On the Impossibility of Highly-Efficient Blockcipher-Based Hash Functions

J. Black * M. Cochran * T. Shrimpton †

Abstract

Fix a small, non-empty set of blockcipher keys $\mathcal{K}$. We say a blockcipher-based hash function is *highly-efficient* if it makes exactly one blockcipher call for each message block hashed, and all blockcipher calls use a key from $\mathcal{K}$. Although a few highly-efficient constructions have been proposed, no one has been able to prove their security. In this paper we prove, in the ideal-cipher model, that it is *impossible* to construct a highly-efficient iterated blockcipher-based hash function that is provably secure. Our result implies, in particular, that the Tweakable Chain Hash (TCH) construction suggested by Liskov, Rivest, and Wagner [6] is *not* correct under an instantiation suggested for this construction, nor can TCH be correctly instantiated by any other efficient means.

Keywords: Collision-resistant hash functions, tweakable blockciphers, provable security.

* Department of Computer Science, 430 UCB, Boulder, Colorado 80309 USA. E-mail: jrblack@cs.colorado.edu, Martin.Cochran@colorado.edu WWW: www.cs.colorado.edu/∼jrblack/, ucsu.colorado.edu/∼cochranm

† Department of Computer Science, Portland State University, Portland, Oregon, 97207, USA. E-mail: teshrim@cs.pdx.edu WWW: www.cs.pdx.edu/∼teshrim/

Contents

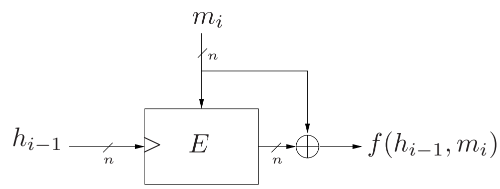


Figure 1: The Matyas-Meyer-Oseas (MMO) compression function [7]. $E\colon \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ is a block cipher; the hatch mark denotes the location of the key. Iterating this compression function results in a provably-secure hash function [1], however notice that the above compression function will be rekeyed each round.

1 Introduction

Background. Essentially all modern hash functions are built by iterating a compression function according to the Merkle-Damgård paradigm [3, 9]. Moreover, these compression functions are almost always built from a blockcipher. Constructions like the Matyas-Meyer-Oseas (MMO) compression function [7] are explicit about their use of a blockcipher, but even so-called "dedicated" hashing primitives like MD5 and SHA-1 are in fact blockcipher-based. The SHA-1 compression function, for example, uses a 160-bit blockcipher that takes a 512-bit key; this blockcipher has been named SHACAL-1 [5].

This idea of building hash functions from blockciphers goes back more than 25 years. The earliest construction by Rabin [11] proposed to hash a message $M = m_1 m_2 \cdots m_\ell$ by fixing an initial value h_0 and computing $H(M) = \mathrm{DES}_{m_\ell}(\mathrm{DES}_{m_{\ell-1}}(\cdots(\mathrm{DES}_{m_1}(h_0))))$; effectively, this is a Merkle-Damgård construction with blockcipher-based compression function $f(h_{i-1}, m_i) = E_{m_i}(h_{i-1})$. Other constructions like Davies-Meyers [8] and MMO followed, but it was Preneel, Govaerts, and Vandewalle [10] who conducted the first systematic study of blockcipher-based hash functions. They considered the security of 64 iterated constructions with compression functions of the form $f(h_{i-1}, m_i) = E_a(b) \oplus c$ where $a, b, c \in \{h_{i-1},\ m_i,\ h_{i-1} \oplus m_i,\ v\}$ for some fixed constant v. The analysis of PGV was attack-based, and schemes not broken by their attacks were deemed secure. Subsequently, Black, Rogaway and Shrimpton [1] considered these same 64 constructions using a proof-based approach. They showed that, in the ideal-cipher model, 20 of the 64 schemes are collision-resistant up to the birthday bound; Figure 1 gives one example.

Although provably secure, these 20 schemes could be viewed as inefficient in the following sense: in each, the blockcipher key is changed every round. For all conventional blockciphers, changing the key each round is undesirable since scheduling a new key entails a significant computational cost. It is natural to ask then if it is possible to achieve provable security without incurring this cost, and this question is the focus of our work.

Main Result. Fix a small, non-empty set of blockcipher keys $\mathcal{K}$. We term a blockcipher-based hash function *highly-efficient* if its compression function uses exactly one call to a blockcipher (ie, it is rate-1), and if the blockcipher uses only keys from $\mathcal{K}$. Since we can preschedule each key in $\mathcal{K}$, we enjoy a significant performance gain: key scheduling reduces to looking up a precomputed permutation. It is possible that those researchers who have worked on blockcipher-based hash functions over the past 25 years have considered highly-efficient constructions, but found attacks that broke these constructions, or were unable to prove their security. Indeed, the present authors also spent some time trying to find highly-efficient constructions without success. We now explain why.

One would like to construct a highly-efficient hash function that is provably collision resistant. If such a construction did exist, its underlying compression function could be constructed as follows (see Figure 2): let $f_1\colon \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ and $f_2\colon \{0,1\}^n \times \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ be arbitrary functions. We define $f\colon \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ as

$$f(h_{i-1}, m_i) = f_2(h_{i-1}, m_i, E_K(f_1(h_{i-1}, m_i))),$$

where E_K is an n-bit blockcipher with key $K \in \mathcal{K}$ that is the output of a deterministic key-selection function g.

It isn't hard see that this construction captures all possible rate-1 compression functions built from a blockcipher used in the forward direction and keyed from $\mathcal{K}$: both f_1 and f_2 may process every bit of the input to f in any arbitrary way. Notice that it isn't necessary to feed forward the output of f_1 to f_2, since f_2 can

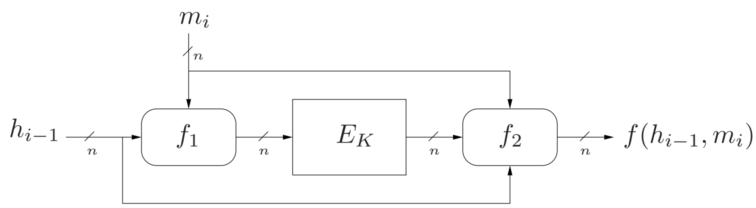


Figure 2: The general compression function built from a blockcipher keyed from $\mathcal{K}$. Functions $f_1\colon \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ and $f_2\colon \{0,1\}^n \times \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ are arbitrary; E_K is some n-bit blockcipher with key $K \in \mathcal{K}$ selected by a deterministic key-selection function g.

compute $f_1(h_{i-1}, m_i)$ itself. The key-selection function g must be deterministic and well-defined, but beyond this may depend on other message blocks, chaining values, the round number, or other parameters, provided it always returns a value from $\mathcal{K}$. These notions are made precise in Section 2. (Note that this approach does not capture *all* blockcipher-based hash functions with a fixed key-set, only those that are iterated via Merkle-Damgård and that use the blockcipher in the forward direction.)

In this paper we prove that any compression function constructed as just described *cannot* produce a provably collision-resistant hash function when iterated. Specifically, we show—in the ideal-cipher model—that for any functions f_1, f_2, there exists an information-theoretic adversary that finds a collision in the iterated function H^f in at most $|\mathcal{K}|(n + \lceil \lg(n) \rceil + 2\lceil \lg(|\mathcal{K}|)\rceil + 1) + 1$ blockcipher invocations. In fact, for many natural functions f_1, f_2 (like XOR) we find collisions in just $2|\mathcal{K}|$ blockcipher invocations. This is in stark contrast to the $\Theta(2^{n/2})$ expected invocations needed to produce a collision in the 20 rekeying constructions proven secure in [1]. Our impossibility proof uses a greedy algorithm that builds large numbers of messages along with their associated hash outputs. In the case $|\mathcal{K}| = 1$, we prove that this algorithm builds a tree with height at most $n + \lceil \lg(n) \rceil$ containing at least $2^n(n + \lceil \lg(n) \rceil) + 1$ hash values, thereby yielding a collision on some level of the tree. A similar pigeonhole argument holds for $|\mathcal{K}| > 1$. We stress that our results should *not* be interpreted as giving practical attacks on all highly-efficient hash functions: our attacks have exponential running time. Instead we are exhibiting a proof that no highly-efficient iterated blockcipher-based hash functions can exist, in the model we have described.

