Provable Security Support for the Skein Hash Family

Version 1.0, April 29, 2009

Mihir Bellare	University of California San Diego, mihir@cs.ucsd.edu
Tadayoshi Kohno	University of Washington, yoshi@cs.washington.edu
Stefan Lucks	Bauhaus-Universität Weimar, stefan.lucks@uni-weimar.de
Niels Ferguson	Microsoft Corp., niels@microsoft.com
Bruce Schneier	BT Group plc, schneier@schneier.com
Doug Whiting	Hifn, Inc. dwhiting@hifn.com
Jon Callas	PGP Corp., jon@pgp.com
Jesse Walker	Intel Corp., jesse.walker@intel.com

1 Introduction

Skein [13] is a fast, versatile, and secure hash function that has been submitted as an AHS candidate. One of Skein's features is that it is backed by security proofs. This paper presents, explains, and justifies the various provable security claims underlying Skein.

1.1 Background and Summary

The base (also called atomic) primitives underlying Skein are the tweakable block cipher Threefish and derived compression function. Skein is built on top of these in some mode of operation. A proof that Skein possesses some security property S is a proof of a statement of the form: "If the atomic primitive has security property A then Skein is guaranteed to have security property S." The proof takes the form of a reduction which, given an attacker violating property S of Skein, constructs an attacker violating property A of the atomic primitive. We will be providing such proofs for various different choices of S.

It should be understood that a proof of security does not say that it would be impossible to find attacks violating security property S for Skein. What it says is that it would be impossible to find such attacks without uncovering attacks violating security property A of the atomic primitive. The proof transfers confidence from the atomic primitive to Skein. It validates the mode of operation, meaning, the higher-level design. It says that there are no flaws in this design. The practical consequence is that cryptanalysis can be confined to the atomic primitives. There is no need to attempt to attack Skein itself. You may as well invest your effort in attacking Threefish and the compression function.

The first and most basic property about which we have proofs is collision resistance. However, this isn't the only security property we support via proofs. A look at the contemporary usage of hash functions makes it clear that they are used in ways that call for security properties well beyond, and different from, collision resistance. In particular, hash functions are used for message authentication and as pseudorandom functions (PRFs) in key derivation. (These usages refer to keyed versions of the hash function, e.g., HMAC [2, 1].) They are also used to instantiate random oracles in public key cryptography schemes. We believe that this type of usage will continue, and modern hash functions should support it. This is the design philosophy that has underlain Skein.

We approach providing provable support for these additional properties by showing that the mode of operation underlying Skein is MPP (Multi-Property Preserving) [5]. This means that a number of different security attributes, if possessed by the atomic primitive, are guaranteed to be possessed by Skein. The first such property is collision resistance. The second is the PRF property, as a consequence of which we obtain provable support for the use of keyed Skein as a KDF and MAC. The third is being a pseudorandom oracle (PRO), meaning indifferentiability from a random oracle.

One of the most widespread current usages of hash functions is for HMAC [2, 16]. This use is supported by proofs of security for the current generation of hash functions that use Merkle-Damgård mode [2, 1]. We expect that any future hash function will continue to be utilized in HMAC mode and that such use should continue to be supported by proofs of security. We supply these proofs.

We also provide provable support the use of Skein as a PRNG and as a stream cipher.

Figure 1 summarizes the provable security results about Skein. We now discuss these items in more detail.

Skein property / mode	Assumption on atomic primitive
Hash (collision-resistance)	C is collision resistant
PRF, KDF, MAC, HMAC, PRNG, stream cipher	Threefish is a (tweakable) PRP
Indifferentiability from RO	Threefish is an ideal (tweakable) cipher

Figure 1: Summary of provable security attributes of Skein. For each property we indicate the assumption on the atomic primitive under which it is established. Here C is the compression function.

1.2 Provable Properties of Skein

Collision Resistance. We prove that if the compression function is collision resistant then so is Skein. (Referring to the above discussion, here S is the collision resistance of Skein and A is the collision resistance of the compression function.) The implication is that it is no easier to find collisions for Skein than for its compression function. Given that (strengthened) Merkle-Damgård [11, 21], used in the SHA family, is backed by a similar security guarantee, such a guarantee would seem to be a necessary requirement for a new hash function. We are asserting that we can provide this.

PRF, MAC, and KDF. We prove that if Threefish is a tweakable PRP (pseudorandom permutation) then Skein is a PRF. It is important to understand that we are referring, in this context, to the keyed version of Skein. The PRF property is that the input-output behavior of keyed Skein should look like that of a random function to an attacker who *is not given the key.* This proof supports the usage of keyed Skein for key derivation (KDF). It also supports the use of keyed Skein as a MAC. This is true because any PRF is a secure MAC [4].

The PRF property reflects the increased versatility of Skein compared to the SHA family. The functions in the latter family are not PRFs. (When keyed in the natural way, namely, via the initial vector.) This is because of the extension attack.

We highlight an attractive feature of the proof of PRF security. Namely, the assumption made pertains to the (tweakable) block cipher rather than to the compression function, and, moreover, is in fact the standard assumption on this object, namely, that it is a PRP. Indeed, in the case of other modes such as EMD [5] which are PRF preserving, the assumption is that the compression function is a PRF, which relies on the underlying block cipher being a PRF when keyed through the message rather than the key port. The difference in the case of Skein arises because the compression function runs the block cipher in Matyas-Meyer-Oseas mode [19].

We emphasize that we provide provable support for the use of keyed Skein as a MAC. This is by dint of the fact that we show that keyed Skein is a secure MAC under the assumption that Threefish is a PRP. (This in turn is because, as indicated above, under this assumption, keyed Skein is a PRF, and any PRF is a secure MAC.)

A novel feature of Skein in these modes is the variable output length. The desired output length is one of the inputs to the hash function. Skein has been designed so that its output values are independent for different values of this output length parameter, even if other inputs (such as the message) are the same. This attribute of Skein is also supported by the security proofs. We define the (new) concept of a VOL (Variable Output Length) PRF. This is what the proofs show Skein

to achieve under the assumption that Threefish is a PRP.

Keyed Skein is a fast alternative to HMAC-Skein with regard to providing a PRF and secure MAC. To support legacy applications, however, we will also support HMAC-Skein via proofs.

Indifferentiability from a Random Oracle. We prove that the Skein mode of operation preserves indifferentiability from a random oracle. This has, since [10, 5], become an important requirement for hash functions due to their use for instantiating random oracles.

What the results says is that if we replace Threefish with an ideal block cipher, then the resulting hash function produced by the Skein mode of operation is what is called a pseudorandom oracle (PRO). This means it is indifferentiable from a random oracle. Indifferentiability [20, 10] is a technical term underlain by a formal definition. If a function is indifferentiable from a random oracle, it means that we can securely replace a random oracle with this function in most (not all) usages of the random oracle.

This can be viewed as saying that the Skein mode of operation has no structural weaknesses. It is evidence that attacks that differentiate it from a random oracle, such as the extension attack, are absent.

We should, however, add a word of warning and explanation. The result pertains to the mode of operation and not to the block cipher. The latter has been replaced by an ideal block cipher. The subtle point here is that there is no formal notion or assumption that we can state to capture "Threefish is, or approximates, an ideal block cipher". This is different from the other results discussed above. It is for example perfectly meaningful to say that Threefish is a PRP. We emphasize that the subtleties associated to indifferentiability are not peculiar to our results but rather endemic to the notion as a whole. They are, and will be, present for any hash function for which a proof of indifferentiability from a random oracle is supplied.

All this withstanding, the general consensus in the community is that indifferentiability buys you something. It is just difficult to *formally* say exactly what.

Support for HMAC Mode. Current hash functions are used in HMAC mode to obtain a MAC or a PRF. The widespread standardization and use of HMAC means that this represents a large and important domain of hash function usage. (HMAC is standardized via an IETF RFC [17], a NIST FIPS [16] and ANSI X9.71 [15]. It is in IEEE 802.11. It is implemented in SSL, SSH, IPsec and TLS amongst other places.) It is thus important that any new hash function continue to support usage in HMAC mode.

The issue this raises with regard to proofs is as follows. For hash functions that use Merkle-Damgård [11, 21] mode (in particular the MD and SHA families), HMAC mode is supported by proofs [2, 1] which arguably played an important role in the widespread and continuing adoption of HMAC. Current support for HMAC in this domain is represented by [1] who showed that HMAC with a Merkle-Damgård hash function is a secure PRF (and hence MAC) assuming that the compression function is itself a secure PRF. If Skein is to become a replacement for current hash functions, it is important that we provide a similar provable guarantee for its usage in HMAC mode. But since our underlying iteration method is not Merkle-Damgård, the previous proofs do not apply.

Our contribution in this regard is to supply new proofs. These show the analog of the above-mentioned result. Namely, if the compression function is a PRF then so is HMAC-Skein. This means that Skein has the same provable guarantees in HMAC mode as existing hash functions.

