On the Impossibility of Highly Efficient Blockcipher-Based Hash Functions

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

February 24, 2004

Abstract

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 single underlying key. 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 black-box model, that it is *impossible* to construct a highly efficient blockcipher-based hash function which is provably secure. Our result implies, in particular, that the Tweakable Chain Hash (TCH) construction suggested by Liskov, Rivest, and Wagner [3] 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.

1 Introduction

Background. Preneel, Govaerts, and Vandewalle [6] considered turning a blockcipher E into a hash function by iterating a compression function f that is derived from E; for v a fixed n-bit constant, they considered 64 compression functions f of the form $f(h_{i-1}, m_i) = E_a(b) \oplus c$ where $a, b, c \in \{h_{i-1},\ m_i,\ h_{i-1} \oplus m_i,\ v\}$. They then construct hash function $H^f : (\{0,1\}^n)^* \to \{0,1\}^n$ via

```
function $H^f(m_1 \cdots m_\ell)$
    for $i \leftarrow 1$ to $\ell$ do $h_i \leftarrow f(h_{i-1}, m_i)$
    return $h_\ell$
```

where h_0 is a fixed constant and $|m_i| = n$ for each $i \in [1..\ell]$. Using an attack-based approach, PGV evaluated the security of each of the schemes. Black, Rogaway and Shrimpton [1] considered these same 64 constructions; using a proof-based approach they showed that, in the black-box model, 20 of the 64 schemes are collision-resistant up to the birthday bound. However in each of these 20 secure schemes, the blockcipher key is changed every round. Figure 1 gives one example of a provably-secure scheme.

Changing the key every round has unfortunate implications for the efficiency of these constructions, since scheduling a new key entails a significant cost. We therefore term a blockcipher-based hash function *highly efficient* if its compression function uses exactly one call to a blockcipher whose key is fixed. It is very likely that the many researchers who have worked on blockcipher-based hash functions over the past 25 years have considered highly efficient constructions, but found that each scheme proved to be insecure. Indeed, the present authors also spent some time trying to find highly efficient constructions without success. We now explain why.

[*] 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 Electrical and Computer Engineering, University of California, Davis, California, 95616, USA. E-mail: teshrim@ucdavis.edu WWW: www.ece.ucdavis.edu/~teshrim/

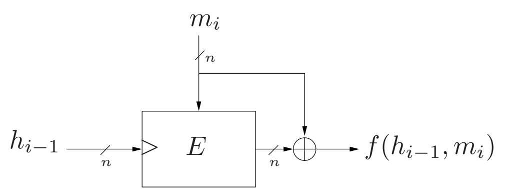


Figure 1: The Matyas-Meyer-Oseas (MMO) compression function [4]. $E\colon \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ is a block cipher; the hatch mark denotes the location of the key. Iterating this compression function results in a provably-secure hash function [1], however notice that the above compression function will be rekeyed each round.

Main Result. We would like to construct a highly-efficient hash function which 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 $E\colon \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ is a blockcipher and $K \in \{0,1\}^k$ is some public constant.

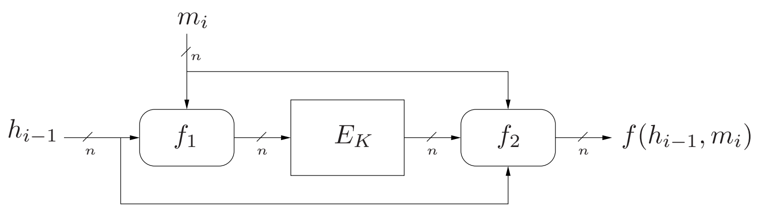


Figure 2: The general compression function built from a blockcipher with a fixed key. Functions $f_1\colon \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ and $f_2\colon \{0,1\}^n \times \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ are arbitrary; E_K is some n-bit blockcipher with key K, a fixed public value.

It isn't hard see that this construction captures all possible rate-1 compression functions built from a blockcipher with fixed-key: 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 compute $f_1(h_{i-1}, m_i)$ itself.

In this paper we prove that any compression function constructed as just described *cannot* produce a collision-resistant hash function when iterated. Specifically, we show—in the black-box model—that for any functions f_1, f_2, there exists an information-theoretic adversary which finds a collision in the iterated function H^f in at most $n + \lceil \lg(n) \rceil$ blockcipher invocations. This is in stark contrast to the $\Omega(2^{n/2})$ expected invocations needed to produce a collision in the 20 rekeying constructions known to be secure [1]. Our proof is constructive: it describes an attack using a greedy algorithm which builds large numbers of messages along with their associated hash outputs. 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.