Security of Tweak Chain Hash. Tweakable blockciphers were introduced by Liskov, Rivest, and Wagner [6]. They define a tweakable blockcipher as a map $\widetilde{E}\colon \{0,1\}^k \times \{0,1\}^t \times \{0,1\}^n \to \{0,1\}^n$ where the inputs are called the *key*, the *tweak* and the *message*. We sometimes write $\widetilde{E}_K(T, M)$ instead of $\widetilde{E}(K, T, M)$. For any fixed $K \in \{0,1\}^k$ and $T \in \{0,1\}^t$, we require that $\widetilde{E}(K, T, \cdot)$ is a permutation on n bits. The idea is for the tweakable blockcipher to act like a normal blockcipher but with an extra (public) input, the tweak, which adds variability. The key may be expensive to schedule and to change, but changes to the tweak should be inexpensive. Security is defined as indistinguishability of a family of random permutations from $\widetilde{E}_K(\cdot, \cdot)$ with random key K, where the adversary controls the tweak and the message. See Section 2 for a formal definition.

Along with several other constructions, Liskov, Rivest, and Wagner suggest a new iterated hash-function construction built on tweakable blockciphers called the "Tweak Chain Hash" (TCH). This is a straightforward adaptation of the MMO construction into the tweakable setting: let $\widetilde{E}$ be a tweakable blockcipher with $t = n$, and fix a key $K \in \{0,1\}^k$. For any $M \in (\{0,1\}^n)^+$ write $M = m_1 \cdots m_\ell$ where each $|m_i| = n$, define $TCH^{\widetilde{E}_K}(M)$ as

function $TCH^{\widetilde{E}_K}(m_1 \cdots m_\ell)$
  for $i \leftarrow 1$ **to** ℓ **do** $h_i \leftarrow \widetilde{E}_K(h_{i-1}, m_i) \oplus m_i$
  return h_ℓ

where h_0 is a fixed constant, say 0^n. See Figure 3. A main motivation for TCH is efficiency: in each round the (expensive to change) key K remains fixed while the (cheap to change) tweak and message vary. One might therefore expect TCH to be substantially faster than MMO.

However the security of TCH is left as an open question. In the same paper, the authors propose two ways to create tweakable blockciphers from conventional blockciphers: one construction based on the CBC-MAC

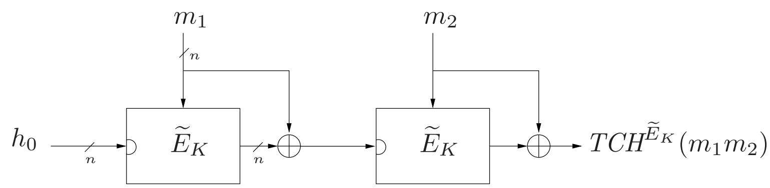


Figure 3: Two rounds of the Tweak Chain Hash compression function. $\widetilde{E}_K$ is the tweakable blockcipher, K is a fixed public key; the arc denotes the location of the tweak.

and one using universal hash families. So it is natural to wonder whether TCH is secure when built using either of these constructions. But inserting the second construction into TCH yields a fixed-key rate-1 hash function constructed from a *conventional* blockcipher and given our results above, we would expect that any such construction should be insecure. In fact we show something even stronger.

We demonstrate that using *either* tweakable-blockcipher construction from Liskov et al., the resulting TCH construction admits a simple attack. These attacks produce an infinite number of same-length colliding message pairs under TCH, regardless of the parameters chosen for the underlying tweakable blockcipher and regardless of the security model. Appealing to our main result, we further show in the ideal-cipher model that *any* tweakable blockcipher—built using one call to a conventional blockcipher—will yield an insecure TCH construction. Our result does not, however, rule out TCH being secure when constructed from a tweakable blockcipher *primitive*, such as the Hasty Pudding cipher [12]. This is discussed further in Section 4.

Security Model. The standard-model assumption for blockciphers is that they are good pseudo-random permutations (PRPs) [8]. However this assumption is insufficient for proving the security of hash functions based on blockciphers; indeed, Simon has shown [14] that the PRP assumption alone is insufficient for secure blockcipher-based hashing. For this reason, all proofs of security for blockcipher-based hash functions have been done in the ideal-cipher model [1, 8, 9, 14, 15]. This model, which dates back to Shannon [13], treats a blockcipher as a random and independent permutation for each key. Some believe that modeling blockciphers in this way is not realistic: often we find correlations and biases in real blockciphers that one would not expect to see if the object were drawn uniformly from the family of all blockciphers with the same block and key size. Nonetheless, proofs of security in this model do have meaning: security is guaranteed against adversaries that ignore the structure of the underlying blockcipher.

Except for the two simple attacks given for TCH, all attacks in this paper are in the ideal-cipher model. We do not make any probabilistic assumptions nor do we depend on the permutivity of the blockciphers.

The normal measure of security for blockcipher-based hash functions is that they are secure against information-theoretic adversaries in the ideal-cipher model [1]. Our adversaries are therefore information-theoretic. While this may seem to be giving too much power to an adversary when thinking of real-world attacks, we once again stress that our goal is to demonstrate the nonexistence of *provably* collision-resistant schemes, not to give practically instantiable attacks. Indeed, it is clearly impossible to exhibit a practical attack given the generality of our setting: f_1 could itself be a collision-resistant hash function and a practical attack on the resulting hash function would imply a practical attack on f_1.

Message Lengths. Our definition for collision resistance will count as valid *any* pair of messages that produce the same hash value. Finding collisions in practice is often much harder than this due to techniques such as Merkle-Damgård strengthening [8]. In view of this, all attacks in this paper produce colliding messages of the same length, and therefore still apply even in the presence of such techniques.

2 Security Definitions

Basic Notions. Let k and n be positive integers. A *blockcipher* is a function $E\colon \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ where for each $K \in \{0,1\}^k$ we require that $E_K(\cdot) = E(K,\cdot)$ is a permutation on $\{0,1\}^n$. Let $\mathrm{Perm}(n)$ be the set of all permutations on $\{0,1\}^n$, and let $\mathrm{Bloc}(k,n)$ be the set of all blockciphers $E\colon \{0,1\}^k \times$

$\{0,1\}^n \to \{0,1\}^n$. A function $f\colon \mathrm{Perm}(n) \times \mathcal{D} \to \{0,1\}^c$ is a (permutation-based) *compression function* if $\mathcal{D} = \{0,1\}^a \times \{0,1\}^n$ for some $a \geq 1$ where $a + n \geq c$. If the program for computing f uses a single query $\pi(x)$ to compute $f^\pi(h,m) = f(\pi,h,m)$ then f is *rate-1*.

For non-empty sets $\mathcal{K} \subset \{0,1\}^k$ and $\mathcal{S} \subseteq \{0,1\}^*$, fix a deterministic *key-selection function* $g\colon \mathcal{S} \times \{0,1\}^n \times \{0,1\}^n \to \mathcal{S} \times \mathcal{K}$. Fix a constant h_0 and let $\sigma_0 = \varepsilon$. We define a (highly-efficient, blockcipher-based) *hash function* by the following program:

function $H[g,\mathcal{K}]^E(m_1 \cdots m_\ell)$
  for $i \leftarrow 1$ **to** ℓ **do**
    $(\sigma_i, K) \leftarrow g(\sigma_{i-1}, h_{i-1}, m_i)$
    $\pi \leftarrow E_K$
    $h_i \leftarrow f^\pi\,(h_{i-1}, m_i)$
  return h_ℓ

where $f\colon \mathrm{Perm}(n) \times \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ is a rate-1 compression function. This program computes a map $H[g,\mathcal{K}]^E\colon \mathrm{Bloc}(k,n) \times (\{0,1\}^n)^+ \to \{0,1\}^n$, and we say that $H[g,\mathcal{K}]^E$ is rate-1 because f is. Sometimes we will call $H[g,\mathcal{K}]^E$ the *iterated hash* of f, for obvious reasons.