As a result, there are two different modes of operation in which Skein can provide a PRF or

MAC: HMAC mode and Skein's native keyed mode as discussed above. The latter is faster. However, the former needs to be supported for legacy reasons.

PRNG and Stream Cipher. The target security property for a stream cipher is that of [9, 24]. (The output on input a random seed should be computationally indistinguishable from random.) The target for the PRNG is that it should be forward-secure as defined by [7]. We prove both these properties under the assumption that Threefish is a PRP.

1.3 Plan

The design of Skein is modular. The base primitive is a tweakable block cipher. From this is built a tweakable compression function. The latter is iterated in a mode called UBI. Skein itself is built through invocations of UBI. We approach the proofs in a way that is similarly modular. Rather than proving properties of Skein directly (which would result in complex proofs), we establish properties of the lower-level primitives and propagate them upwards. Thus we will begin with the tweakable compression function then move to UBI and finally Skein.

2 Definitions

If M is a string then $|M|$ denotes its length. If $a_1, \ldots, a_n$ are strings then $a_1 \| \cdots \| a_n$ denotes their concatenation. If M is a string with length a positive multiple of l, then we let $|M|_l = |M|/l$ and let $M[i,l]$ denote the i-th l-bit block of M, meaning $M = M[1,l] \| \cdots \| M[m,l]$ where $m = |M|_l$. We fix throughout an integer b referred to as the block length and abbreviate $M[i,b]$ by $M[i]$. We use $\mathsf{IntToStr}_l(N)$, N is an integer, to denote the encoding of N mod 2^l as an l-bit string. If i, j are positive integers and Y is a string, where $i \le j \le |Y|$, then $Y[i \ldots j]$ denotes the substring of Y beginning at bit position i and ending at bit position j.

If $\mathcal{X}$ is a set, then $\mathcal{X}^+$ denotes the set of all sequences (tuples) over $\mathcal{X}$. The length of a sequence is defined as the sum of the lengths of its components.

If $f\colon\ D_1 \times \cdots \times D_n \to D$ is a function then $f_K\colon\ D_2 \times \cdots \times D_n \to D$ is the function defined by $f_K(x_2, \ldots, x_n) = f(K, x_2, \ldots, x_n)$ for all $K \in D_1$ and $(x_2, \ldots, x_n) \in D_2 \times \cdots \times D_n$.

We say that $f\colon\ D_0 \times \cdots \times D_n \to \{0,1\}^*$ is a variable output length (VOL) function if $D_0 \subseteq \mathbb{N}$ and $|f(N, x_1, \ldots, x_n)| = N$ for all $N \in D_0$ and all $(x_1, \ldots, x_n) \in D_1 \times \cdots \times D_n$.

Let $\mathsf{E}\colon\ \{0,1\}^b \times \{0,1\}^t \times \{0,1\}^b \to \{0,1\}^b$ be a map that takes a b-bit key K, a t-bit tweak T, and a b-bit input X to return a b-bit output $\mathsf{E}(\mathsf{K}, \mathsf{T}, X)$. We say that E is a tweakable block cipher [18] if the map $\mathsf{E}(\mathsf{K}, \mathsf{T}, \cdot)$ is a permutation on $\{0,1\}^b$ for all K, T.

A tweakable compression function is simply a map $\mathsf{TComp}\colon\ \{0,1\}^b \times \{0,1\}^t \times \{0,1\}^b \to \{0,1\}^b$. The first input is a key or the chaining variable, depending on usage. The second input is the tweak.

Throughout this paper we assume for simplicity that b and t are both multiples of 8, that $b \ge 256$, and that $t \ge 128$; the results herein are, however, generalizable beyond these restrictions.

Games. Our security definitions and proofs use code-based games [6], and so we recall some background from [6]. A game has an **Initialize** procedure, procedures to respond to adversary oracle queries, and (sometimes) a **Finalize** procedure. A game G is executed with an adversary A as follows. First, **Initialize** executes, and its outputs are the inputs to A. Then A executes,

its oracle queries being answered by the corresponding procedures of G. If there is no **Finalize** procedure, then when A terminates, its output is also called the output of the game. If there is a **Finalize** procedure, then when A terminates, its output becomes the input to the **Finalize** procedure. In this case the output of **Finalize** is called the output of the game. In either case, we let $G^A \Rightarrow y$ denote the event that this game output takes value y. The boolean flag bad is assumed initialized to false. Games G, H are identical-until-bad if their code differs only in statements that follow the setting of bad to true. We say that "G^A sets bad" to denote the event that game G, when executed with adversary A, sets bad to true. It is shown in [6] that if G, H are identical-until-bad and A is an adversary, then

$$\Pr\left[\, G^A \text{ sets } \mathsf{bad} \,\right] = \Pr\left[\, H^A \text{ sets } \mathsf{bad} \,\right] . \tag{1}$$

The fundamental lemma of game-playing [6] says that if G, H are identical-until-bad then

$$\Pr\left[\, G^A \Rightarrow y \,\right] - \Pr\left[\, H^A \Rightarrow y \,\right] \leq \Pr\left[\, G^A \text{ sets } \mathsf{bad} \,\right] .$$

Stateful Algorithms. We say that an algorithm is stateful if it maintains state across invocations. When run on some input the algorithm computes a response as a function of the input and its current state and also updates its state. We model "ideal" primitives via stateful algorithms. For example, an ideal tweakable block cipher with key and block length b and tweak length t is the stateful algorithm P that maintains state consisting of a pair E, E^{-1} of tables initially undefined everywhere. On input $c, (K, T, Z)$, where $c \in \{+1, -1\}$, $K, Z \in \{0,1\}^b$, and $T \in \{0,1\}^t$, P does the following:

If $c = +1$ then
  $x \leftarrow Z$
  If not $E(K, T, x)$ then
    $y \xleftarrow{\$} \{0,1\}^b \backslash \{\, E(K, T, x') \,:\, x' \in \{0,1\}^b \,\}$; $E(K, T, x) \leftarrow y$; $E^{-1}(K, T, y) \leftarrow x$
  Return $E(K, T, x)$
If $c = -1$ then
  $y \leftarrow Z$
  If not $E^{-1}(K, T, y)$ then
    $x \xleftarrow{\$} \{0,1\}^b \backslash \{\, E^{-1}(K, T, y') \,:\, y' \in \{0,1\}^b \,\}$; $E^{-1}(K, T, y) \leftarrow x$; $E(K, T, x) \leftarrow y$
  Return $E^{-1}(K, T, y)$

Similarly a RO with range D is the stateful algorithm R that maintains a table T, initially undefined everywhere, and responds to query x via

If not $T[x]$ then $T[x] \xleftarrow{\$} D$
Return $T[x]$

The Cascade Construction. The Cascade (or un-strengthened MD [11, 21]) transform takes a compression function $\mathsf{Comp}\colon\ \{0,1\}^b \times \mathcal{X} \to \{0,1\}^b$ and returns the function $\mathsf{Comp}^*\colon\ \{0,1\}^b \times \mathcal{X}^+ \to \{0,1\}^b$ defined in Figure 2.

3 Constructions

Threefish. *Threefish* [14] is a tweakable block cipher $\mathsf{E}\colon\ \{0,1\}^b \times \{0,1\}^t \times \{0,1\}^b \to \{0,1\}^b$ with $t = 128$ and $b \in \{256, 512, 1024\}$.

Function $\mathsf{Comp}^*(K, (X_1, \ldots, X_m))$
$h_0 \leftarrow K$
For $i = 1, \ldots, m$ do $h_i \leftarrow \mathsf{Comp}(h_{i-1}, X_i)$
Return h_m

Figure 2: The Cascade transform associates to compression function $\mathsf{Comp}\colon\ \{0,1\}^b \times \mathcal{X} \to \{0,1\}^b$ the function $\mathsf{Comp}^*\colon\ \{0,1\}^b \times \mathcal{X}^+ \to \{0,1\}^b$ defined above.

TComp. Our tweakable compression function TComp is obtained by running the tweakable block cipher E in Matyas-Meyer-Oseas [19] mode. Specifically, the associated tweakable compression function $\mathsf{TComp}\colon\ \{0,1\}^b \times \{0,1\}^t \times \{0,1\}^b \to \{0,1\}^b$ is defined by

$$\mathsf{TComp}(\mathsf{K}, \mathsf{T}, X) \ = \ \mathsf{E}(\mathsf{K}, \mathsf{T}, X) \oplus X \ . \tag{2}$$

This is an important feature that allows us to eventually prove that Skein as a PRF based on an assumption (and moreover, the standard one) about the underlying block cipher rather than on an assumption on the compression function, the latter being the case for the bulk of hash functions.