The Tweak Chain Hash. Tweakable blockciphers were introduced by Liskov, Rivest, and Wagner [3]. 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 hash-function construction built on tweakable blockciphers called the "Tweak Chain Hash" (TCH). This is a straightforward

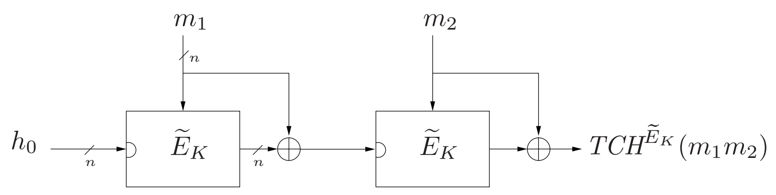


Figure 3: Two rounds of the Tweak Chain Hash. $\widetilde{E}_K$ is the tweakable blockcipher, K is a fixed public key; the arc denotes the location of the tweak.

adaptation of the MMO construction (Figure 1) 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 $\ell$ do $h_i \leftarrow \widetilde{E}_K(h_{i-1}, m_i) \oplus m_i$
    return $h_\ell$
```

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 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 black-box 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 [7]. This is discussed further in Section 4.

Security Model. The usual assumption for blockciphers is that they are good pseudo-random permutations [5]. However this assumption seems to be insufficient for proving the security of hash functions based on blockciphers; indeed, Simon has shown [9] that there always exists an efficient adversary which finds collisions in any hash function based on a blockcipher if we assume only that the blockcipher is a pseudo-random permutation. For this reason, all proofs of security in this domain are done in the black-box model [1, 5, 9, 10]. This model, which dates back to Shannon [8], treats a blockcipher as a random and independent permutation for each key.

Except for the two simple attacks given for TCH, all attacks in this paper are in the black-box model. Our fixed-key blockciphers are treated as random permutations and queries to this object will be the measured adversarial resource. In reality, our main results require neither that the fixed-key blockcipher be random, nor even a permutation: we do not make any probabilistic assumptions nor do we depend on the permutivity of the blockcipher.

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 [5]. 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 n bits. We call a blockcipher E that always uses the same key $K \in \{0,1\}^k$ a *fixed-key blockcipher.* Let $n > 0$ and let $\text{Perm}(n)$ be the set of all permutations on $\{0,1\}^n$. We define a (fixed-key-blockcipher-based) *hash function* as a map $H\colon \text{Perm}(n)\times\mathcal{D} \to \{0,1\}^c$ where $n, c \geq 1$, and $\mathcal{D} \subseteq \{0,1\}^*$. The function H must be computed by a program that, given $M \in \mathcal{D}$, computes $H^\pi(M) = H(\pi, M)$ using a π-oracle. A function $f\colon \text{Perm}(n) \times \mathcal{D} \to \{0,1\}^c$ is a (fixed-key-blockcipher-based) *compression function* if $\mathcal{D} = \{0,1\}^a \times \{0,1\}^n$ for some $a \geq 1$ where $a+n \geq c$. Fix $h_0 \in \{0,1\}^a$. The *iterated hash* of compression function $f\colon \text{Perm}(n) \times (\{0,1\}^a \times \{0,1\}^n) \to \{0,1\}^a$ is the hash function $H\colon \text{Perm}(n) \times \mathcal{D} \to \{0,1\}^a$ defined by $H^\pi(m_1 \ldots m_\ell) = h_\ell$ where $h_i = f^\pi(h_{i-1}, m_i)$ and $|m_i| = n$ for each $i \in [1..\ell]$. If the program for f uses a single query $\pi(x)$ to compute $f^\pi(h, m)$ then f (and its iterated hash H) is *rate-1.* When it is understood from context, we will omit the superscript π to f and H.

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 fixed-key-blockcipher-based hash function, we model the fixed-key blockcipher as a randomly chosen $\pi \in \text{Perm}(n)$. An adversary A is given oracles for $\pi(\cdot)$ and its inverse $\pi^{-1}(\cdot)$, and wants to find a *collision* for H^π—that is, $M, M' \in \mathcal{D}$ where $M \neq M'$ but $H^\pi(M) = H^\pi(M')$. 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 $n, c > 0$. Let H be a fixed-key, blockcipher-based hash function, $H\colon \text{Perm}(n) \times \mathcal{D} \to \{0,1\}^c$, and let A be an adversary. Then the advantage of A in finding collisions in H is the real number

$$\mathbf{Adv}_H^{\text{coll}}(A) \quad = \quad \Pr\left[\pi \xleftarrow{\$} \text{Perm}(n);\, (M, M') \xleftarrow{\$} A^{\pi,\pi^{-1}} :\ M \neq M' \wedge H^\pi(M) = H^\pi(M')\right]$$

For $q \geq 1$ we write $\mathbf{Adv}_H^{\text{coll}}(q) = \max_A\{\mathbf{Adv}_H^{\text{coll}}(A)\}$ 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 $\widetilde{\pi} \xleftarrow{\$} \{0,1\}^t \times \text{Perm}(n)$ 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

$$\mathbf{Adv}_E^{\text{prp}}(A) \quad = \quad \Pr[K \xleftarrow{\$} \mathcal{K}:\ A^{E_K(\cdot)} = 1] - \Pr[\pi \xleftarrow{\$} \text{Perm}(n):\ A^{\pi(\cdot)} = 1]$$

$$\mathbf{Adv}_{\widetilde{E}}^{\text{tprp}}(A) \quad = \quad \Pr[K \xleftarrow{\$} \{0,1\}^k:\ A^{\widetilde{E}_K(\cdot,\cdot)} = 1] - \Pr[\widetilde{\pi} \xleftarrow{\$} \{0,1\}^t \times \text{Perm}(n):\ A^{\widetilde{\pi}(\cdot,\cdot)} = 1]$$

For $q \geq 1$ we write $\mathbf{Adv}_E^{\text{prp}}(q) = \max_A\{\mathbf{Adv}_E^{\text{prp}}(A)\}$ and $\mathbf{Adv}_{\widetilde{E}}^{\text{tprp}}(q) = \max_A\{\mathbf{Adv}_{\widetilde{E}}^{\text{tprp}}(A)\}$ 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 two blockcipher invocations) and we argue that it probably applies to many "reasonable" constructions for the compression function. The second is fully general, and proves that there cannot exist an iterated hash function based on this type of compression function which is collision resistant in our model.

Generalized Rate-1 Blockcipher-based Compression Function. We consider any compression function f which is built in the following way. Let $f_1\colon \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$ and $f_2\colon \{0,1\}^n \times \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$ be arbitrary functions. We define $f\colon \mathrm{Perm}(n) \times (\{0,1\}^n \times \{0,1\}^n) \rightarrow \{0,1\}^n$ as $f^\pi(h,m) = f_2(h,m,\pi(f_1(h,m)))$. See Figure 2. We will not formally argue that this construction covers all possible $2n$ to n bit functions which 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 "pre-processing" 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 [6], for example, these functions are never more complex than XOR. It would make little sense to have f_1 be, say, MD5 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 which yield some particular n-bit output for f_1 in constant time.