When it is understood from context, we will omit the superscript π to f, and E to H. We will also often omit explicit reference to g and $\mathcal{K}$, simply writing H for $H^E[g,\mathcal{K}]$.

We write $x \xleftarrow{\$} S$ for the experiment of choosing a random element from the finite set S and calling it x. An *adversary* is an algorithm with access to one or more oracles, which we write as superscripts.

Collision resistance. To quantify the collision resistance of a highly-efficient, blockcipher-based hash function, we model the blockcipher as a randomly chosen $E \in \mathrm{Bloc}(k,n)$. An adversary A is given oracles for $E(\cdot,\cdot)$ and its inverse $E^{-1}(\cdot,\cdot)$, and wants to find a *collision* for H—that is, $M, M' \in \mathcal{D}$ where $M \neq M'$ but $H(M) = H(M')$. We look at the number of queries that the adversary makes and compare this with the probability of finding a collision.

Definition 1 [Collision resistance of a hash function] Fix $k,n > 0$, and let $E\colon \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ be a blockcipher. Let $\mathcal{K} \subset \{0,1\}^k$ and $\mathcal{S} \subseteq \{0,1\}^*$ be non-empty sets, and let $g\colon \mathcal{S} \times \{0,1\}^n \times \{0,1\}^n \to \mathcal{S} \times \mathcal{K}$ be a key-selection function. Let $H[g,\mathcal{K}]$ be a highly-efficient, blockcipher-based hash function, $H[g,\mathcal{K}]\colon \mathrm{Bloc}(k,n) \times \mathcal{D} \to \{0,1\}^n$, and let A be an adversary. Then the advantage of A in finding collisions in H is the real number

$$\begin{aligned}\mathbf{Adv}_H^{\mathrm{coll}}(A) &= \Pr\Big[E \xleftarrow{\$} \mathrm{Bloc}(k,n);\, (M,M') \xleftarrow{\$} A^{E(\cdot,\cdot),E^{-1}(\cdot,\cdot)}:\\ &\qquad\qquad M \neq M' \wedge H^E(M) = H^E(M')\Big]\end{aligned}$$

For $q \geq 1$ we write $\mathbf{Adv}_H^{\mathrm{coll}}(q) = \max_A\{\mathbf{Adv}_H^{\mathrm{coll}}(A)\}$ where the maximum is taken over all adversaries that ask at most q oracle queries.

Tweakable Blockciphers. Fix $k,t,n > 0$. A *tweakable blockcipher* is a function $\widetilde{E}\colon \{0,1\}^k \times \{0,1\}^t \times \{0,1\}^n \to \{0,1\}^n$ such that for any $K \in \{0,1\}^k$ and any $T \in \{0,1\}^t$ we are guaranteed that $\widetilde{E}(K,T,\cdot) = \widetilde{E}_K(T,\cdot)$ is a permutation on $\{0,1\}^n$. If we write $\widetilde{\pi} \xleftarrow{\$} \{0,1\}^t \times \mathrm{Perm}(n)$ we are choosing 2^t random permutations on $\{0,1\}^n$, one for each $T \in \{0,1\}^t$. The permutation associated to T is $\widetilde{\pi}(T,\cdot)$.

Definition 2 [Security of Conventional and Tweakable Blockciphers] Let $\widetilde{E}\colon \{0,1\}^k \times \{0,1\}^t \times \{0,1\}^n \to \{0,1\}^n$ be a tweakable blockcipher, and let A be an adversary. Then

$$\begin{aligned}\mathbf{Adv}_E^{\mathrm{prp}}(A) &= \Pr[K \xleftarrow{\$} \mathcal{K}:\ A^{E_K(\cdot)} = 1] - \Pr[\pi \xleftarrow{\$} \mathrm{Perm}(n):\ A^{\pi(\cdot)} = 1]\\ \mathbf{Adv}_{\widetilde{E}}^{\mathrm{tprp}}(A) &= \Pr[K \xleftarrow{\$} \{0,1\}^k:\ A^{\widetilde{E}_K(\cdot,\cdot)} = 1] -\\ &\qquad \Pr[\widetilde{\pi} \xleftarrow{\$} \{0,1\}^t \times \mathrm{Perm}(n):\ A^{\widetilde{\pi}(\cdot,\cdot)} = 1]\end{aligned}$$

Write $\mathbf{Adv}_E^{\mathrm{prp}}(q) = \max_A\{\mathbf{Adv}_E^{\mathrm{prp}}(A)\}$ and $\mathbf{Adv}_{\widetilde{E}}^{\mathrm{tprp}}(q) = \max_A\{\mathbf{Adv}_{\widetilde{E}}^{\mathrm{tprp}}(A)\}$ for $q \geq 1$ and where the maxima are taken over all adversaries that ask at most q oracle queries.

3 Hash Function Constructions and Attacks

We begin this section with a more detailed discussion of the generalized rate-1 blockcipher-based compression function shown in Figure 2, and of certain assumptions we might make in practice (though our later proofs will make no such assumptions). Next we consider attacks on the iterated hash of this compression function. The first attack is particularly efficient (it requires only $2|\mathcal{K}|$ blockcipher invocations) and we argue that it probably applies to many "reasonable" constructions for the compression function. The second is more general: it shows there cannot exist an iterated hash function based on this type of compression function that is provably collision resistant in our model.

Generalized Rate-1 Blockcipher-based Compression Function. We consider any compression function f that is built in the following way. Let $f_1\colon \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ and $f_2\colon \{0,1\}^n \times \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ be arbitrary functions. We define $f\colon \mathrm{Perm}(n) \times (\{0,1\}^n \times \{0,1\}^n) \to \{0,1\}^n$ as $f^\pi(h,m) = f_2(h,m,\pi(f_1(h,m)))$. See Figure 2 with $\pi = E_K$. We will not formally argue that this construction covers all possible $2n$ to n bit functions that call π at most once; this would take us quite far afield. Instead we give the following informal justification.

The function f takes two n-bit inputs, h and m. We make both of these inputs available to the "preprocessing" function f_1 and to the "postprocessing" function f_2. Additionally, f_2 has access to the output of π. We do not feed the output of f_1 to f_2 since f_2 is capable of recomputing f_1 itself. Similarly, we do not feed the output of f_2 back to f_1 since any computation performed by f_2 only on inputs h and m could have been computed by f_1; if the output of f_2 depends also on π then it cannot be fed back into f_1 (and thus π) because we are requiring f be rate-1.

Although f_1 and f_2 are fully arbitrary, we imagine that in practice they will be simple and fast-to-compute functions. In PGV [10], for example, these functions are never more complex than XOR. It would make little sense to have f_1 be, say, SHA-1 since our overall construction is itself aiming to be a cryptographic hash function. Nonetheless, our results continue to hold even for such far-fetched constructions: since our adversary is information-theoretic, it is able to find all $2n$-bit inputs that yield some particular n-bit output for f_1 in constant time.

The Two-Fiber Attack. We begin by describing a simple collision-finding attack called the "two-fiber attack" which works on many natural highly-efficient constructions, including all of the fixed-key constructions from [10]. In this attack the function f_1 has a certain property, which we call the "two-fiber property." We now explain.

Let $f_1^{-1}(i)$ represent the set $\{(h,m)\colon f_1(h,m) = i\}$. This is commonly called the *fiber* of f_1 under i, or the i-fiber. We now define the notion of a well-balanced fiber or function.

Definition 3 [Well-Balanced Fibers] Fix integer $n > 0$, and let $f\colon \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ be a function. Then the fiber $f^{-1}(i)$ is *well-balanced* if each $h \in \{0,1\}^n$ appears exactly once as a first coordinate of some ordered pair of $f^{-1}(i)$. If every fiber of f is well-balanced, then we say that f is well-balanced.

An example of a well-balanced function is $f_1(h,m) = h \oplus m$. In fact, it's not hard to see that if $f_1(h,\cdot)$ is a permutation on $\{0,1\}^n$ for each $h \in \{0,1\}^n$ then f_1 will be well-balanced.