Unique Block Iteration (UBI). *Unique Block Iteration* (*UBI*) is a transform that takes a tweakable compression function $\mathsf{TComp} : \{0,1\}^b \times \{0,1\}^t \times \{0,1\}^b \to \{0,1\}^b$ and returns a function $\mathsf{f}\colon\ \{0,1\}^b \times \mathsf{MsgSp} \times \mathsf{TypeSp} \times \{0,1\}^7 \times [0, 2^{t-32}-1] \to \{0,1\}^b$. The message space MsgSp is the set of all strings of length at most $(2^{t-32}-1)2^3$ bits. The type space TypeSp is the set of all 6-bit strings. The fourth parameter will later be used to encode the level in a hash tree, and the fifth parameter will be used to encode the starting offset of a hash computation. The function f is defined in Figure 3 along with the subroutine Encode that it invokes. The Encode function also invokes a subroutine MkTw (make-tweak). This is a map $\mathsf{MkTw}\colon\ \{0,1\}\times\{0,1\}\times\{0,1\}^6\times\{0,1\}\times\{0,1\}^7\times[0,2^{t-32}-1] \to \{0,1\}^t$. Our only assumption on this map is that it is injective. An example instantiation achieving this is

$$\mathsf{MkTw}(\mathsf{fin}, \mathsf{fir}, \mathsf{type}, \mathsf{bitpad}, \mathsf{lvl}, \mathsf{L}) = \mathsf{fin}\|\mathsf{fir}\|\mathsf{type}\|\mathsf{bitpad}\|\mathsf{lvl}\|0^{16}\|\mathsf{IntToStr}_{t-32}(\mathsf{L}) \ .$$

SSkein. *SSkein* is a transform that takes a function $\mathsf{f}\colon\ \{0,1\}^b \times \mathsf{MsgSp} \times \mathsf{TypeSp} \times \{0,1\}^7 \times [0, 2^{t-32}-1] \to \{0,1\}^b$ and returns a function $\mathsf{SSkein}\colon\ \mathsf{OutLens}\times\{0,1\}^b\times((\mathsf{TypeSp}-\{\mathsf{TyCfg}, \mathsf{TyOut}\}) \times \mathsf{MsgSp})^* \to \{0,1\}^*$. The message space MsgSp is the set of all strings of length at most $(2^{t-32}-1)2^3$ bits. The type space TypeSp is the set of all 6-bit strings, and $\mathsf{TyCfg}, \mathsf{TyOut}$ are two distinct, fixed 6-bit strings. The first parameter to SSkein is the output length in bits, namely, $|\mathsf{SSkein}(\mathsf{N}, \cdot, \cdot)| = \mathsf{N}$. Here $\mathsf{OutLens}$ is defined as the set $\{\ 8i\ :\ i \in [0, 2^{64}-1]\ \}$, though in [14] it is defined as $\{\ i\ :\ i \in [0, 2^{64}]\ \}$. The function SSkein is defined in Figure 4 along with the subroutine Output that it invokes. The function SSkein also invokes a subroutine $\mathsf{SMkConfig}$ (make-config). This is a map $\mathsf{SMkConfig}\colon\ \mathsf{OutLens} \to \{\mathsf{TyCfg}\} \times \{0,1\}^{256}$. Our only assumption is that this map is injective. An example instantiating this is

$$\mathsf{SMkConfig}(\mathsf{N}) = (\mathsf{TyCfg}, 0^{64}\|\mathsf{IntToStr}_{64}(\mathsf{N}/8)\|0^8\|0^8\|0^8\|0^{104}) \ .$$

4 Encoding Lemmas

Definitions. Let $M^1 = (M^1_1, \ldots, M^1_{m^1}), M^2 = (M^2_1, \ldots, M^2_{m^2}) \in \mathcal{X}^+$ for some set $\mathcal{X}$. We say that

```
Function f(K, M, type, lvl, startl)
(M, T_1, ..., T_m) ←
    Encode(M, type, lvl, startl)
h_0 ← K
For i = 1, ..., m do
    h_i ← TComp(h_{i−1}, T_i, M[i])
Return h_m
```

```
Function Encode(M, type, lvl, startl)
bitpad ← 0
If |M| mod 8 ≠ 0 then
    bitpad ← 1 ; s ← 7 − (|M| mod 8) ; M ← M‖1‖0^s
L ← |M|/8
If |M| = 0 then M ← 0^b
Else
    If |M| mod b = 0 then s ← 0
    Else s ← b − (|M| mod b)
    M ← M‖0^s
m ← |M|_b ; l_0 ← startl
If m = 1 then
    T_m ← MkTw(1, 1, type, bitpad, lvl, L + startl mod 2^{t−32})
Else
    l_1 ← l_0 + (b/8) ; T_1 ← MkTw(0, 1, type, 0, lvl, l_1 mod 2^{t−32})
    For i = 2, ..., m − 1 do
        l_i ← l_{i−1} + (b/8)
        T_i ← MkTw(0, 0, type, 0, lvl, l_i mod 2^{t−32})
    T_m ← MkTw(1, 0, type, bitpad, lvl, L + startl mod 2^{t−32})
Return (M, T_1, ..., T_m)
```

Figure 3: The UBI transform associates to tweakable compression function $\mathsf{TComp}\colon\ \{0,1\}^b \times \{0,1\}^t \times \{0,1\}^b \to \{0,1\}^b$ the function $\mathsf{f}\colon\ \{0,1\}^b \times \mathsf{MsgSp} \times \mathsf{TypeSp} \times \{0,1\}^7 \times [0, 2^{t-32}-1] \to \{0,1\}^b$ defined above.

```
Function SSkein(N, K, ((type_1, M_1), ..., (type_r, M_r)))
 //   r ≥ 0 ; TyCfg, TyOut ∉ {type_1, ..., type_r} ; |K| = b
h_{−1} ← K
(type_0, M_0) ← SMkConfig(N)
For i = 0, ..., r do
    h_i ← f(h_{i−1}, M_i, type_i, 0^7, 0)
h ← Output(h_r, N)
Return h
```

```
Function Output(h, N)
// |h| = b ; N ∈ OutLens
y ← ε
d ← ⌈N/b⌉
For i = 1, ..., d do
    y ← y‖f(h, IntToStr_64(i), TyOut, 0^7, 0)
h ← y[1 ... N]
Return h
```

Figure 4: The SSkein transform associates to the function $\mathsf{f}\colon\ \{0,1\}^b \times \mathsf{MsgSp} \times \mathsf{TypeSp} \times \{0,1\}^7 \times [0, 2^{t-32}-1] \to \{0,1\}^b$ the function $\mathsf{SSkein}\colon\ \mathsf{OutLens} \times \{0,1\}^b \times ((\mathsf{TypeSp} - \{\mathsf{TyCfg}, \mathsf{TyOut}\}) \times \mathsf{MsgSp})^* \to \{0,1\}^*$ defined above.

M^1 is a prefix of M^2, written $M^1 \leq_P M^2$, if $m^1 \leq m^2$ and $M^1[i] = M^2[i]$ for all $i \in \{1, \ldots, m^1\}$. If M^1 is not a prefix of M^2, we write $M^1 \not\leq_P M^2$.

Let $M^1 = (M^1_1, \ldots, M^1_{m_1}), M^2 = (M^2_1, \ldots, M^2_{m_2}) \in \mathcal{X}^+$ for some set $\mathcal{X}$. We say that M^1 is a suffix of M^2, written $M^1 \leq_S M^2$, if $m_1 \leq m_2$ and $M^1[i] = M^2[i + m_2 - m_1]$ for all $i \in \{1, \ldots, m_1\}$. If M^1 is not a suffix of M^2, we write $M^1 \not\leq_S M^2$.

PROPERTIES OF Encode. We say that 4-tuple $(\mathsf{M}, \mathsf{type}, \mathsf{lvl}, \mathsf{startl})$ is start-respecting if startl is a multiple of $b/8$, $\mathsf{startl} = 0$ whenever $\mathsf{M} = \varepsilon$, and $|\mathsf{M}| + \mathsf{startl} \cdot 2^3 \leq (2^{t-32} - 1)2^3$. Without the start-respecting restriction, it may be possible for Encode to output the same values on input two distinct tuples $(\mathsf{M}^1, \mathsf{type}^1, \mathsf{lvl}^1, \mathsf{startl}^1)$ and $(\mathsf{M}^2, \mathsf{type}^2, \mathsf{lvl}^2, \mathsf{startl}^2)$.

We now state the following lemmas:

Lemma 4.1 Let $(\mathsf{M}^1, \mathsf{type}^1, \mathsf{lvl}^1, \mathsf{startl}^1)$ and $(\mathsf{M}^2, \mathsf{type}^2, \mathsf{lvl}^2, \mathsf{startl}^2)$ be distinct start-respecting 4-tuples and let

$(M_1, T^1_1, \ldots, T^1_{m_1}) \leftarrow \mathsf{Encode}(\mathsf{M}^1, \mathsf{type}^1, \mathsf{lvl}^1, \mathsf{startl}^1)$
$(M_2, T^2_1, \ldots, T^2_{m_2}) \leftarrow \mathsf{Encode}(\mathsf{M}^2, \mathsf{type}^2, \mathsf{lvl}^2, \mathsf{startl}^2)$
For $i = 1, \ldots, m_1$ do $M^*_i \leftarrow (T^1_i, M_1[i])$
For $i = 1, \ldots, m_2$ do $N^*_i \leftarrow (T^2_i, M_2[i])$
$M^* \leftarrow (M^*_1, \ldots, M^*_{m_1})$
$N^* \leftarrow (N^*_1, \ldots, N^*_{m_2})$

Then $M^* \not\leq_P N^*$.

