Formalizing Human Ignorance
Collision-Resistant Hashing Without the Keys

Phillip Rogaway\textsuperscript{1,2}

\textsuperscript{1} Dept. of Computer Science, University of California, Davis, California 95616, USA
\textsuperscript{2} Dept. of Computer Science, Chiang Mai University, Chiang Mai 50200, Thailand

Abstract. There is a rarely mentioned 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 $H: \{0,1\}^* \rightarrow \{0,1\}^n$ 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.

1 Introduction

Foundations-of-hashing dilemma. In cryptographic practice, a collision-resistant hash-function (an object like SHA-1) maps arbitrary-length strings to fixed-length ones; it’s an algorithm $H: \{0,1\}^* \rightarrow \{0,1\}^n$ for some fixed $n$. But in cryptographic theory, a collision-resistant hash-function is always keyed; now $H: \mathcal{K} \times \{0,1\}^* \rightarrow \{0,1\}^n$ where each $K \in \mathcal{K}$ names a function $H_K(\cdot) = H(K, \cdot)$. In this case $H$ can be thought of as a collection or family of hash functions $H = \{H_K: \ K \in \mathcal{K}\}$, each key (or index) $K \in \mathcal{K}$, naming one.

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 $H: \{0,1\}^* \rightarrow \{0,1\}^n$ 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 $H: \{0,1\}^* \rightarrow \{0,1\}^n$ is collision resistant cannot be that there is no efficient

\textsuperscript{1} (a) We call $K$ a key, but it is not secret; one chooses $K$ from $\mathcal{K}$ and then makes it public. (b) Writing $H: \mathcal{K} \times \{0,1\}^* \rightarrow \{0,1\}^n$ assumes a concrete-security formalization; early formalizations were instead asymptotic. We’ll discuss both. (c) Alternative terms for collision-resistant are collision-free and collision-intractable.
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 can’t hope to construct a meaningful theory based on what Xiaoyun Wang [28,29] does or doesn’t know. Regarding a hash function like SHA-1 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 $\Pi$ 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 $\Pi$ implies the existence of an effective adversary $C$ against $H$. But this won’t work when we have an unkeyed $H : \{0,1\}^* \rightarrow \{0,1\}^n$ 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 $\Pi$ 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 $\Pi$ then they know how to find collisions in $H$; and if nobody can find collisions in $H$ then nobody can break $\Pi$. 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 approach (or, less colorfully, the 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