For the purposes of the present attack, we require only that there exist distinct $i_1, i_2 \in \{0,1\}^n$ such that $f_1^{-1}(i_1)$ and $f_1^{-1}(i_2)$ are well-balanced. If f_1 has two such fibers, we say that f_1 has the *two-fiber property* and the resulting attack is called the *two-fiber attack*. Notice that this property can be determined by the adversary without requiring any E-queries.

The attack is given in the theorem below. The idea behind the attack is that by doing just $2|\mathcal{K}|$ queries to E on points i_1 and i_2, an adversary can produce arbitrarily many same-length messages along with their hash values.

Theorem 4 [Two-Fiber Attack] Fix $k, n > 0$ and let $E\colon \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ be a blockcipher. Fix $\mathcal{K} \subset \{0,1\}^k$, $\mathcal{S} \subseteq \{0,1\}^*$, and let $g\colon \mathcal{S} \times \{0,1\}^n \times \{0,1\}^n \to \mathcal{S} \times \mathcal{K}$ be a key-selection function. Let the compression function $f\colon \mathrm{Perm}(n) \times (\{0,1\}^n \times \{0,1\}^n) \to \{0,1\}^n$ be defined as usual by $f^{\pi}(h,m) = f_2(h,m,\pi(f_1(h,m)))$, where f_1 has the two-fiber property. Finally, let hash function $H[g,\mathcal{K}]\colon \mathrm{Bloc}(k,n) \times \mathcal{D} \to \{0,1\}^n$ be the iterated hash of f. Then $\mathbf{Adv}_H^{\mathrm{coll}}(2|\mathcal{K}|) = 1$.

Proof: Let $i_1, i_2 \in \{0,1\}^n$, $i_1 \neq i_2$, be chosen such that $f_1^{-1}(i_1)$ and $f_1^{-1}(i_2)$ are well-balanced fibers. We now define $A^{E,E^{-1}}$, a collision-finding adversary for H. First A makes $2|\mathcal{K}|$ queries to its left oracle at (K, i_1) and (K, i_2) for each $K \in \mathcal{K}$. With the resulting values, A grows a rooted tree T. Tree T will be annotated with node-labels and edge-labels; the edge-labels will represent message blocks and the node-labels will contain the intermediate hash values obtained by traversing T from the root to that node. Edges are added by specifying an ordered pair (u, v) of node labels where u is already in T and v is a new node with label v. Thus each edge-addition always creates a leaf. Each time we add an edge to T we will also specify the label for that edge.

Let the root of T be labeled h_0. Since $h_0 \in \{0,1\}^n$ and $f_1^{-1}(i_1)$ and $f_1^{-1}(i_2)$ are well-balanced fibers, then there exist distinct m_1, m_2 such that $i_1 = f_1(h_0, m_1)$ and $i_2 = f_1(h_0, m_2)$. Therefore A can now compute $x_1 = f(h_0, m_1)$ and $x_2 = f(h_0, m_2)$ without any oracle queries since $E_K(i_1)$ and $E_K(i_2)$ have been pre-computed for any K that was output by g. If $x_1 = x_2$, then A halts returning the collsion m_1, m_2. If not, A adds an edge (h_0, x_1) labeled m_1 and an edge (h_0, x_2) labeled m_2. Then A continues at the leaves of T, doubling their number using the same technique as above; no additional oracle queries are required. This process is continued by A until a collision occurs. Since there are only 2^n possible output values for f and because the number of leaves doubles at each step, we are guaranteed that A will find a collision among the leaves within $n+1$ iterations of this process. ▮

Note that the proof holds even if E_K is not a permutation; we require only that E_K be a map from n bits to n bits. Also notice that the colliding messages produced by A are the same length; this means that a length-encoding scheme like MD-strengthening does not help avert the attack.

There are several obvious extensions to the two-fiber attack: for example, had we not insisted that messages be of the same length, a single well-balanced fiber would have sufficed. Also, if f_1 did not have the two-fiber property, perhaps it had ℓ fibers in which every $h \in \{0,1\}^n$ occurred at least twice among the first coordinates in those ℓ fibers. This would admit an analogous attack using $\ell|\mathcal{K}|$ oracle queries. Rather than pursue these ideas further, we instead proceed to the generalized attack that shows that $|\mathcal{K}|(n + \lceil \lg(n) \rceil + 2\lceil \lg(|\mathcal{K})\rceil + 1) + 1$ oracle queries are sufficient to find distinct same-length messages that collide for *any* generalized compression function.

Main Result. The central result of this paper is to show that *no* rate-1 compression function using blockcipher keys from a small fixed set $\mathcal{K}$ can give rise to a provably collision-resistant hash function when iterated. We show this by using at most $|\mathcal{K}|(n + \lceil \lg(n) \rceil + 2\lceil \lg(|\mathcal{K}|)\rceil + 1) + 1$ oracle queries to produce an overwhelming number of hash outputs that correspond to distinct messages. More specifically, our attack implements an algorithm to grow a tree of messages where the number of nodes in the tree at least doubles with each level added to it. In the case where $|\mathcal{K}| = 1$, we then show that the tree will have height at most $n + \lceil \lg(n) \rceil$ but with more than $2^n(n + \lceil \lg(n) \rceil)$ nodes which means there must exist a collision at some level of the tree. A similar pigeonhole argument is used for the case where $|\mathcal{K}| > 1$.

Although the theorems below hold for all $n > 0$, we restrict our statements to $n \geq 8$ since small values are of no interest and addressing them would introduce special cases into the proofs.

Theorem 5 [General Attack] Fix $k > 0$, $n \geq 8$ and let $E\colon \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ be a blockcipher. Fix $\mathcal{K} \subset \{0,1\}^k$, $\mathcal{S} \subseteq \{0,1\}^*$, and let $g\colon \mathcal{S} \times \{0,1\}^n \times \{0,1\}^n \to \mathcal{S} \times \mathcal{K}$ be a key-selection function. Define function $f\colon \mathrm{Perm}(n) \times (\{0,1\}^n \times \{0,1\}^n) \to \{0,1\}^n$ as usual by $f^{\pi}(h,m) = f_2(h,m,\pi(f_1(h,m)))$, with f_1 and f_2 arbitrary. Let $H[g,\mathcal{K}]\colon \mathrm{Bloc}(k,n) \times \mathcal{D} \to \{0,1\}^n$ be the iterated hash of f. Then $\mathbf{Adv}_H^{\mathrm{coll}}(|\mathcal{K}|(n + \lceil \lg(n) \rceil + 2\lceil \lg(|\mathcal{K}|)\rceil + 1) + 1) = 1$.

Proof: Like the two-fiber attack above, we will focus our attention on f_1. We will conduct the proof initially for the case $|\mathcal{K}| = 1$, and then generalize at the end. This means we may assume that g is trivial and K is fixed for all oracle queries. We construct an adversary A with oracles E, E^{-1}. We begin the proof by introducing an abstraction that will allow us to focus on the most important features of the problem. Let

$N = 2^n$ and for all $i \in [0..N-1]$ define $\mathcal{R}_i = \{(h, m, f(h, m)) \colon h, m \in \{0,1\}^n \wedge f_1(h, m) = i)\}$. Define $\mathcal{R} = \{\mathcal{R}_0, \cdots, \mathcal{R}_{N-1}\}$ and notice several things about $\mathcal{R}$:

- Over all of the $\mathcal{R}_i$ there are exactly N ordered triples of the form $(h, \cdot, \cdot)$ for each $h \in \{0,1\}^n$ since there are exactly N possible values for m.
- Since f_1 is a public function, A can sort each triple $(h, m, \cdot)$ into the appropriate set $\mathcal{R}_i$, since membership depends only on $f_1(h, m)$.
- Since evaluating $f(h, m)$ requires an oracle query, A will not initially know the value of the last coordinate of any ordered triple.