Proof: If $m_1 < m_2$, then $M^* \not\leq_P N^*$ because $T^1_{m_1}$ has the final bit set but $T^2_{m_1}$ does not. If $\mathsf{type}^1 \neq \mathsf{type}^2$ or $\mathsf{lvl}^1 \neq \mathsf{lvl}^2$, then $M^* \not\leq_P N^*$ because the tweaks encode these values.

Suppose now that $m_1 = m_2$, $\mathsf{type}^1 = \mathsf{type}^2$, and $\mathsf{lvl}^1 = \mathsf{lvl}^2$. If $\mathsf{M}^1 \neq \mathsf{M}^2$ and $\mathsf{startl}^1 = \mathsf{startl}^2$, then $M^* \not\leq_P N^*$ since either the final lengths or bit-padding fields will encode a difference or $M_1[i] \neq M_2[i]$ for some index $i \in \{1, \ldots, m_1\}$. If $\mathsf{M}^1 = \mathsf{M}^2$ and $\mathsf{startl}^1 \neq \mathsf{startl}^2$, then $M^* \not\leq_P N^*$ since the final lengths will encode a difference.

If $\mathsf{M}^1 \neq \mathsf{M}^2$ and $\mathsf{startl}^1 \neq \mathsf{startl}^2$ we must be more careful. If $m_1 = m_2 > 1$ then $T^1_1 \neq T^2_1$ since the first tweak will encode a different length and hence $M^* \not\leq_P N^*$. So suppose $m_1 = m_2 = 1$. If $\mathsf{M}^1 \neq \varepsilon$ and $\mathsf{M}^2 \neq \varepsilon$ then $T^1_1 \neq T^2_1$ since the first tweak will encode a different length (under the assumption that the starting length values are all multiples of $b/8$) and hence $M^* \not\leq_P N^*$. If without loss of generality $\mathsf{M}^1 = \varepsilon$, then $\mathsf{startl}^1 = 0$ by assumption and $T^1_1 \neq T^2_1$ since the first (and only) tweak will encode a different length and hence $M^* \not\leq_P N^*$. ∎

Lemma 4.2 Let $(\mathsf{M}^1, \mathsf{type}^1, \mathsf{lvl}^1, \mathsf{startl}^1)$ and $(\mathsf{M}^2, \mathsf{type}^2, \mathsf{lvl}^2, \mathsf{startl}^2)$ be distinct start-respecting 4-tuples and let

$(M_1, T^1_1, \ldots, T^1_{m_1}) \leftarrow \mathsf{Encode}(\mathsf{M}^1, \mathsf{type}^1, \mathsf{lvl}^1, \mathsf{startl}^1)$
$(M_2, T^2_1, \ldots, T^2_{m_2}) \leftarrow \mathsf{Encode}(\mathsf{M}^2, \mathsf{type}^2, \mathsf{lvl}^2, \mathsf{startl}^2)$
For $i = 1, \ldots, m_1$ do $M^*_i \leftarrow (T^1_i, M_1[i])$
For $i = 1, \ldots, m_2$ do $N^*_i \leftarrow (T^2_i, M_2[i])$
$M^* \leftarrow (M^*_1, \ldots, M^*_{m_1})$
$N^* \leftarrow (N^*_1, \ldots, N^*_{m_2})$

Then $M^* \not\leq_S N^*$.

Proof: The proof of Lemma 4.2 is identical to the proof of Lemma 4.1, except with the first bit playing the role of the final bit in the case where $m_1 < m_2$ (because T_1^1 has the first bit set but $T_{1+m_2-m_1}^2$ does not). ∎

5 Collision Resistance

5.1 Definitions

Let $f\colon\ D_1 \times D_2 \times \cdots \times D_d \to \{0,1\}^n$ be a function. A CR-adversary A is a randomized algorithm that does not take any input and that returns a pair of distinct tuples $(x_1, x_2, \ldots, x_d)$, $(x'_1, x'_2, \ldots, x'_d) \in D_1 \times D_2 \times \cdots \times D_d$. Let $\mathbf{Adv}^{\mathrm{cr}}_f(A)$ denote the probability, over the random coins for A, that $f(x_1, x_2, \ldots, x_d) = f(x'_1, x'_2, \ldots, x'_d)$.

5.2 Collision Resistance for TComp

Collision resistance of Skein is proved based on the collision resistance of TComp. This proof is in the standard (i.e., not RO) model. In the standard model we know of no proof of CR of TComp based on a standard assumption on the tweakable block cipher. But we note that TComp can be proved collision resistant if the tweakable block cipher is ideal (extending [8] to the tweakable setting).

5.3 Collision Resistance for UBI

We show that UBI preserves collision-resistance, meaning that if TComp is CR then f is also CR against startl-restricted adversaries. In saying TComp is CR we regard it simply as a 3-argument function, meaning a collision is any pair of distinct triples on which TComp has the same output. Similarly, a startl-restricted collision against f is any pair of distinct 5-tuples $((\mathsf{K}^1, \mathsf{M}^1, \mathsf{type}^1, \mathsf{lvl}^1, \mathsf{startl}^1), (\mathsf{K}^2, \mathsf{M}^2, \mathsf{type}^2, \mathsf{lvl}^2, \mathsf{startl}^2))$ on which f has the same output, under the restriction that $\mathsf{startl}^1 = \mathsf{startl}^2$.

Theorem 5.1 *Let* $\mathsf{TComp}\colon\ \{0,1\}^b \times \{0,1\}^t \times \{0,1\}^b$ *be a tweakable compression function and let* $\mathsf{f}\colon\ \{0,1\}^b \times \mathsf{MsgSp} \times \mathsf{TypeSp} \times \{0,1\}^7 \times [0, 2^{t-32} - 1] \to \{0,1\}^b$ *be obtained by applying the UBI transform to* TComp *as per Figure 3. Let* A *be a* startl*-restricted CR-adversary against* f*. Let* B *be the CR-adversary against* TComp *as defined in Figure 5. Then*

$$\mathbf{Adv}^{\mathrm{cr}}_{\mathsf{f}}(A) \ \leq\ \mathbf{Adv}^{\mathrm{cr}}_{\mathsf{TComp}}(B)\ .$$

Further, the running time of B *is that of* A *plus the time for* $m^1 + m^2$ *applications of* TComp*, where* m^1 *and* m^2 *are as in Figure 5.*

Proof: Suppose $(\mathsf{K}^1, \mathsf{M}^1, \mathsf{type}^1, \mathsf{lvl}^1, \mathsf{startl}^1) \neq (\mathsf{K}^2, \mathsf{M}^2, \mathsf{type}^2, \mathsf{lvl}^2, \mathsf{startl}^2)$, $\mathsf{startl}^1 = \mathsf{startl}^2$, and $\mathsf{f}(\mathsf{K}^1, \mathsf{M}^1, \mathsf{type}^1, \mathsf{lvl}^1, \mathsf{startl}^1) = \mathsf{f}(\mathsf{K}^2, \mathsf{M}^2, \mathsf{type}^2, \mathsf{lvl}^2, \mathsf{startl}^2)$.

Adversary B
$((\mathsf{K}^1, \mathsf{M}^1, \mathsf{type}^1, \mathsf{lvl}^1, \mathsf{startl}^1), (\mathsf{K}^2, \mathsf{M}^2, \mathsf{type}^2, \mathsf{lvl}^2, \mathsf{startl}^2)) \xleftarrow{\$} A$
For $j = 1$ to 2 do
  $(\overline{\mathsf{M}}^j, \mathsf{T}^j_1, \ldots, \mathsf{T}^j_{m^j}) \leftarrow \mathsf{Encode}(\mathsf{M}^j, \mathsf{type}^j, \mathsf{lvl}^j, \mathsf{startl}^j)$
  $\mathsf{h}^j_0 \leftarrow \mathsf{K}^j$
  For $i = 1, \ldots, m^j$ do $\mathsf{h}^j_i \leftarrow \mathsf{TComp}(\mathsf{h}^j_{i-1}, \mathsf{T}^j_i, \overline{\mathsf{M}}^j[i])$
For $i = 0$ to $\min(m^1, m^2) - 1$ do
  If $(\mathsf{h}^1_{m^1-i-1}, \mathsf{T}^1_{m^1-i}, \overline{\mathsf{M}}^1[m^1 - i]) \neq (\mathsf{h}^2_{m^2-i-1}, \mathsf{T}^2_{m^2-i}, \overline{\mathsf{M}}^2[m^2 - i])$ then
    return $(\mathsf{h}^1_{m^1-i-1}, \mathsf{T}^1_{m^1-i}, \overline{\mathsf{M}}^1[m^1 - i]), (\mathsf{h}^2_{m^2-i-1}, \mathsf{T}^2_{m^2-i}, \overline{\mathsf{M}}^2[m^2 - i])$

Figure 5: Adversary B referred to in Theorem 5.1. The function Encode is defined as in Figure 3.

