contains all strings of a given length if it contains any string of that length. Algorithm *Verify* is a deterministic polynomial-time algorithm that, on input (PK, X, σ), outputs a bit. We require that if $(PK, SK) \xleftarrow{\$} Gen(1^n)$ and $X \in \mathcal{X}_n$ and $\sigma \xleftarrow{\$} Sign(SK, X)$ then $Verify(PK, X, \sigma) = 1$. We sometimes write $Sign_{SK}(X)$ and $Verify_{PK}(X, \sigma)$ instead of $Sign(SK, X)$ and $Verify(PK, X, \sigma)$. The message space of Π is the collection $\langle \mathcal{X}_n : n \in \mathbb{N} \rangle$. Throughout, an algorithm is *polynomial time* if it is polynomial time in the length of its first input. Now let B be an adversary for a signature scheme $\Pi = (Gen, Sign, Verify)$ as above. Then define $\mathbf{Adv}^{\mathrm{sig}}_{\Pi}(B, n)) = \Pr[(PK, SK) \xleftarrow{\$} Gen(1^n) : B^{Sign_{SK}(\cdot)}(PK)$ forges] where B is said to *forge* if it outputs a pair (X, σ) such that $Verify_{PK}(X, \sigma) = 1$ and B never asked a query X during its attack. We say that Π is *secure* (in the sense of existential unforgeability under an adaptive chosen-message attack) if for any polynomial-time adversary B the function $\mathbf{Adv}^{\mathrm{sig}}_{\Pi}(B, n)$ is negligible.

Let $\Pi = (Gen, Sign, Verify)$ be a signature scheme (for the asymptotic setting) with message space $\langle \mathcal{M}_n \rangle$ where $\mathcal{M}_n \supseteq \{0,1\}^n$. In this case we say that the message space of Π is "at least" $\langle \{0,1\}^n \rangle$. Let H be an asymptotic-and-unkeyed hash-function with message space $\langle \mathcal{X}_n \rangle$. Then define the hash-then-sign construction $\Pi^H = (Gen, Sign^H, Verify^H)$ by setting $Sign^H_{SK}(X) = Sign_{SK}(H(X))$ and $Verify^H_{PK}(X, \sigma) = Verify_{PK}(H(X), \sigma)$. The message space for Π^H is the message space for H. The security of the construction is captured by the following theorem. We omit a proof because it only involves writing down the asymptotic counterpart to the proof of Theorem 2.

Theorem 5 [hash-then-sign, unkeyed, asymptotic, C0-form] *If* Π *is a secure signature scheme with message space at least* $\langle \{0,1\}^n \rangle$ *and* H *is a collision-resistant asymptotic-and-unkeyed hash-function having message space* $\langle \mathcal{X}_n \rangle$ *then* Π^H *is a secure signature scheme with message space* $\langle \mathcal{X}_n \rangle$. $\diamondsuit$

Comparing Theorem 5 with Theorem 1 or 2, note that in stepping back to the asymptotic setting we also reverted to the existential style of theorem statement. But these choices are independent; one can give explicitly constructive (C1- or C2-style) theorem statements for the asymptotic setting.

Existence and constructions. We do not investigate the complexity assumption necessary to construct a collision-resistant asymptotic-and-unkeyed hash-function, but we do regard this as an interesting question. Natural constructions and cryptographic assumptions would seem to present themselves by adapting prior work like that in [7, 28].

9 Discussion

Using unkeyed hash-functions is no more complex than using keyed ones. For ease of comparison, we recall Damgård's definition of a collection of collision-free hash-functions [7] in Appendix A, and we provide a keyed treatment of hash-then-sign, in the concrete-security setting, in Appendix B.

Some readers may instinctively feel that there is something fishy about the approach advocated in this paper. One possible source of uneasiness is that, under our concrete-security treatment, no actual definition was offered for when an unkeyed hash-function is collision-resistant. But concrete-security treatments of cryptographic goals *never* define an absolute notion for when a cryptographic object is secure. Similarly, it might seem fishy that, in the asymptotic setting, we restricted attention to uniform adversaries. We proffer that collision-resistance of an unkeyed output-length-parameterized hash-function makes intuitive sense, but only in the uniform setting. Regardless, we suspect that the greater part of any sense of unease stems from our community having internalized the belief that an unkeyed treatment of collision-resistance just cannot work.

In Damgård's words, *Instead of considering just one hash function, we will consider families of them, in order to make a complexity theoretic treatment possible* [7]. This refrain has been repeated often enough to have become undisputed fact. But Damgård was thinking in terms of asymptotic complexity and nonuniform adversaries; when one moves away from this, and makes a modest shift in viewpoint about what our theorem statements should say, what was formerly impossible becomes not just possible, but easy.