In light of the last bullet above, we will think of each triple in each $\mathcal{R}_i$ as $(h, m, ?)$ where "?" is a distinguished symbol indicating we do not yet know the value. Once we have A perform an oracle query (K, i) for some $i \in [0..N-1]$, we may fill in the last-coordinates for each triple in $\mathcal{R}_i$. Of course we are going to be stingy with our oracle queries, since we can spend at most $|\mathcal{K}|(n + \lceil \lg(n) \rceil + 2\lceil \lg(|\mathcal{K}|) \rceil + 1) + 1$ of them. Now A grows a rooted tree T, the same as we did for the two-fiber attack: tree T will be annotated with node-labels and edge-labels; the edge-labels will represent message blocks and the node-labels will contain the intermediate hash values obtained by traversing T from the root to that node. Edges are added by specifying an ordered pair (u, v) of node labels where u is already in T and v is a new node with label v. Thus each edge-addition always creates a leaf. Each time we add an edge to T we will also specify the label for that edge. Edges $(h, f(h, m))$ added to T will indicate that message block m gets us from chaining-value h to chaining-value $f(h, m)$. Therefore, we would label edge $(h, f(h, m))$ with m. We start with T having just one node labeled h_0. Because A has not yet queried E, the tree can be extended no further at this point.

Before doing any E-queries, we make the following three simplifications to the abstraction all of which *remove* power from the adversary, and therefore an attack in this simplified setting still yields an attack in the original setting.

Recall that each time A queries E at point (K, i), we may fill in all last-coordinates in set $\mathcal{R}_i$. Our first simplification is to fill in only those triples that are of immediate use. In other words, for each ordered triple $(h, m, ?)$ in $\mathcal{R}_i$, we replace "?" with $f(h, m)$ only if a node labelled h appears in T. We think of the remaining triples in $\mathcal{R}_i$ as being distributed arbitrarily among the sets $\mathcal{R}_j$ where j has not yet been queried. This adjustment clearly does not increase the power of A.

The second simpification is to impose a restriction on T: it may grow at most one level for each query A makes. This means that if A makes a query that adds an edge e to T that increases its height, A may not then extend T with a new edge attached to e. This limitation also does not increase A's power.

Finally, our third simplification is to notice that two triples (h, m, j) and (h, m', j) in the ordered triples contained in the sets of $\mathcal{R}$ only helps A. This is because repeating h and j allows an immediate collision at the same level of T as soon as h appears in T. So we will assume that for every pair of triples (h, m, j) and (h', m', j') in the sets of $\mathcal{R}$, that $h = h'$ implies $j \neq j'$. Once again, this assumption only makes A's job harder.

Before proceeding to the attack, we establish some notation. At any time during the attack, we define t as the number of nodes in T and we let $\mathcal{E}$ be the set of triples used in T thus far. (Note that the number of edges in T may be larger than the number of triples used: if several nodes labeled h appear in T, then triple (h, m, j) may be used to create an edge from each of them.) Let $\mathcal{R}^*$ be all ordered triples in $\mathcal{R}$; that is $\mathcal{R}^* = \cup_{i=0}^{N-1} \mathcal{R}_i$. Let $\overline{\mathcal{E}}$ denote the set of ordered triples not in $\mathcal{E}$. That is, $\overline{\mathcal{E}} = \mathcal{R}^* - \mathcal{E}$. The attack now proceeds as follows: define $v \colon [0..N-1] \to \mathbb{N}$ as $v(i) =$ the number of nodes in T with label i. Now A scores each unqueried set $\mathcal{R}_i$ according to the function $s(\mathcal{R}_i) = \sum_{(h,m,j) \in \mathcal{R}_i} v(h)$.

The score of $\mathcal{R}_i$ measures the number of nodes we can add to T as a direct result of querying $E(K, i)$. The tree-building algorithm for A is the natural greedy algorithm: ask the query (K, i) that maximizes $s(\mathcal{R}_i)$ where ties are broken arbitrarily. Once A has filled in the triples of $\mathcal{R}_i$, it extends T by each relevant triple available; that is, if $(h, m, f(h, m)) \in \mathcal{R}_i$ and h is a node in T, add edge $(h, f(h, m))$ to T with edge-label m. But A may be able to add further edges for already-discovered triples as well. So for each triple (h, m, j) in

Algorithm BuildTree
$J \leftarrow \{0,1\}^n$; $\quad \mathcal{E} \leftarrow \varnothing$; $\quad T \leftarrow \varnothing$; $\quad$ AddNode(T, h_0)
for $\ell \leftarrow 1$ **to** $n + \lceil \lg(n) \rceil$ **do**
$\quad i \leftarrow \max_{j \in J}\{s(\mathcal{R}_j)\}$
$\quad p \leftarrow E(K, i)$; $\quad J \leftarrow J - \{i\}$
$\quad$ **for** $(h, m, ?) \in \mathcal{R}_i$ **do**
$\quad\quad (h, m, ?) \leftarrow (h, m, f_2(h, m, p))$
$\quad$ **for** $(h, m, j) \in \mathcal{R}_i$ **do**
$\quad\quad$ **for** $v \in T$ **do**
$\quad\quad\quad$ **if** $h = v$ **then** AddEdge$(T, (v, j), m)$; $\quad \mathcal{E} \leftarrow \mathcal{E} \cup \{(v, m, j)\}$
$\quad$ **for** $(h, m, j) \in \mathcal{E}$ **do**
$\quad\quad$ **for** $v \in T$ **do**
$\quad\quad\quad$ **if** $h = v$ **and** $(v, j) \notin T$ **then** AddEdge$(T, (v, j), m)$
$\quad$ **if** collision on any level of T **then** halt

Figure 4: The tree-building algorithm used by adversary A. Set J tracks the unqueried points; set $\mathcal{E}$ tracks the triples used in tree T. The algorithm chooses the maximum-scoring set $\mathcal{R}_i$ that has not been previously queried, and queries E at (K, i). It then expands the tree using triples from the newly-discovered $\mathcal{R}_i$ and from $\mathcal{E}$. Function $\mathrm{AddEdge}(T, (u, v), m)$ inserts into tree T an edge from the node labeled u to a new leaf labeled v using edge-label m. With each iteration of the main loop, the height of T grows by exactly one while the number of nodes in T at least doubles. We stop any time a collision occurs at some level of T; we omit specifying the data structures for T and how collisions are detected.

$\mathcal{E}$, adversary A also adds an edge to any h in T where (h, j) does not already appear as an edge. See Figure 4 for the complete algorithm.

Our goal here is to argue that T increases exponentially in the number of nodes as it increases linearly in height. First, notice that an invariant of T is that $s(\mathcal{E}) + s(\overline{\mathcal{E}}) = tN$. This is because $s(\mathcal{E}) + s(\overline{\mathcal{E}}) = s(\mathcal{E} \cup \overline{\mathcal{E}}) = s(\mathcal{R}^*) = tN$. We now state and prove the key lemma.

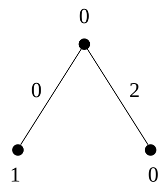
Lemma 6 [Tree-Doubling Lemma] At any point during the attack, if $t > 1$ there exists some unqueried value $i \in \{0,1\}^n$ such that querying $E(K, i)$ allows at least $t + 1$ nodes to be added to T. Furthermore, if $t = 1$ there exists a query that allows at least 1 node to be added to T.

Proof: For $t = 1$ no E-queries have been asked. Therefore there must exist some $\mathcal{R}_i \in \mathcal{R}$ with at least one triple of the form $(h_0, m, ?)$ for any $m \in \{0,1\}^n$, which means there exists some $\mathcal{R}_i$ with $s(\mathcal{R}_i) \geq 1$.