Consider the final for loop of B. Let $i \in \{0, \ldots, \min(m^1, m^2) - 1\}$ be the least value such that

$$(\mathsf{h}^1_{m^1-i-1}, \mathsf{T}^1_{m^1-i}, \overline{\mathsf{M}}^1[m^1 - i]) \neq (\mathsf{h}^2_{m^2-i-1}, \mathsf{T}^2_{m^2-i}, \overline{\mathsf{M}}^2[m^2 - i])$$

Such an index exists for reasons we explain below. By the choice of i and since $\mathsf{f}(\mathsf{K}^1, \mathsf{M}^1, \mathsf{type}^1, \mathsf{lvl}^1, \mathsf{startl}^1) = \mathsf{f}(\mathsf{K}^2, \mathsf{M}^2, \mathsf{type}^2, \mathsf{lvl}^2, \mathsf{startl}^2)$, it follows that $\mathsf{h}^1_{m^1-i} = \mathsf{h}^2_{m^2-i}$. Therefore $\mathsf{TComp}(\mathsf{h}^1_{m^1-i-1}, \mathsf{T}^1_{m^1-i}, \overline{\mathsf{M}}^1[m^1 - i]) = \mathsf{TComp}(\mathsf{h}^2_{m^2-i-1}, \mathsf{T}^2_{m^2-i}, \overline{\mathsf{M}}^2[m^2 - i])$ and B outputs a collision.

To justify the existence of such an index i, we consider several cases. Clearly if $\mathsf{type}^1 \neq \mathsf{type}^2$ or $\mathsf{lvl}^1 \neq \mathsf{lvl}^2$, then such an index i exists because of the injectivity of the function MkTw which Encode invokes. So assume $\mathsf{type}^1 = \mathsf{type}^2$ and $\mathsf{lvl}^1 = \mathsf{lvl}^2$. By the startl-restriction assumption on A, $\mathsf{startl}^1 = \mathsf{startl}^2$. Consider the case where $|\mathsf{M}^1| = |\mathsf{M}^2|$. By the definition of a collision for A, $\mathsf{K}^1 \neq \mathsf{K}^2$ or $\mathsf{M}^1 \neq \mathsf{M}^2$ or both and thus the chaining values are different for index $i = m^1 - 1$ (if $\mathsf{K}^1 \neq \mathsf{K}^2$) or there exists an index i such that $\overline{\mathsf{M}}^1[i] \neq \overline{\mathsf{M}}^2[i]$ (if $\mathsf{M}^1 \neq \mathsf{M}^2$). Consider now the case where $|\mathsf{M}^1| \neq |\mathsf{M}^2|$. Then $(\overline{\mathsf{M}}^1[m^1], \mathsf{T}^1_{m^1}) \neq (\overline{\mathsf{M}}^2[m^2], \mathsf{T}^2_{m^2})$ because of Encode. Here we must consider multiple cases, corresponding to whether M^1 or M^2 is bit-padded. If neither are bit-padded, then the lengths encoded in $\mathsf{T}^1_{m^1}$ and $\mathsf{T}^2_{m^2}$ will be different. If both are bit-padded, then $\overline{\mathsf{M}}^1[m^1] \neq \overline{\mathsf{M}}^2[m^2]$ or the final encoded lengths are different. If only one is bit-padded, then the bit padding flag will only be set for one of the tweak values. ∎

5.4 Collision Resistance for SSkein

We now show that SSkeinpreserves collision-resistance, meaning that if f is CR against startl-restricted adversaries as defined in Section 5.3, then $\mathsf{H}(\cdot,\cdot) = \mathsf{SSkein}(\mathsf{N}, \cdot, \cdot)$ is also CR where N is any output length at least b. (By definition, an N-bit output does not collide with an N'-bit output when $\mathsf{N} \neq \mathsf{N}'$.)

Theorem 5.2 *Let* $\mathsf{f}\colon\ \{0,1\}^b \times \mathsf{MsgSp} \times \mathsf{TypeSp} \times \{0,1\}^7 \times [0, 2^{t-32} - 1] \to \{0,1\}^b$ *be the UBI compression function and let* $\mathsf{SSkein}\colon\ \mathsf{OutLens} \times \{0,1\}^b \times ((\mathsf{TypeSp} - \{\mathsf{TyCfg}, \mathsf{TyOut}\}) \times \mathsf{MsgSp})^* \to \{0,1\}^*$ *be the VOL function defined from* f *as per Figure 4. Let* $\mathsf{H}(\cdot,\cdot) = \mathsf{FSkein}(\mathsf{N}, \cdot, \cdot)$ *for some fixed* $\mathsf{N} \geq b$. *Let* A *be a CR-adversary against* H. *Let* B *be the* startl*-restricted CR-adversary against* f *as defined in Figure 6. Then*

$$\mathbf{Adv}^{\mathrm{cr}}_{\mathsf{H}}(A) \ \leq\ \mathbf{Adv}^{\mathrm{cr}}_{\mathsf{f}}(B)\,.$$

Adversary B
$(\mathsf{K}^1, ((\mathsf{type}_1^1, \mathsf{M}_1^1), \ldots, (\mathsf{type}_{r^1}^1, \mathsf{M}_{r^1}^1))), (\mathsf{K}^2, ((\mathsf{type}_1^2, \mathsf{M}_1^2), \ldots, (\mathsf{type}_{r^2}^2, \mathsf{M}_{r^2}^2))) \xleftarrow{\$} A$
For $j = 1$ to 2 do
    $\mathsf{h}_{-1}^j \leftarrow \mathsf{K}^j$
    $(\mathsf{type}_0^j, \mathsf{M}_0^j) \leftarrow \mathsf{SMkConfig}(\mathsf{N})$
    $\mathsf{type}_{r+1}^j \leftarrow \mathsf{TyOut}$; $\mathsf{M}_{r+1}^j \leftarrow \mathsf{IntToStr}_{64}(1)$
    For $i = 0, \ldots, r^j + 1$ do
        $\mathsf{h}_i^j \leftarrow \mathsf{f}(\mathsf{h}_{i-1}^j, \mathsf{M}_i^j, \mathsf{type}_i^j, 0^7, 0)$
For $i = -1$ to $\min(r^1, r^2)$
    If $(\mathsf{h}_{r^1-i-1}^1, \mathsf{M}_{r^1-i}^1, \mathsf{type}_{r^1-i}^1) \neq (\mathsf{h}_{r^2-i-1}^2, \mathsf{M}_{r^2-i}^2, \mathsf{type}_{r^2-i}^2)$ then
(*)   return $(\mathsf{h}_{r^1-i-1}^1, \mathsf{M}_{r^1-i}^1, \mathsf{type}_{r^1-i}^1, 0^7, 0), (\mathsf{h}_{r^2-i-1}^2, \mathsf{M}_{r^2-i}^2, \mathsf{type}_{r^2-i}^2, 0^7, 0)$

Figure 6: Adversary B referred to in Theorem 5.2.

Further, the running time of B is that of A plus the time for $r^1 + r^2 + 4$ applications of f, where r^1 and r^2 are as in Figure 6.

Proof: Suppose A returns a collision against H. Then $\mathsf{SSkein}(\mathsf{N}, \mathsf{K}^1, ((\mathsf{type}_1^1, \mathsf{M}_1^1), \ldots, (\mathsf{type}_{r^1}^1, \mathsf{M}_{r^1}^1))) = \mathsf{SSkein}(\mathsf{N}, \mathsf{K}^2, ((\mathsf{type}_1^2, \mathsf{M}_1^2), \ldots, (\mathsf{type}_{r^2}^2, \mathsf{M}_{r^2}^2)))$. Consequently $\mathsf{h}_{r^1+1}^1 = \mathsf{h}_{r^2+1}^2$.

Consider the final for loop in B. The loop iterates i from -1 to $\min(r^1, r^2)$. A loop invariant is that, at the beginning of each loop and prior to the loop's exit, $\mathsf{h}_{r^1-i}^1 = \mathsf{h}_{r^2-i}^2$. This follows since the conditional in the loop checks for $\mathsf{h}_{r^1-i-1}^1 = \mathsf{h}_{r^2-i-1}^2$ and returns if not equal.

If the loop exits early, then the precondition for this return, combined with the loop invariant, ensures that the returned value is a collision for f. It remains to show that the loop will exit early if A returns a collision.

First, if $r^1 \neq r^2$, then $\mathsf{type}_{r^1-\min(r^1,r^2)}^1 \neq \mathsf{type}_{r^2-\min(r^1,r^2)}^2$ since only type_0^1 and type_0^2 can be the TyCfg type. Hence, if a previous iteration of the loop has not already returned a collision, then the iteration with $i = \min(r^1, r^2)$ will return a collision.