Going further, one could make the argument that it is historical tradition that has made our hash functions keyed more than the specious argument from Section 1 about the infeasibility of formalizing what human beings do not know. When Damgård defined collision-resistance [7] we already had well-entrenched traditions favoring asymptotic notions, non-uniform security, number-theoretic constructions, assumptions like claw-free pairs, and existential-format (C0-style) theorem statements. These traditions point away from the human-ignorance approach. Besides, it was never Damgård's goal to demonstrate how to do provable-security cryptography with an unkeyed hash-function $H\colon \{0,1\}^* \to \{0,1\}^n$. While such hash functions were known (eg, [21, 24, 31]), they probably were not looked upon as suitable starting points for doing rigorous cryptographic work.

Protocols that use cryptographic hash-functions are often proved secure in the random-oracle (RO) model [4]. In such a case, when one replaces the RO-modeled hash-function H by some concrete function one would like to preserve the function's domain and range, $H\colon \mathcal{X} \to \mathcal{Y}$ for $\mathcal{X}, \mathcal{Y} \subseteq \{0,1\}^*$. So replacing a RO by a concrete hash-function *always* takes you away from the keyed-hash-function setting. Concretely, one can prove security for hash-then-sign in the RO model but one cannot instantiate the RO with a keyed hash-function without changing the protocol first.

One could argue that mandating an explicitly-specified reduction is a sensible way to state provable-security results in general. After all, if a reduction actually were nonconstructive it would provide a less useful guarantee. That's because a constructive reduction says something meaningful about cryptographic practice *now*, independent of mathematical truth. A constructive statement along the lines of "if you know how to break this signature scheme then you know how to factor huge numbers" tells us that, right now, he who can do the one task can already do the other. If it takes 100 years until anyone can factor huge numbers then signature schemes that enjoy the constructive provable-security guarantee are guaranteed to protect against forgeries for all those intervening years.

The human-ignorance approach can be used for cryptographic goals beyond collision resistance. For example, one might assume of a blockcipher $E\colon \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ that nobody can find a point (K, X) such that $E_K(X) = X$. Unlike the ideal-cipher model, the assumption is meaningful for a concretely instantiated blockcipher.

The topic of this paper is largely about *language*: how, exactly, to express provable-security results. Some may interpret this to mean that the topic is insignificant, being *only* an issue of language. But language is key. In a case like this, language shapes our basic ideas, their development, and their utility.

In recent years, MD4-family hash functions (MD4, MD5, SHA-0, SHA-1, RIPEMD) have suffered an onslaught of successful attacks. This paper provides no guidance in how to recognize or build unkeyed hash-functions for which mankind will *not* find collisions. It only illustrates how, when you *do* have such a hash function in hand, you can formulate the security of a higher-level protocol that uses it, obtaining the usual benefits of provable-security cryptography.

Acknowledgments

Some ideas in this paper go back to long-ago discussions with Mihir Bellare. Well over a decade ago we talked about the utility of making theorem statements explicitly constructive, and, occasionally, we did so. Stefan Lucks asked a question following a presentation at FSE '04 [27] that helped engender this paper. Jesse Walker, and others later on, asked me about the foundations-of-hashing dilemma (of course not in that language), further motivating this writeup. I received excellent comments from Mihir Bellare, Dan Brown, Tom Shrimpton, and Doug Stinson. Andy Okun pointed me to [6], which considers human ignorance from a rather different perspective. This work was supported by NSF grant CCR-0208842 and a generous gift from Intel Corporation.

References

[1] M. Bellare, R. Guérin, and P. Rogaway. XOR MACs: New methods for message authentication using finite pseudorandom functions. Full version of CRYPTO '95 paper. Available on-line from the author's web page.

[2] M. Bellare, J. Kilian, and P. Rogaway. The security of the cipher block chaining message authentication code. *J. of Computer and System Sciences* (JCSS), vol. 61, no. 3, pp. 362–399, 2000.

[3] M. Bellare and P. Rogaway. Collision-resistant hashing: towards making UOWHFs practical. *Advances in Cryptology – CRYPTO '97*, LNCS, Springer, 1997.

[4] M. Bellare and P. Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. *First ACM Conference on Computer and Communications Security* (CCS '93), ACM Press, pp. 62–73, 1993.

[5] D. Brown. Generic groups, collision resistance, and ECDSA. *Designs, Codes and Cryptography*, vol. 35, no. 1, pp. 119–152, 2005.