The Two-Fiber Attack. When the function f_1 exhibits a special property, which we will call the "two-fiber property," we can mount a very simple attack. 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 an integer $n > 0$, and let $f\colon \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$ 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 π-queries.

The attack is given in the theorem below. The idea behind the attack is that by doing just two π-queries 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] Let $f_1\colon \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$ be an arbitrary function with the two-fiber property where 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 fibers. Let $f_2\colon \{0,1\}^n \times \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$ be an arbitrary function and $\pi\colon \{0,1\}^n \rightarrow \{0,1\}^n$ be a permutation. Let compression function $f\colon \mathrm{Perm}(n) \times (\{0,1\}^n \times \{0,1\}^n) \rightarrow \{0,1\}^n$ be defined by $f^\pi(h,m) = f_2(h,m,\pi(f_1(h,m)))$. Let $H\colon \mathrm{Perm}(n) \times \mathcal{D} \rightarrow \{0,1\}^n$ be the iterated hash of f. Then $\mathbf{Adv}_H^{\mathrm{coll}}(2) = 1$.

Proof: Let $A^{\pi,\pi^{-1}}$ be a collision-finding adversary for H. First A queries its left oracle at i_1 and i_2 receiving the values $\pi(i_1)$ and $\pi(i_2)$ in return. With these 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 $\pi(i_1)$ and $\pi(i_2)$ have been pre-computed. 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 π is not a permutation; we require only that π 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 k fibers in which every $h \in \{0,1\}^n$ occurred at least twice among the first coordinates in those k fibers. This would admit an analogous attack using k oracle queries. Rather than pursue these ideas further, we instead skip to the generalized attack which shows that $n + \lceil \lg(n) \rceil$ oracle queries are sufficient to find distinct same-length messages which collide for *any* generalized compression function.

Main Result. The central result of this paper is to show that *every* rate-1 compression function which does not rekey the blockcipher cannot give rise to a collision-resistant hash function when iterated. We prove this partly reusing ideas from the two-fiber attack: we give an attack which requires at most $n + \lceil \lg(n) \rceil$ 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. 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. Although the theorem below holds for all $n > 0$, we restrict our statement to $n \geq 8$ since small values are of no interest and addressing them would introduce special cases into the proof.

Theorem 5 [General Attack] Fix $n \geq 8$ and 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. Let $\pi\colon \{0,1\}^n \to \{0,1\}^n$ be a permutation. Let compression function $f\colon \mathrm{Perm}(n) \times (\{0,1\}^n \times \{0,1\}^n) \to \{0,1\}^n$ be defined by $f^\pi(h,m) = f_2(h, m, \pi(f_1(h,m)))$. And let $H\colon \mathrm{Perm}(n) \times \mathcal{D} \to \{0,1\}^n$ be the iterated hash of f. Then $\mathbf{Adv}_H^{\mathrm{coll}}(n + \lceil \lg(n) \rceil) = 1$.

Proof: Like the two-fiber attack above, we will focus our attention on f_1. Let A be our adversary with oracles π, π^{-1}. We begin the proof by introducing an abstraction which will allow us to focus on the most important features of the problem. Let $N = 2^n$ and consider the set $\mathcal{R}$ which contains sets R_0 through R_{N-1}. Each set R_i contains zero or more ordered triples. For all $i \in [0..N-1]$ we define $R_i = \{(h, m, f(h,m))\colon h, m \in \{0,1\}^n \wedge f_1(h,m) = i)\}$. Let's notice several things about $\mathcal{R}$:

- There are exactly N ordered triples of the form $(h, \cdot, \cdot)$ in $\mathcal{R}$ 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 know the first two coordinates of every triple in $\mathcal{R}$ and therefore group them into the appropriate set R_i.
- 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 $\mathcal{R}$.

In light of the last bullet above, we will think of each triple in $\mathcal{R}$ as $(h, m, ?)$ where "?" is a distinguished symbol indicating we do not yet know the value. Once we have A perform an oracle query at i, we may fill in the last-coordinates for each triple in R_i. Of course we are going to be stingy with our oracle queries, since we can spend at most $n + \lceil \lg(n) \rceil$ 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 π, the tree can be extended no further at this point.

Before doing any π-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 π at point i, we may fill in all last-coordinates in set R_i. Our first simplification is to fill in only those triples which are of immediate use. In other words, for each ordered triple $(h, m, ?)$ in R_i, we replace "?" with $f(h, m)$ only if h is a node in T. We think of the remaining triples in R_i as being distributed arbitrarily among the sets 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 which 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 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 $\mathcal{R}$, 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 k as the number of nodes in T and we let 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 some node labeled h appears multiple times in T, then triple (h, m, j) may be used to create an edge from each node labeled h.) Let $\mathcal{R}^*$ be all ordered pairs in $\mathcal{R}$; that is $\mathcal{R}^* = \cup_{i=0}^{N-1} R_i$. Let $\bar{E}$ denote the set of ordered triples not in E. That is, $\bar{E} = \mathcal{R}^* - E$.

The attack now proceeds as follows: A scores each unqueried set R_i according to the function $s\colon R \to \mathbb{N}$. We define $v\colon [0..N-1] \to \mathbb{N}$ such that $v(i) =$ the number of times node i appears in T and define $s(R) = \sum_{(h,m,j) \in R} v(h)$.

The score of R_i measures the number of nodes we can add to T as a direct result of querying π at i. The tree-building algorithm for A is the natural greedy algorithm: ask the query i which maximizes $s(R_i)$ where ties are broken arbitrarily. Once A has filled in the triples of R_i, it extends T by each relevant triple available; that is, if $(h, m, f(h, m)) \in R_i$ 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 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 $s(E) + s(\bar{E}) = kN$. This is because $s(E) + s(\bar{E}) = s(E \cup \bar{E}) = s(\mathcal{R}^*) = kN$. We now state and prove the key lemma.