Suppose henceforth that $r^1 = r^2$. If $\mathsf{K}^1 \neq \mathsf{K}^2$, then the loop will return during its last iteration unless a previous iteration of the loop already returned. If there exists an index $r' \in \{1, \ldots, r^1\}$ such that $\mathsf{type}_{r'}^1 \neq \mathsf{type}_{r'}^2$, then the loop will execute return at index $i = r^1 - r'$ unless a previous iteration of the loop already returned.

Henceforth suppose also that $\mathsf{K}^1 = \mathsf{K}^2$ and $\mathsf{type}_{r'}^1 = \mathsf{type}_{r'}^2$ for all $r' \in \{1, \ldots, r^1\}$. Since A returned a collision, there must exist an index $r' \in \{1, \ldots, r^1\}$ such that $\mathsf{M}_{r'}^1 \neq \mathsf{M}_{r'}^2$. The loop will therefore return early at index $i = r^1 - r'$ unless a previous iteration of the loop already returned. ∎

Game Real_f	Game Rand_D
Procedure **Initialize** $K \xleftarrow{\$} D_1$	Procedure **Initialize**
Procedure $\mathbf{Fn}(x_2, \ldots, x_n)$ Return $f(K, x_2, \ldots, x_n)$	Procedure $\mathbf{Fn}(x_2, \ldots, x_n)$ Return $y \xleftarrow{\$} D$

Figure 7: Games Real_f and Rand_D for the definition of a pseudorandom function (PRF).

Game Real_f	Game Perm_D
Procedure **Initialize** $K \xleftarrow{\$} D_1$	Procedure **Initialize** $R \leftarrow \emptyset$
Procedure $\mathbf{Fn}(x)$ Return $f(K, x)$	Procedure $\mathbf{Fn}(x)$ $y \xleftarrow{\$} D \backslash R$; $R \leftarrow R \cup \{y\}$ Return y

Figure 8: Games Real_f and Perm_D for the definition of a pseudorandom permutation (PRP).

6 Pseudorandomness

6.1 Definitions and Tools

A prf-adversary against $f\colon\ D_1 \times \cdots \times D_n \to D$ has access to one oracle and outputs a bit. Each oracle query must have the form $x_2, \ldots, x_n$ where $(x_2, \ldots, x_n) \in D_2 \times \cdots \times D_n$ and the queries must all be distinct. Its advantage is

$$\mathbf{Adv}^{\mathrm{prf}}_f(A) \;=\; \Pr\left[\,\mathsf{Real}^A_f \Rightarrow 1\,\right] - \Pr\left[\,\mathsf{Rand}^A_D \Rightarrow 1\,\right]$$

where games Real_f and Rand_D are shown in Figure 7.

A prp-adversary against $f\colon\ D_1 \times D \to D$ has access to one oracle and outputs a bit. Each oracle query must be a member of D and the queries must be all distinct. Its advantage is

$$\mathbf{Adv}^{\mathrm{prp}}_f(A) \;=\; \Pr\left[\,\mathsf{Real}^A_f \Rightarrow 1\,\right] - \Pr\left[\,\mathsf{Perm}^A_D \Rightarrow 1\,\right]$$

where games Real_f and Perm_D are shown in Figure 8.

A vol-prf-adversary against a VOL function $F\colon\ D_0 \times D_1 \times \cdots \times D_n \to \{0,1\}^*$ has access to one oracle and outputs a bit. Each oracle query must have the form $N, x_2, \ldots, x_n$ where $N \in D_0 \subseteq \mathbb{Z}_+ \cup \{0\}$ and $(x_2, \ldots, x_n) \in D_2 \times \cdots \times D_n$ and the queries must all be distinct. Its advantage is

$$\mathbf{Adv}^{\text{vol-prf}}_F(A) \;=\; \Pr\left[\,\mathsf{Real}^A_F \Rightarrow 1\,\right] - \Pr\left[\,\mathsf{Rand}^A \Rightarrow 1\,\right]$$

where games Real_F and Rand are shown in Figure 9.

Game Real_F	Game Rand
Procedure **Initialize** $K \xleftarrow{\$} D_1$	Procedure **Initialize**
Procedure $\mathbf{Fn}(N, x_2, \ldots, x_n)$ Return $F(N, K, x_2, \ldots, x_n)$	Procedure $\mathbf{Fn}(N, x_2, \ldots, x_n)$ Return $y \xleftarrow{\$} \{0,1\}^N$

Figure 9: Games Real_F and Rand for the definition of pseudorandomness for a VOL function (VOL-PRF).

Pseudorandomness and the Cascade Construction. In our proofs we will use a result from [3] on the Cascade transform, which takes a compression function $\mathsf{Comp}\colon\ \{0,1\}^b \times \mathcal{X} \to \{0,1\}^b$ and returns the function $\mathsf{Comp}^*\colon\ \{0,1\}^b \times \mathcal{X}^+ \to \{0,1\}^b$ defined in Figure 2. An extension attack can be applied to show that Comp^* is not a PRF even if Comp is a PRF. However, Comp^* is a PRF as long as no message to which it is applied is a prefix of another. We will use this fact from [3] to prove Theorem 6.3 and Theorem 6.4.

A prf-adversary B against Comp^* is said to be prefix-free if $M^1 \not\leq_P M^2$ for any two different oracle queries M^1, M^2 of B.

Lemma 6.1 (From [3]) Let $\mathsf{Comp}\colon\ \{0,1\}^b \times \mathcal{X} \to \{0,1\}^b$ be a compression function and let Comp^* be obtained by applying the Cascade transform to Comp. Let B^* be a prefix-free prf-adversary against Comp^* making q oracle queries, each of length at most s elements from $\mathcal{X}$. Then there is a prf-adversary B against Comp such that

$$\mathbf{Adv}^{\mathrm{prf}}_{\mathsf{Comp}^*}(B^*) \ \leq\ qs \cdot \mathbf{Adv}^{\mathrm{prf}}_{\mathsf{Comp}}(B)\ .$$

Furthermore B makes qs oracle queries and its running time is that of B^* plus $O(q(\log q)(s+b+x))$, where x is the longest string in $\mathcal{X}$.

6.2 Pseudorandomness of TComp

Proposition 6.2 *Let* $\mathsf{E}\colon\ \{0,1\}^b \times \{0,1\}^t \times \{0,1\}^b \to \{0,1\}^b$ *be a tweakable block cipher and let* $\mathsf{TComp}\colon\ \{0,1\}^b \times \{0,1\}^t \times \{0,1\}^b \to \{0,1\}^b$ *be defined by*

$$\mathsf{TComp}(\mathsf{K}, \mathsf{T}, X) \ =\ \mathsf{E}(\mathsf{K}, \mathsf{T}, X) \oplus X\ .$$

Let A_{TComp} *be a prf-adversary against* TComp *that makes* q *oracle queries. Then there is a prp-adversary* A_{E} *against* E *such that*

$$\mathbf{Adv}^{\mathrm{prf}}_{\mathsf{TComp}}(A_{\mathsf{TComp}}) \ \leq\ \mathbf{Adv}^{\mathrm{prp}}_{\mathsf{E}}(A_{\mathsf{E}}) + \frac{q^2}{2^{b+1}}\ .$$

Furthermore, A_{E} *makes* q *oracle queries and its running time is that of* A_{TComp} *plus* $O(q(b+t))$.

Proof: A_{E} runs A_{TComp}. When the latter makes an oracle query (T, X), adversary A_{E} queries (T, X) to its own oracle and gets back a value it denotes Y and then returns $Y \oplus X$ to A_{TComp}. It outputs whatever A_{TComp} outputs. The analysis is standard and uses (an extension to the tweakable settings of) the PRP/PRF Switching Lemma [6]. ▌

6.3 Pseudorandomness for UBI

We say that a PRF adversary against UBI is start-respecting if all its oracle queries are start-respecting. We claim that UBI is PRF-preserving against start-respecting adversaries. This means that if TComp is a PRF then so is f. This is implied by the following.

Theorem 6.3 *Let* $\mathsf{TComp}\colon\ \{0,1\}^b \times \{0,1\}^t \times \{0,1\}^b \to \{0,1\}^b$ *be a tweakable compression function and let* $\mathsf{f}\colon\ \{0,1\}^b \times \mathsf{MsgSp} \times \mathsf{TypeSp} \times \{0,1\}^7 \times [0, 2^{t-32}-1] \to \{0,1\}^b$ *be obtained by applying the UBI transform to* TComp *as per Figure 3. Let A be a start-respecting prf-adversary against* f *that makes q oracle queries, the message field of each query of length at most L. Then there is a prf-adversary B against* TComp *such that*

$$\mathbf{Adv}^{\mathrm{prf}}_{\mathsf{f}}(A) \ \le\ sq \cdot \mathbf{Adv}^{\mathrm{prf}}_{\mathsf{TComp}}(B)$$

where $s = \lceil L/b \rceil$. Furthermore, the running time of B is that of A plus $O(q(\log q)(s+b+t))$.