Now assume $t > 1$, which means at least one E-query has been asked by A. Thus there are at most $N - 1$ unqueried values remaining. We will now bound $s(\overline{\mathcal{E}})$. Let d be the number of "free extensions" we can make to T by applying triples already in $\mathcal{E}$. Now, notice that $s(\mathcal{E}) = t - 1 + d$. This is because $s(\mathcal{E})$ gets a score of $t - 1$ from each of the $t - 1$ added nodes thus far, but also gets an added score of d from the d triples in $\mathcal{E}$ we can add for free. Therefore $s(\overline{\mathcal{E}}) = tN - t + 1 - d$ and since this score must be distributed among at most $N - 1$ sets we can use the pigeonhole principle to show that the minimum node-expansion possible when using the maximally-scoring set $\mathcal{R}_i$ is

$$\left\lceil \frac{tN - t + 1 - d}{N - 1} \right\rceil + d = \left\lceil \frac{t(N-1)}{N-1} - \frac{d-1}{N-1} \right\rceil + d = \left\lceil t - \frac{d-1}{N-1} \right\rceil + d \geq \left\lceil t - d + 1 \right\rceil + d = t + 1.$$

■



$$\mathcal{R}_0 = \{(0,1,?),\ (1,2,?),\ (1,3,?),\ (2,2,?)\}$$
$$\mathcal{R}_1 = \{(1,0,?),\ (2,0,?),\ (3,2,?),\ (3,3,?)\}$$
$$\mathcal{R}_2 = \{(0,0,1),\ (0,2,0),\ (2,1,3),\ (3,0,0)\}$$
$$\mathcal{R}_3 = \{(0,3,?),\ (1,1,?),\ (2,3,?),\ (3,1,?)\}$$

Figure 5: An example tree built during the attack for $N = 4$. We assume $h_0 = 0^n$ and label the root accordingly. Thus far A has queried E at $(K, 2)$, so the last coordinates of $\mathcal{R}_2$ are filled in. Only the first two ordered triples of $\mathcal{R}_2$ were useful, so $\mathcal{E} = \{(0,0,1),\ (0,2,0)\}$ and the edges $(0,1)$ and $(0,0)$ were added to T with edge labels 0 and 2, respectively. Also, $t = 3$, $d = 2$, and $s(\mathcal{E}) = 4$, so $s(\overline{\mathcal{E}}) = 8$.

See Figure 5 for a small example. With this result in hand we can now conclude the proof of the theorem. Since t increases by at least 1 from the first query, and by at least $t + 1$ from subsequent queries, we can see by induction that after ℓ queries we will have at least $2^\ell + 2^{\ell-1} - 1$ nodes in T.

Let $m = \lceil \lg(n) \rceil$. Then after $n + m$ queries we are guaranteed at least $nN + 2^{m-1}N - 1$ nodes in $n + m + 1$ levels of T. Ignoring the root node, this is $nN + 2^{m-1}N - 2$ nodes in $n + m$ levels. Since $n \geq 8$ then $2^{m-1} \geq m + 1$ and

$$nN + 2^{m-1}N - 2 \geq (n + m)N + N - 2 > (n + m)N.$$

Thus there are more than $(n + \lceil \lg(n) \rceil)N$ nodes on $n + \lceil \lg(n) \rceil$ levels of T yielding a collision on some level. The same-length messages M and M' that collide under hash function H are extracted from T by traversing T from the root to each colliding node and reading off the edge labels that form the message blocks of M and M'.

Finally, we extend this attack to the case where $|\mathcal{K}| > 1$. Let $\kappa = \lceil \lg(|\mathcal{K}|) \rceil$. In this case, we make at most $|\mathcal{K}|(m + n + 2\kappa + 1) + 1$ queries. We proceed exactly as above, except that each time that attack calls for a single $E(K, i)$ query we now ask $|\mathcal{K}|$ queries $E(K, i)$, one for each $K \in \mathcal{K}$. Since g at each step will choose one $K \in \mathcal{K}$ based on the path taken in the tree, we are guaranteed to have covered the query and therefore the scoring function will still accurately count the tree expansion that would result from our queries. However, in this case we cannot be satisfied with finding one level of the tree with more than 2^n nodes, because for two messages M, M' which hash to the same intermediate value, the MD-strengthening round may use any of the keys in $\mathcal{K}$.[1] We must instead use the pigeonhole principle in several steps. First we make some extra queries such that we are guaranteed more than $|\mathcal{K}|2^n$ nodes on some level of the tree. Then there must be an intermediate hash value such that at least $|\mathcal{K}| + 1$ messages hash to that value before MD-strengthening. Thus, by a final application of the pigeonhole principle, there must be at least two of the $|\mathcal{K}| + 1$ messages which use the same key in the MD-strengthening round and, therefore, collide.

Now we show that it is sufficient to make at most $|\mathcal{K}|(m + n + 2\kappa)$ queries in order to build the tree T with the required properties. After $|\mathcal{K}|(m + n + 2\kappa)$ queries, there are at least $n|\mathcal{K}|^2N + 2^{m-1}|\mathcal{K}|^2N - 1$ nodes in $n + m + 2\kappa + 1$ levels of T. By a similar argument as above, ignoring the root node, this is more than $(n + m)|\mathcal{K}|^2N$ nodes in $n + m + 2\kappa$ levels. We now just need to show that $|\mathcal{K}|(n + m) \geq n + m + 2\kappa$ or that the following inequality holds:

$$(|\mathcal{K}| - 1)(n + m) \geq 2\kappa$$

This can be verified by noting that $n + m \geq 11$ and then by using induction. After the tree is built, we need to make an extra $|\mathcal{K}| + 1$ queries to determine the colliding messages. We therefore will make a total of $|\mathcal{K}|(n + \lceil \lg(n) \rceil + 2\lceil \lg(|\mathcal{K}|) \rceil + 1) + 1$ queries to E. ∎

[1] We may not be satisifed with the fact that if we apply MD-strengthening to all the (at least) $2^n + 1$ messages on that level we are guaranteed a collision, because in order to *find* the two messages that collide, we will likely have to query $E(\cdot, \cdot)$ too many times.

Interpreting the Result. We have used the ideal-cipher model for the blockcipher and endowed the adversary with limitless computational abilities. In this setting we were able to find an attack far more efficient than we can for known-secure constructions like MMO. However, we must realize that this model is not realistic in two ways: (1) When we plug a real blockcipher in for E, say 256-bit Rijndael, and fix a key-selection algorithm g and key-set $\mathcal{K}$, we do not then have a random object. We have a fixed public object that can be attacked via directed cryptanalysis. (2) If we attempt to mount the attacks described here, we will be using real computers with real computational limitations. Building a tree with $\Omega(2^n)$ nodes is not feasible for typical values of n. Of course collisions will appear long before the tree reaches this size, under reasonable probabilistic assumptions, but even a tree containing $\Omega(2^{n/2})$ nodes is impractical to store when n is (say) 160 or 256.

So one might reasonably ask if the attacks just shown are really of any concern at all. Perhaps we can use 256-bit Rijndael, fix a single key 0^{256}, and find some fast and simple functions f_1 and f_2 that do not admit any "obvious" attacks on the resulting iterated hash function. This may very well produce a collision-resistant hash function in the same sense that SHA-1 or RIPEMD-160 is thought to be collision resistant: no one has yet found collisions. However, we are taking a step backwards in this way of thinking because we are once again relying on the lack of effective attacks to give evidence of security. In a sense, we would be designing yet another primitive when we already have several primitives without any known attacks (at the time of this writing) and a longer established presence. But one thing we *can* guarantee about such an object is this: it will never admit a proof of security in the established model.

4 The Tweak Chain Hash

Tweakable blockciphers [6] are a map $\widetilde{E}\colon \{0,1\}^k \times \{0,1\}^t \times \{0,1\}^n \to \{0,1\}^n$ where the inputs are called the "key," the "tweak," and the "message," respectively. We sometimes write $\widetilde{E}_K(T,M)$ instead of $\widetilde{E}(K,T,M)$. For any fixed $K \in \{0,1\}^k$ and $T \in \{0,1\}^t$, we require that $\widetilde{E}(K,T,\cdot)$ is a permutation on n bits. The idea is for the tweakable blockcipher to act like a normal blockcipher but with an extra (public) input, the tweak, which adds variability. The key may be expensive to schedule and to change, but changes to the tweak should be cheap. Security for a tweakable blockcipher was defined in Section 2.