[6] C. Cipolla. *Le leggi fondamentali della stupidità* (The fundamental laws of human stupidity). In *Allegro ma non troppo con Le leggi fondamentali della stupidità*, Società editrice il Malino, Bologna, 1988.

[7] I. Damgård. Collision free hash functions and public key signature schemes. *Advance in Cryptology – EUROCRYPT '87*, LNCS vol. 304, Springer, pp. 203–216, 1987.

[8] I. Damgård. A design principle for hash functions. *Advances in Cryptology – CRYPTO '89*, LNCS vol. 435, Springer, 1990.

[9] A. De Santis and M. Yung. On the design of provably secure cryptographic hash functions. *Advance in Cryptology – EUROCRYPT '90*, LNCS vol. 473, Springer, pp. 412–431, 1991.

[10] P. Devanbu, M. Gertz, A. Kwong, C. Martel, G. Nuckolls, and S. Stubblebine. Flexible authentication of XML documents. *J. of Computer Security*, vol. 12, no. 6, pp. 841–864, 2004.

[11] O. Goldreich and H. Krawczyk. On the composition of zero-knowledge proof systems. *SIAM Journal on Computing*, vol. 25, no. 1, pp. 169–192, Feb 1997.

[12] O. Goldreich and Y. Oren. *Definitions and properties of zero-knowledge proof systems. J. of Cryptology*, vol. 7, no. 1, pp. 1–32, 1994.

[13] S. Goldwasser and S. Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, vol. 28, no. 2, pp. 270–299, 1984. Earlier version in *STOC 82*.

[14] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, vol. 18, no. 1, pp. 186–208, 1989.

[15] S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. on Comp.*, vol. 17, pp. 281–308, 1988.

[16] S. Halevi and H. Krawczyk. Strengthening digital signatures by randomized hashing. Manuscript, 6 June 2006. Proceedings version in *Advances in Cryptology – CRYPTO '06*, LNCS, Springer, 2006.
[17] G. Laccetti and G. Schmid. On a probabilistic approach to the security analysis of cryptographic hash functions. Cryptology ePrint report 2004/324, September 2004.
[18] R. Merkle. Method of providing digital signatures. US Patent #4,309,569, 1982.
[19] R. Merkle. One way hash functions and DES. *Advances in Cryptology – CRYPTO '89*, LNCS vol. 435, Springer, pp. 428–446, 1990.
[20] R. Merkle. Protocols for public key cryptosystems. *Proceedings of the 1980 IEEE Symposium on Security and Privacy*, IEEE Press, pp. 122–134, 1980.
[21] S. Matyas, C. Meyer, and J. Oseas. Generating strong one-way functions with cryptographic algorithm. *IBM Tech. Disclosure Bulletin*, 27, pp. 5658–5659, 1985.
[22] National Institute of Standards and Technology. FIPS PUB 180-2, Secure Hash Standard, Aug 1, 2002.
[23] Y. Oren. On the cunning power of cheating verifiers: some observations about zero-knowledge proofs. *28th Annual Symposium on the Foundations of Computer Science* (FOCS 87), IEEE Press, pp. 462–471, 1987.
[24] M. Rabin. Digital signatures. In *Foundations of secure computation*, R. DeMillo, D. Dobkin, A. Jones, and R. Lipton, editors, Academic Press, pp. 155–168, 1978.
[25] O. Reingold, L. Trevisan, and S. Vadhan. Notions of reducibility between cryptographic primitives. *Theory of Cryptography Conference, TCC 2004*, LNCS vol. 2951, Springer, pp. 1–20, 2004.
[26] R. Rivest. The MD4 message digest algorithm. *Advance in Cryptology – CRYPTO '90*, LNCS vol. 537, Springer, pp. 303–311, 1991.
[27] P. Rogaway and T. Shrimpton. Cryptographic hash-function basics: definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. *Fast Software Encryption* (FSE 2004), LNCS vol. 3017, Springer, pp. 371–388, 2004.
[28] A. Russell. Necessary and sufficient conditions for collision-free hashing. *J. of Cryptology*, vol. 8, no. 2, pp. 87–99, 1995.
[29] D. Stinson. Some observations on the theory of cryptographic hash functions. Cryptology ePrint report 2001/020, March 2001. Preliminary version of [30].
[30] D. Stinson. Some observations on the theory of cryptographic hash functions. *Designs, Codes and Cryptography*, vol. 38, pp. 259–277, 2006. Journal version of [29].
[31] R. Winternitz. A secure one-way hash function built from DES. *Proceedings of the IEEE Symposium on Inf. Security and Privacy*, pp. 88–90, IEEE Press, 1984.