Proof: Let $\mathcal{X} = \{0,1\}^t \times \{0,1\}^b$ and define $\mathsf{Comp}\colon \{0,1\}^b \times \mathcal{X} \to \{0,1\}^b$ by

$$\mathsf{Comp}(h,(T,X)) \ =\ \mathsf{TComp}(h,T,X)$$

for all $h, X \in \{0,1\}^b$ and $T \in \{0,1\}^t$.

Now define B^* as follows against Comp^*, the Cascade transform of Comp (Figure 2). B^* runs A. When the latter makes an oracle query $(\mathsf{M}, \mathsf{type}, \mathsf{lvl}, \mathsf{startl})$, adversary B^* computes $M^* \in \mathcal{X}^+$ as follows:

$(M, T_1, \ldots, T_m) \leftarrow \mathsf{Encode}(\mathsf{M}, \mathsf{type}, \mathsf{lvl}, \mathsf{startl})$
For $i = 1, \ldots, m$ do $M^*_i \leftarrow (T_i, M[i])$
$M^* \leftarrow (M^*_1, \ldots, M^*_m)$

B^* invokes its own oracle on M^* to get back a response Y, and returns Y to A as the response to its query M. When A halts with output a bit, B^* halts with the same output.

This construction guarantees that

$$\mathbf{Adv}^{\mathrm{prf}}_{\mathsf{Comp}^*}(B^*) \ =\ \mathbf{Adv}^{\mathrm{prf}}_{\mathsf{f}}(A)\ . \tag{3}$$

On the other hand, Lemma 4.1 guarantees that B^* is prefix-free, regardless of the queries made by A, so we can apply Lemma 6.1. Let D be the resulting prf-adversary against Comp. We now transform D into a prf-adversary B against TComp such that

$$\mathbf{Adv}^{\mathrm{prf}}_{\mathsf{TComp}}(B) \ =\ \mathbf{Adv}^{\mathrm{prf}}_{\mathsf{Comp}}(D)\ . \tag{4}$$

The theorem follows from Equation (5), Equation (6), and Lemma 6.1. It remains to describe B.

B runs D. When the latter makes an oracle query $W \in \mathcal{X}$, adversary B parses it as $W = (T, X)$ with $|T| = t$ and $|X| = b$. It calls its own oracle on T, X and returns the answer to D. ∎

6.4 Pseudorandomness for SSkein

It is possible to prove the pseudorandomness of SSkein with a reduction from the pseudorandomness of UBI. For a tighter bound, we state and prove a reduction from the pseudorandomness of SSkein's underlying compression function.

Theorem 6.4 *Let* $\mathsf{TComp}\colon\ \{0,1\}^b \times \{0,1\}^t \times \{0,1\}^b \to \{0,1\}^b$ *be a tweakable compression function. Let* $\mathsf{f}\colon\ \{0,1\}^b \times \mathsf{MsgSp} \times \mathsf{TypeSp} \times \{0,1\}^7 \times [0, 2^{t-32}-1] \to \{0,1\}^b$ *be obtained by applying the UBI transform to* TComp *per Figure 3. Let* $\mathsf{SSkein}\colon\ \mathsf{OutLens} \times \{0,1\}^b \times ((\mathsf{TypeSp} - \{\mathsf{TyCfg}, \mathsf{TyOut}\}) \times \mathsf{MsgSp})^* \to \{0,1\}^*$ *be the VOL function defined from* f *as per Figure 4. Let* A_{SSkein} *be a vol-prf-adversary against* SSkein *that makes at most* q *oracle queries, where the* i*-th query consists of an integer (the output length value) that is at most* N_i *and a list of at most* l *pairs, with each message in a pair at most* l' *bits long. Then there is a prf-adversary* A_{TComp} *such that*

$$\mathbf{Adv}^{\text{vol-prf}}_{\mathsf{SSkein}}(A_{\mathsf{SSkein}}) \le q'\Big(l \cdot \lceil l'/b \rceil + 3\Big) \cdot \mathbf{Adv}^{\text{prf}}_{\mathsf{TComp}}(A_{\mathsf{TComp}})\ ,$$

where $q' = \sum_{i=1}^{q} \lceil \mathsf{N}_i/b \rceil$. *Furthermore,* A_{TComp} *makes at most* $q'(l \cdot \lceil l'/b \rceil + 3)$ *oracle queries and its running time is that of* A_{SSkein} *plus* $O(q'(\log q')(l \cdot \lceil l'/b \rceil + b + t))$.

Proof: Let $\mathcal{X} = \{0,1\}^t \times \{0,1\}^b$ and define $\mathsf{Comp}\colon \{0,1\}^b \times \mathcal{X} \to \{0,1\}^b$ by

$$\mathsf{Comp}(h, (T, X)) \ =\ \mathsf{TComp}(h, T, X)$$

for all $h, X \in \{0,1\}^b$ and $T \in \{0,1\}^t$.

Now define B^* as follows against Comp^*, the Cascade transform of Comp (Figure 2). B^* runs A_{SSkein}. When the latter makes an oracle query $(\mathsf{N}, (\mathsf{type}_1, M_1), \ldots, (\mathsf{type}_r, M_r))$, adversary B^* computes $M^i \in \mathcal{X}^+$, $i \in \{1, \ldots, \lceil \mathsf{N}/b \rceil\}$, as follows:

$y \leftarrow \varepsilon$
$\mathsf{type}_0 \leftarrow \mathsf{TyCfg}$
$M_0 \leftarrow \mathsf{SMkConfig}(\mathsf{N})$
$d \leftarrow \lceil \mathsf{N}/b \rceil$
For $i = 1, \ldots, d$ do
    $M_{r+1} \leftarrow \mathsf{IntToStr}_{64}(i)$
    $\mathsf{type}_{r+1} \leftarrow \mathsf{TyOut}$
    $m \leftarrow 1$
    For $j = 0, \ldots, r+1$ do
        $(M', T_1, \ldots, T_{m'}) \leftarrow \mathsf{Encode}(\mathsf{M}_j, \mathsf{type}_j, 0^7, 0)$
        For $k = 1, \ldots, m'$ do $N_m \leftarrow (T_k, M'[k])$; $m \leftarrow m + 1$
    $M^i \leftarrow (N_1, \ldots, N_m)$

B^* invokes its own oracle on $M^1, \ldots, M^d$ to get back responses $Y_1, \ldots, Y_d$, and returns the first N bits of $Y^1 \| \cdots \| Y^d$ to A_{SSkein} as the response to its query. When A_{SSkein} halts with output a bit, B^* halts with the same output.

This construction guarantees that

$$\mathbf{Adv}^{\text{prf}}_{\mathsf{Comp}^*}(B^*) \ =\ \mathbf{Adv}^{\text{vol-prf}}_{\mathsf{SSkein}}(A_{\mathsf{SSkein}})\ . \tag{5}$$

Game $\mathsf{Real}_{H,P}$	Game Sim_S
Procedure **Initialize**	Procedure **Initialize**
Procedure $\mathbf{Fn}(N, X)$ Return $H^{\mathbf{Prim}}(N, X)$	Procedure $\mathbf{Fn}(N, X)$ Return $Y \xleftarrow{\$} \{0,1\}^N$
Procedure $\mathbf{Prim}(\alpha)$ Return $P(\alpha)$	Procedure $\mathbf{Prim}(\alpha)$ Return $S(\alpha)$

Figure 10: Games $\mathsf{Real}_{H,P}$ and Sim_S for the definition of a pseudorandom oracle (PRO).

Further note that B^* is prefix-free because of the use of the special TyOut type in the final element of each oracle query. Thus we can apply Lemma 6.1. Notice here that B^* makes q' oracle queries where $q' = \sum_{i=1}^{q} \lceil \mathsf{N}_i/b \rceil$, and that each oracle query consists of at most $l \cdot \lceil l'/b \rceil + 3$ elements from $\mathcal{X}$. Let D be the resulting prf-adversary against Comp. We now transform D into a prf-adversary A_{TComp} against TComp such that

$$\mathbf{Adv}^{\mathrm{prf}}_{\mathsf{TComp}}(A_{\mathsf{TComp}}) \;=\; \mathbf{Adv}^{\mathrm{prf}}_{\mathsf{Comp}}(D)\;. \tag{6}$$

The theorem follows from Equation (5), Equation (6), and Lemma 6.1. It remains to describe A_{TComp}.

A_{TComp} runs D. When the latter makes an oracle query $W \in \mathcal{X}$, adversary B parses it as $W = (T, X)$ with $|T| = t$ and $|X| = b$. It calls its own oracle on T, X and returns the answer to D. ▮

Our tweakable compression function TComp is obtained by running the tweakable block cipher E in Matyas-Meyer-Oseas [19] mode. As a result, we can prove that if E is a PRP (the standard assumption on E), then TComp is a PRF (Proposition 6.2). This would not be true for Davies-Meyer [22, 23] mode. Combining Proposition 6.2 and Theorem 6.4 we get that the PRF-security of SSkein is implied by the PRP security of E, an advantage of our construction compared to others.