```
**Algorithm** BuildTree($n, \mathcal{R}, f_2, \pi, h_0$)
$J \leftarrow \{0,1\}^n$;    $E \leftarrow \varnothing$;    $T \leftarrow \varnothing$;    AddNode($T, h_0$)
**for** $\ell \leftarrow 1$ **to** $n + \lceil \lg(n) \rceil$ **do**
    $i \leftarrow \max_{j \in J}\{s(R_j)\}$
    $p \leftarrow \pi(i)$;    $J \leftarrow J - \{i\}$
    **for** $(h, m, ?) \in R_i$ **do**
        $(h, m, ?) \leftarrow (h, m, f_2(h, m, p))$
    **for** $(h, m, j) \in R_i$ **do**
        **for** $v \in T$ **do**
            **if** $h = v$ **then** AddEdge($T, (v, j), m$);    $E \leftarrow E \cup \{(v, m, j)\}$
    **for** $(h, m, j) \in E$ **do**
        **for** $v \in T$ **do**
            **if** $h = v$ **and** $(v, j) \notin T$ **then** AddEdge($T, (v, j), m$)
    **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 E tracks the triples used in tree T. The algorithm chooses the maximum-scoring set R_i which has not been previously queried, and queries π at i. It then expands the tree using triples from the newly-discovered R_i and from E. Function 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.