In their paper, Liskov et al. give (among other things) two proposals for constructing tweakable blockciphers from conventional blockciphers, along with several other constructions for using tweakable blockciphers. Their paper suggests a new hash-function construction built on tweakable blockciphers called the "Tweak Chain Hash" (TCH), defined as follows: for any $m \in (\{0,1\}^n)^+$ write $M = m_1 \cdots m_\ell$ where each $|m_i| = n$, define $TCH^{\widetilde{E}_K}(M)$ as

$$\begin{array}{l} \textbf{function } TCH^{\widetilde{E}_K}(m_1 \cdots m_\ell) \\ \qquad \textbf{for } i \leftarrow 1 \textbf{ to } \ell \textbf{ do } h_i \leftarrow \widetilde{E}_K(h_{i-1}, m_i) \oplus m_i \\ \qquad \textbf{return } h_\ell \end{array}$$

where h_0 is a fixed constant, say 0^n, and $\widetilde{E}$ is a tweakable blockcipher with $n = t$ and key K a constant; see Figure 3. Their is idea is that this construction should be faster than blockcipher-based constructions that rekey: the key K is fixed and only the tweak and message change for each message block digested. Since changing these two inputs should be cheap (ie, nothing equivalent to rescheduling a key should be required), each round of TCH should be faster than a round of, say, MMO. The authors leave the security of TCH as an open question. This is a question we aim to address in this section.

The First Attack: TCH-CBC. Liskov et al. give two provably-secure constructions of tweakable blockciphers from conventional blockciphers. The first construction is the CBC MAC of the two-block message $M \| T$. In other words, for a given blockcipher E they define $\widetilde{E}_K(T,M) = E_K(T \oplus E_K(M))$. They show that this construction is birthday-close to the underlying blockcipher E. That is, $\mathbf{Adv}^{\mathrm{tprp}}_{\widetilde{E}}(q) < \mathbf{Adv}^{\mathrm{prp}}_{E}(q) + q^2/2^n$. We call this the "CBC construction."

Taking the CBC construction and inserting it into the TCH construction seems like a natural try at building a collision-resistant hash function from a blockcipher. However, we immediately notice that the resulting TCH-CBC scheme is rate-1/2; that is, two blockcipher calls are required for each message block digested. (This means that our analysis from Section 3 does not apply because the compression function

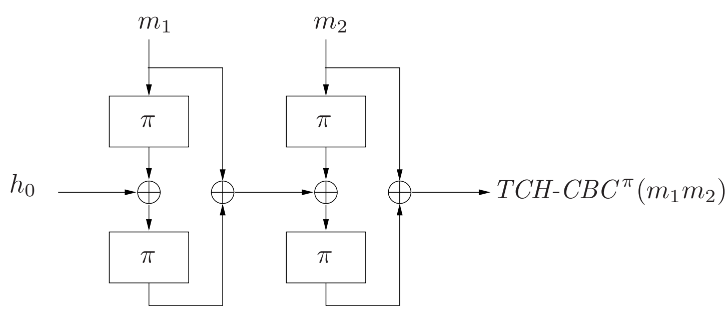


Figure 6: Two rounds of the *TCH-CBC*$^{\pi}$ hash function. Function $\pi\colon \{0,1\}^n \to \{0,1\}^n$ is a fixed permutation. We can easily generate an infinite number of same-length message pairs that collide using this construction.

here is not rate-1.) This may in fact be more expensive than a rate-1 scheme that rekeys (like MMO). But TCH-CBC would be an interesting scheme nonetheless because it fixes the blockcipher key; no secure scheme has ever been exhibited that does this.

Unfortunately TCH-CBC is not collision resistant, as we now show. Fix a key K: this induces a fixed permutation that, for notational convenience, we name $\pi = E_K$. For any $M \in (\{0,1\}^n)^+$ write $M = m_1 \cdots m_\ell$ where each $|m_i| = n$, define $TCH\text{-}CBC^{\pi}(M)$ as

function $TCH\text{-}CBC^{\pi}(m_1 \cdots m_\ell)$
  for $i \leftarrow 1$ **to** ℓ **do** $h_i \leftarrow \pi(h_{i-1} \oplus \pi(m_i)) \oplus m_i$
  return h_ℓ

where as usual, h_0 is some fixed constant. See Figure 6. Now for a two-block message $M = m_1 m_2$ we have

$$TCH\text{-}CBC^{\pi}(M) = \pi(\pi(h_0 \oplus \pi(m_1)) \oplus m_1 \oplus \pi(m_2)) \oplus m_2.$$

Let $M_c^* = \pi^{-1}(c \oplus h_0) \,\|\, c$ and notice that $h(M_c^*) = h_0$ for any $c \in \{0,1\}^n$, yielding a large number of 2-block collisions. This idea can easily be generalized to generate collisions for messages of any even number of blocks > 2 as well.

The Second Attack: TCH-AXU. The second tweakable blockcipher construction proposed by Liskov et al. is based on the use of a universal hash family [2]. The flavor they used are known as ϵ-AXU$_2$ hash families. This is the preferred flavor because it leads to an efficient tweakable-blockcipher construction with good security. However, as we will see, plugging their construction into TCH allows a simple attack, and this attack does not depend on the ϵ-AXU$_2$ property.

Definition 7 [**ϵ-AXU$_2$ Hash Families**] Fix $n > 0$. We say a set of functions $\mathcal{U} = \{u\colon \{0,1\}^n \times \{0,1\}^n\}$ is ϵ-AXU$_2$ if for all $x, y, z \in \{0,1\}^n$ with $x \neq y$,

$$\Pr_{u \in \mathcal{U}}[u(x) \oplus u(y) = z] \leq \epsilon.$$

Now let E be a blockcipher, let $\mathcal{U}$ be an ϵ-AXU$_2$ hash family whose functions map n bits to n bits and define $\widetilde{E}_{K,u}(T, M) = E_K(M \oplus u(T)) \oplus u(T)$ where $K \in \{0,1\}^k$ and $u \in \mathcal{U}$. Liskov et al. show that $\mathbf{Adv}^{\mathrm{tprp}}_{\widetilde{E}}(q) < \mathbf{Adv}^{\mathrm{prp}}_{E}(q) + 3\epsilon q^2$. We call this the "AXU construction."

Let's try inserting the AXU construction into TCH and see if the resulting TCH-AXU construction is secure. Note that the AXU construction has a longer key since both the key for the underlying blockcipher and the function u must be specified. However, since TCH is a keyless object, we once again must fix both of these keys. Of course, fixing u means selecting some single function from $\mathcal{U}$, and since $\mathcal{U}$ is an ϵ-AXU$_2$ hash family, most of the functions in this set will be "good" in the sense that they will be injective or nearly injective. However, as we will see, the properties of the particular function u are irrelevant in our attack: it is effective no matter what n-bit to n-bit function is supplied.

Once we have selected a fixed key K and function u, we have a rate-1 fixed-key blockcipher-based hash function, and our results from Section 3 immediately tell us the construction is insecure. However, it is even

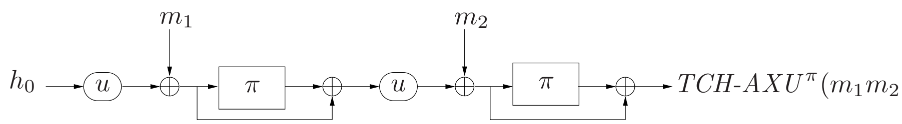


Figure 7: Two rounds of the *TCH-AXU*$^\pi$ hash function. Function $\pi\colon \{0,1\}^n \to \{0,1\}^n$ is a fixed permutation and $u\colon \{0,1\}^n \to \{0,1\}^n$ is a fixed arbitrary function. We can easily generate an infinite number of same-length message pairs that collide using this construction.

worse than this: there is a very simple attack that yields an infinite number of same-length message pairs that collide, as we now demonstrate.