A The Traditional Definition of Collision Resistance

In this section we recall, for comparison, the traditional definition for a collision-resistant hash function, as given by Damgård [7]. The notion is keyed (meaning that the hash functions have an *index*) and asymptotic. Our wording and low-level choices are basically from [28].

A *collection of collision-free hash-functions* is a set of maps $\{h_K\colon\ K \in \mathcal{I}\}$ where $\mathcal{I} \subseteq \{0,1\}^*$ and $h_K\colon\ \{0,1\}^{|K|+1} \to \{0,1\}^{|K|}$ and where:

1. There is an EPT algorithm $\mathcal{K}$ that, on input 1^n, outputs an n-bit string $K \xleftarrow{\$} \mathcal{K}(1^n)$ in $\mathcal{I}$.

2. There is an EPT algorithm H that, on input $K \in \mathcal{I}$ and $X \in \{0,1\}^{|K|+1}$, computes $H(K,X) = h_K(X)$.

3. For any EPT adversary A, $\epsilon(n) = \Pr[K \stackrel{\$}{\leftarrow} \mathcal{K}(1^n);\ (X, X') \stackrel{\$}{\leftarrow} A(K):\ X \neq X' \text{ and } H_K(X) = H_K(X')]$ is negligible.

Above, EPT stands for *expected polynomial time*, and a function $\epsilon(n)$ is *negligible* if for every $c > 0$ there exists an N such that $\epsilon(n) < n^{-c}$ for all $n \geq N$. For simplicity, we assumed that the domain of each h_K is $\{0,1\}^{|K|+1}$. This can be relaxed in various ways.

B Hash-then-Sign with a Keyed Hash-Function

In this section we provide a concrete-security treatment of the hash-then-sign paradigm using a keyed hash-function instead of an unkeyed one. Our purpose is to facilitate easy comparison between the keyed and unkeyed form of a theorem.

First we must modify our formalization of the hash-then-sign construction to account for the differing syntax of a keyed and unkeyed hash function. Let $H\colon \mathcal{K}\times\{0,1\}^* \to \{0,1\}^n$ be a keyed hash-function. Let $\Pi = (\mathit{Gen}, \mathit{Sign}, \mathit{Verify})$ be a signature scheme with message space of at least $\{0,1\}^n$. Define from these the signature scheme $\Pi^H = (\mathit{Gen}^H, \mathit{Sign}^H, \mathit{Verify}^H)$ by saying that Gen^H samples $K \stackrel{\$}{\leftarrow} \mathcal{K}$ and $(PK, SK) \stackrel{\$}{\leftarrow} \mathit{Gen}$ and then outputs $(\langle PK, K\rangle,\ \langle SK, K\rangle)$; define $\mathit{Sign}^H_{\langle SK,K\rangle}(M) = \mathit{Sign}_{SK}(H_K(M))$; and define $\mathit{Verify}^H_{\langle PK,K\rangle}(M,\sigma) = \mathit{Verify}_{PK}(H_K(M),\sigma)$. The message space for Π^H is $\{0,1\}^*$. We have reused the notation Π^H and Sign^H and Verify^H because the "type" of the hash function H makes unambiguous what construction is intended.

The proof of the following, little changed from Theorem 2, is omitted.

Theorem 6 [hash-then-sign, keyed, concrete, C0] *Let $H\colon \mathcal{K} \times \{0,1\}^* \to \{0,1\}^n$ be a keyed hash-function, let $\Pi = (\mathit{Gen}, \mathit{Sign}, \mathit{Verify})$ be a signature scheme with message space at least $\{0,1\}^n$, and let A be an adversary. Then there exist adversaries B and C such that*

$$\mathbf{Adv}^{\mathrm{sig}}_{\Pi}(B) + \mathbf{Adv}^{\mathrm{col}}_{H}(C) \quad\geq\quad \mathbf{Adv}^{\mathrm{sig}}_{\Pi^H}(A)\,.$$

Adversary B runs in time at most $t_A + t_{\mathcal{K}} + t_H(\ell_A) + t_{\mathit{Sign}}(nq_A) + c(\ell_A + nq_A)$ and asks at most q_A queries entailing at most $\ell_A + n$ bits. Adversary C runs in time at most $t_A + t_{\mathit{Gen}} + t_{\mathcal{K}} + t_H(\ell_A) + t_{\mathit{Sign}}(nq_A{+}n) + c(\ell_A + nq_A)\lg(q_A)$. The value c is an absolute constant implicit in the proof of this theorem. $\diamondsuit$