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

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

Now assume $k > 1$, which means at least one π-query has been asked by A. Thus there are at most $N - 1$ unqueried values remaining. We will now bound $s(\bar{E})$. Let d be the number of "free extensions" we can make to T by applying triples already in E. Now, notice that $s(E) = k - 1 + d$. This is because $s(E)$ gets a score of $k - 1$ from each of the $k - 1$ added nodes thus far, but also gets an added score of d from the d triples in E we can add for free. Therefore $s(\bar{E}) = kN - k + 1 - d$ 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 R_i is

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

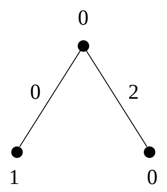
▮

See Figure 5 for a small example. With this result in hand we can now conclude the proof of the theorem. Since k increases by at least 1 from the first query, and by at least $k + 1$ 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.



$$R_0 = \{(0,1,?),\ (1,2,?),\ (1,3,?),\ (2,2,?)\}$$
$$R_1 = \{(1,0,?),\ (2,0,?),\ (3,2,?),\ (3,3,?)\}$$
$$R_2 = \{(0,0,1),\ (0,2,0),\ (2,1,3),\ (3,0,0)\}$$
$$R_3 = \{(0,3,?),\ (1,1,?),\ (2,3,?),\ (3,1,?)\}$$

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 π at 2, so the last coordinates of R_2 are filled in. Only the first two ordered triples of R_2 were useful, so $E = \{(0,0,1),\ (0,2,0)\}$ and the edges $(0,1)$ and $(0,0)$ were added to T with edge labels 0 and 2, respectively. Also, $k = 3$, $d = 2$, and $s(E) = 4$, so $s(\bar{E}) = 8$.

The same-length messages M and M' which collide under H^f are extracted from T by traversing T from the root to each colliding node and reading off the edge labels which form the message blocks of M and M'. ■

Interpreting the Result. We used the black-box model for the blockcipher (although, as we have pointed out, π could be replaced with any n-bit to n-bit function) and we 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 π, say 256-bit Rijndael, and fix its key, we do not have some random object. We have a fixed public object which 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 the key to be 0^{256}, and find some fast and simple functions f_1 and f_2 which do not admit any "obvious" attacks on the iterated hash function which results.

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. 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 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 any established model.

4 The Tweak Chain Hash

Tweakable blockciphers [3] 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 given 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

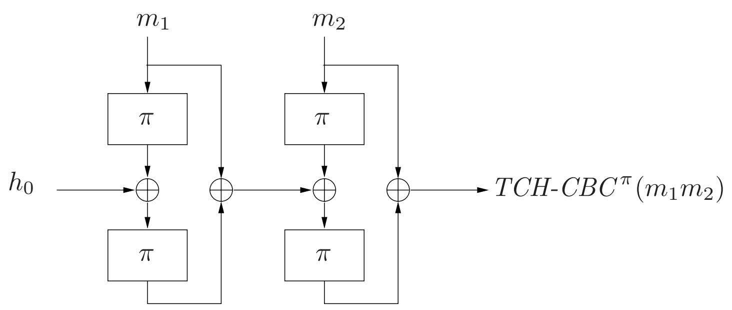


Figure 6: Two rounds of the *TCH-CBC*$^{\pi}$ hash function. Function $\pi\colon \{0,1\}^n \to \{0,1\}^n$ is a fixed permutation. We can easily generate an infinite number of same-length message pairs which collide using this construction.