Fix a key K and a function u from the family $\mathcal{U}$. For notational convenience we name $\pi = E_K$. For any $M \in (\{0,1\}^n)^+$ write $M = m_1 \cdots m_\ell$ where each $|m_i| = n$, define $TCH\text{-}AXU^\pi(M)$ as

$$
\begin{array}{l}
\textbf{function } TCH\text{-}AXU^\pi(m_1 \cdots m_\ell) \\
\quad \textbf{for } i \leftarrow 1 \textbf{ to } \ell \textbf{ do } h_i \leftarrow \pi(m_i \oplus u(h_{i-1})) \oplus m_i \oplus u(h_{i-1}) \\
\quad \textbf{return } h_\ell
\end{array}
$$

where as usual, h_0 is some fixed constant. See Figure 7. Now for a two-block message $M = m_1 m_2$ we have

$$
\begin{aligned}
TCH\text{-}AXU^\pi(M) \quad = \quad & \pi(m_2 \oplus u(\pi(m_1 \oplus u(h_0)) \oplus m_1 \oplus u(h_0))) \\
& \oplus m_2 \oplus u(\pi(m_1 \oplus u(h_0)) \oplus m_1 \oplus u(h_0)).
\end{aligned}
$$

Let $M_c^* = u(h_0) \oplus c \,\|\, u(\pi(c) \oplus c)$ and notice that $h(M_c^*) = \pi(0)$ for any $c \in \{0,1\}^n$, yielding a large number of 2-block collisions. This idea can easily be generalized to generate collisions for messages of any even number of blocks > 2 as well.

The Security of TCH. The preceding two attacks do not imply that any tweakable blockcipher constructed as a mode on a conventional blockcipher will yield an easily-breakable TCH construction. It just so happened that the two modes given by the authors did fall to simple attacks. However, we can imagine other tweakable-blockcipher constructions where attacks on the resulting TCH are not so obvious. But the results of this paper tell us that if the tweakable blockcipher were constructed from a single call to a conventional blockcipher, the resulting TCH would not have a proof of security and would therefore have to be treated as a primitive.

A New Model. It is natural to ask whether TCH works under any model for tweakable blockciphers. And it's fairly clear that extending the ideal-cipher model to the tweakable setting does the trick: let $k, t, n \geq 1$ be numbers. Define $\mathrm{TBloc}(k,t,n)$ be the set of all tweakable blockciphers $\widetilde{E}\colon \{0,1\}^k \times \{0,1\}^t \times \{0,1\}^n \to \{0,1\}^n$. Choosing a random element of $\mathrm{TBloc}(k,t,n)$ means that for each $(K,T) \in \{0,1\}^k \times \{0,1\}^t$ one chooses a random permutation $E_K(T,\cdot)$.

For TCH, we require $t = n$ and we fix the key K to some constant. But this immediately reduces to MMO in the ideal-cipher model for conventional blockciphers, which was proven secure previously [1]. We have essentially lost the distinction between the key and the tweak since in our new ideal-tweakable-cipher model they are equivalent. The notion that the tweak is public and the key is secret has been lost. The notion that the tweak should be cheap to change while the key is normally expensive to change has similarly been lost.

What does provable security in the ideal-tweakable-cipher model mean? Notice that in each of the above attacks on TCH we exploited details of the construction of the underlying tweakable blockcipher. Had we treated these underlying objects as black boxes, we would have had no effective course of attack; we can therefore conclude that any attack on TCH must exploit the internal features of the tweakable blockcipher upon which it is constructed, meaning that perhaps a a tweakable blockcipher *primitive* might yield a secure TCH. The Hasty Pudding cipher is the only tweakable blockcipher primitive we know of [12]. Whether using Hasty Pudding in TCH yields an efficient collision resistant hash function is left as an open question, but we can be certain that any attacks on TCH-HP would require the cryptanalyst delve into the inner workings of the Hasty Pudding cipher.

5 Conclusion and Open Problems

Our results give strong evidence that we cannot build rate-1 collision-resistant hash functions from a blockcipher that uses only a small set of keys. Does this mean we are forced to accept constructions that change the key arbitrarily with each round if we want provable security? Not necessarily. Our results say nothing about schemes in this framework that rekey, say, every other round. It would be interesting to show sufficient conditions on how often the blockcipher must be rekeyed in order to maintain a good collision resistance bound. Alternatively, perhaps the key can be fixed in a non-Merkle-Damgård construction; the results of Gennaro et al. [4], although for a different security property than we considered here, may provide some insight. Or perhaps there is some relaxation of the model and weakening of the adversary that admit security proofs for highly-efficient blockcipher-based schemes. We leave these as open questions.

Acknowledgements

Thanks to Phillip Rogaway for suggesting to spell "blockcipher" as a single word (it saved typing a hyphen more than 20 times) and for various suggestions and comments on an early draft of this manuscript. Thanks as well to several Eurocrypt 2005 reviewers for their insightful suggestions. John Black's work was supported by NSF CAREER-0240000 and a gift from the Boettcher Foundation. Part of this work was conducted while Tom Shrimpton was at UC Davis and was supported by NSF 0208842, NSF 0085961, and a gift from Cisco Systems. We would also like to thank Martijn Stam for finding an error in the previous version of this paper, and for suggesting the fix.

References

[1] Black, J., Rogaway, P., and Shrimpton, T. Black-box analysis of the block-cipher-based hash-function constructions from PGV. In *Advances in Cryptology – CRYPTO '02* (2002), vol. 2442 of *Lecture Notes in Computer Science*, Springer-Verlag.

[2] Carter, L., and Wegman, M. Universal hash functions. *J. of Computer and System Sciences*, 18 (1979), 143–154.

[3] Damgård, I. A design principle for hash functions. In *Advances in Cryptology – CRYPTO '89* (1990), G. Brassard, Ed., vol. 435 of *Lecture Notes in Computer Science*, Springer-Verlag.

[4] Gennaro, R., Gertner, Y., Katz, J., and Trevisan, L. Bounds on the efficiency of generic cryptographic constructions, 2005. To appear in the *SIAM Journal on Computing*.

[5] Handschuh, H., Knudsen, L., and Robshaw, M. Analysis of SHA-1 in encryption mode. In *Advances in Cryptology – CT-RSA '01* (2001), D. Naccache, Ed., Lecture Notes in Computer Science, Springer-Verlag, pp. 70–83.

[6] Liskov, M., Rivest, R., and Wagner, D. Tweakable block ciphers. In *Advances in Cryptology – CRYPTO '02* (2002), M. Yung, Ed., Lecture Notes in Computer Science, Springer-Verlag, pp. 31–46.

[7] Matyas, S., Meyer, C., and Oseas, J. Generating strong one-way functions with cryptographic algorithms. *IBM Technical Disclosure Bulletin 27*, 10a (1985), 5658–5659.

[8] Menezes, A., van Oorschot, P., and Vanstone, S. *Handbook of Applied Cryptography.* CRC Press, 1996.

[9] Merkle, R. One way hash functions and DES. In *Advances in Cryptology – CRYPTO '89* (1990), G. Brassard, Ed., vol. 435 of *Lecture Notes in Computer Science*, Springer-Verlag.

[10] Preneel, B., Govaerts, R., and Vandewalle, J. Hash functions based on block ciphers: A synthetic approach. In *Advances in Cryptology – CRYPTO '93* (1994), Lecture Notes in Computer Science, Springer-Verlag, pp. 368–378.

[11] Rabin, M. Digitalized signatures. In *Foundations of Secure Computation* (1978), R. DeMillo, D. Dobkin, A. Jones, and R. Lipton, Eds., Academic Press, pp. 155–168.

[12] Schroeppel, R., and Orman, H. The hasty pudding cipher. AES candidate submitted to NIST, 1998.

[13] Shannon, C. Communication theory of secrecy systems. *Bell Systems Technical Journal 28*, 4 (1949), 656–715.

[14] Simon, D. Finding collsions on a one-way street: Can secure hash functions be based on general assumptions? In *Advances in Cryptology – EUROCRYPT '98* (1998), Lecture Notes in Computer Science, Springer-Verlag, pp. 334–345.

[15] Winternitz, R. A secure one-way hash function built from DES. In *Proceedings of the IEEE Symposium on Information Security and Privacy* (1984), IEEE Press, pp. 88–90.