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

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 here is not rate-1.) This is probably more expensive than a rate-1 scheme which 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 which does this.

Unfortunately TCH-CBC is not collision resistant, as we now show. Fix a key K: this induces a fixed permutation which 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 $\ell$ do $h_i \leftarrow \pi(h_{i-1} \oplus \pi(m_i)) \oplus m_i$
    return $h_\ell$
```

where as usual, h_0 is some fixed constant. See Figure 6. Now for a two-block message $M = m_1m_2$ 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 $M^* = \pi^{-1}(c \oplus h_0) \| c$ and notice that $h(M^*) = h_0$ 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.

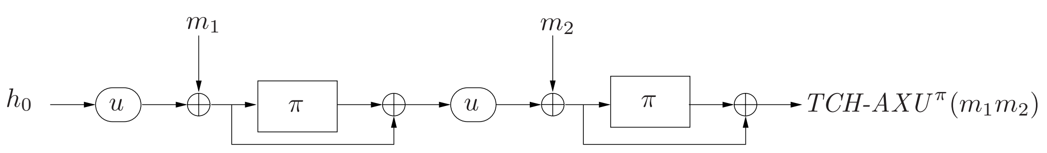


Figure 7: Two rounds of the *TCH-AXU*$^{\pi}$ hash function. Function $\pi\colon \{0,1\}^n \to \{0,1\}^n$ is a fixed permutation and $u\colon \{0,1\}^n \to \{0,1\}^n$ is a fixed arbitrary function. We can easily generate an infinite number of same-length message pairs which collide using this construction.

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 worse than this: there is a very simple attack which yields an infinite number of same-length messages which 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

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

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

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

Let $M^* = u(h_0) \oplus c \,\|\, u(\pi(c)\ \oplus c)$ and notice that $h(M^*) = \pi(0)$ 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.

Applying our Main Result to 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. Let’s look at an example.

Let $n > 0$ be a number. Let E be a blockcipher and let $\widetilde{E}\colon \{0,1\}^{2n} \times \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ be defined as $\widetilde{E}_{K,L}(T, M) = E_K(L \cdot T \oplus M)$ where the indicated multiplication is done in the finite field GF(2^n) under some fixed (public) irreducible polynomial. It turns out that this tweakable blockcipher has good security bounds, but we'll omit proving this. We call this the "XE1 construction."

Plugging XE1 into TCH we get the following blockcipher-based hash function: first (as usual) fix string $K, L \in \{0,1\}^n$. (We will require that $L \notin \{0,1\}$; without this the resulting TCH-XE1 construction becomes easily breakable.) Once again, 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 $\mathit{TCH\text{-}XE1}^{\pi}(M)$ as

```
function $\mathit{TCH\text{-}XE1}^{\pi}(m_1 \cdots m_\ell)$
    for $i \leftarrow 1$ to $\ell$ do $h_i \leftarrow \pi(L \cdot h_{i-1} \oplus m_i) \oplus m_i$
    return $h_\ell$
```

where h_0 and L are fixed constants with $L \notin \{0,1\}$.

We know of no simple attack for TCH-XE1. Perhaps it is "secure" in the same sense that SHA-1 is "secure." But TCH-XE1 is rate-1, uses a blockcipher with a fixed key and is an iterated hash function. Therefore we know from our results in Section 3 that TCH-XE1 cannot be proven secure in the black-box model and so it must 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 black-box model to the tweakable setting does the trick: let $k, t, n \geq 1$ be numbers. Define 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 TBloc(κ, 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 black-box 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 black-box 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 this new black-box 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 [7]. 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 fixed-key blockcipher. Does this mean we are forced to accept constructions which change the key 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. We might try relaxing other constraints; for example, perhaps there is a method which does not use the Merkle-Damgård construction. Or perhaps two blockcipher invocations per message block is faster than running the key schedule and one blockcipher invocation (though this almost certainly not true for AES).

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] 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.

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

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

[6] 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.

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

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

[9] 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.

[10] 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.