7 Indifferentiability

7.1 Definitions and Tools

The definition of being a pseudorandom oracle [10, 20] extends to the case of VOL primitives. Let $H\colon\ D_0 \times D_1 \to \{0,1\}^*$ be a VOL function with oracle access to an idealized primitive P. A simulator is a stateful algorithm S. We let

$$\mathbf{Adv}^{\mathrm{pro}}_{H,P,S} \;=\; \Pr\left[\,\mathsf{Real}^{A}_{H,P} \Rightarrow 1\,\right] - \Pr\left[\,\mathsf{Sim}^{A}_{S} \Rightarrow 1\,\right]$$

where the games are defined in Figure 10. It is assumed that A never repeats an oracle query and never makes a query outside the domain of H.

Pre-image Awareness. We use the notion of pre-image awareness of [12]. let H be a function with oracle access to an idealized primitive P, and let E be an algorithm called the extractor. Let

Game $\mathsf{PrA}_{H,P,E}$	Procedure $\mathbf{Ex}(Z)$
Procedure $\mathbf{Initialize}$	$V[Z] \leftarrow E(Z,T)$
$T \leftarrow \varepsilon$	Return $V[Z]$
Procedure $\mathbf{Prim}(\alpha)$	Procedure $\mathbf{Finalize}(X)$
$\beta \xleftarrow{\$} P(\alpha)$	$Z \xleftarrow{\$} H^{\mathbf{Prim}}(X)$
$T \leftarrow T \| (\alpha, \beta)$	$V \xleftarrow{\$} \mathbf{Ex}(Z)$
Return β	Return $X \neq V$

Figure 11: Game $\mathsf{PrA}_{H,P,E}$ for the definition of a pre-image awareness (PrA).

A be an adversary. We let

$$\mathbf{Adv}^{\mathrm{pra}}_{H,P,E}(A) \;=\; \Pr\left[\, \mathsf{PrA}^{A}_{H,P,E} \Rightarrow \mathsf{true} \,\right]$$

where the game PrA is defined in Figure 11.

We need two results from [12]. The first is that if Comp is PrA then its cascade is PrA for suffix-free inputs. Let us now formalize this.

We say that a function $g\colon\ D \to \mathcal{X}^+$ is suffix-free if $X_1 \neq X_2$ implies $g(X_1) \not\leq_S g(X_2)$ for all $X_1, X_2 \in D$. Note that if g is suffix-free then it is also injective, something we will use below.

Lemma 7.1 (From [12]) Let $\mathsf{Comp}\colon\ \{0,1\}^b \times \mathcal{X} \to \{0,1\}^b$ be a function with oracle access to an idealized primitive P. Let $g\colon\ D \to \mathcal{X}^+$ be a suffix-free function. Let $\mathsf{Comp}^*\colon\ \{0,1\}^b \times \mathcal{X}^+ \to \{0,1\}^b$ be the cascade of Comp. Let $K \in \{0,1\}^b$ and let $\mathsf{h}\colon\ D \to \{0,1\}^b$ be defined by

$$\mathsf{h}(X) \;=\; \mathsf{Comp}^*(K, g(X))$$

for all $x \in D$. Then for any extractor E_{Comp} there exists an extractor E_{h} such that for any adversary A_{h} there exits an adversary A_{Comp} against Comp such that

$$\mathbf{Adv}^{\mathrm{pra}}_{\mathsf{h},\mathsf{P},E_{\mathsf{h}}}(A_{\mathsf{h}}) \;\leq\; \mathbf{Adv}^{\mathrm{pra}}_{\mathsf{Comp},\mathsf{P},E_{\mathsf{Comp}}}(A_{\mathsf{Comp}})\ .$$

The second result is that the composition of a PrA function and a RO is indifferentiable from a random oracle.

Lemma 7.2 (From [12]) Let $\mathsf{h}\colon\ D \to \{0,1\}^b$ be a function with access to an idealized primitive P and let R be a VOL RO. Let $\mathsf{H}\colon\ \mathbb{N} \times D \to \{0,1\}^b$ be the VOL function defined by

$$\mathsf{H}(\mathsf{N}, X) = \mathsf{R}(\mathsf{N}, \mathsf{h}(X))$$

for all $\mathsf{N} \in \mathbb{N}$ and $X \in D$. Then for any extractor E_{h} there exists a simulator S such that for any adversary A_{H} against H that makes q_1 queries to its $\mathbf{Fn}$ oracle and q_2 queries to its $\mathbf{Prim}$ oracle there exists an adversary A_{h} against h such that

$$\mathbf{Adv}^{\mathrm{pro}}_{\mathsf{H},\mathsf{P},S}(A_{\mathsf{H}}) \;\leq\; \mathbf{Adv}^{\mathrm{pra}}_{\mathsf{h},\mathsf{P},E_{\mathsf{h}}}(A_{\mathsf{h}})\ .$$

7.2 Pre-image Awareness of TComp

Proposition 7.3 *Let P be an ideal tweakable block cipher with key and block length b and tweak length t. Let*

$$\mathsf{TComp}(K,T,X) \;=\; \mathsf{E}(K,T,X) \oplus X$$

where $\mathsf{E}(\cdot,\cdot,\cdot) = P(1,(\cdot,\cdot,\cdot))$. Then there exists an extractor E such that for all A making q_p **Prim** *queries and q_e* **Ex** *queries,*

$$\mathbf{Adv}^{\mathrm{pra}}_{H,P,E}(A) \le \frac{2q_eq_p + q_p(q_p+1)}{2^b} \;.$$

Proof: Extractor E works as follows:

Algorithm $E(Z,T)$
  $(C_1,K_1,T_1,Z_1,W_1),\ldots,(C_r,K_r,T_r,Z_r,W_r) \leftarrow T$
  For $i = 1,\ldots,r$ do
    If $W_i \oplus Z_i = Z$ then
      If $c_i = 1$ then return (K_i,T_i,Z_i)
      If $c_i = -1$ then return (K_i,T_i,W_i)
  Return $\perp$

The analysis is the same as in [12]. ▌

7.3 Pre-image Awareness for UBI

Lemma 7.4 let $\mathsf{TComp}\colon\ \{0,1\}^b \times \{0,1\}^t \times \{0,1\}^b \to \{0,1\}^b$ be a function with oracle access to an idealized primitive P. Let $\mathsf{f}\colon\ \{0,1\}^b \times \mathsf{MsgSp} \times \mathsf{TypeSp} \times \{0,1\}^6 \times [0,2^{t-32}-1] \to \{0,1\}^b$ be obtained from TComp via the UBI transform. Let $K \in \{0,1\}^b$ and $\mathsf{F} = \mathsf{f}_K$. Then there is an extractor E_{F} such that for every extractor E_{TComp} and every adversary A_{F} there is an adversary A_{TComp} such that

$$\mathbf{Adv}^{\mathrm{pra}}_{\mathsf{F},P,E_{\mathsf{F}}}(A_{\mathsf{F}}) \le \mathbf{Adv}^{\mathrm{pra}}_{\mathsf{TComp},P,E_{\mathsf{TComp}}}(A_{\mathsf{TComp}}) \;.$$

Lemma 7.5 let $\mathsf{TComp}\colon\ \{0,1\}^b \times \{0,1\}^t \times \{0,1\}^b \to \{0,1\}^b$ be a function with oracle access to an idealized primitive P. Let $\mathsf{f}\colon\ \{0,1\}^b \times \mathsf{MsgSp} \times \mathsf{TypeSp} \times \{0,1\}^6 \times [0,2^{t-32}-1] \to \{0,1\}^b$ be obtained from TComp via the UBI transform. Let $K \in \{0,1\}^b$ and $\mathsf{F} = \mathsf{f}_K$ with the restricted domain $D \subset \mathsf{MsgSp} \times \mathsf{TypeSp} \times \{0,1\}^6 \times [0,2^{t-32}-1]$ such that for all tuples $(\mathsf{M},\mathsf{type},\mathsf{lvl},\mathsf{startl}) \in D$ it is the case that startl is a multiple of $b/8$, $\mathsf{startl} = 0$ if $\mathsf{M} = \varepsilon$, and $|\mathsf{M}| + \mathsf{startl} \cdot 2^3 \le (2^{t-32}-1)2^3$. Then there is an extractor E_{F} such that for every extractor E_{TComp} and every adversary A_{F} there is an adversary A_{TComp} such that

$$\mathbf{Adv}^{\mathrm{pra}}_{\mathsf{F},P,E_{\mathsf{F}}}(A_{\mathsf{F}}) \le \mathbf{Adv}^{\mathrm{pra}}_{\mathsf{TComp},P,E_{\mathsf{TComp}}}(A_{\mathsf{TComp}}) \;.$$

Proof: Let $\mathcal{X} = \{0,1\}^t \times \{0,1\}^b$ and $\mathsf{Comp}\colon\ \{0,1\}^b \times \mathcal{X} \to \{0,1\}^b$ be defined by $\mathsf{Comp}(h,(T,X)) = \mathsf{TComp}(h,T,X)$. Let $D = \mathsf{MsgSp} \times \mathsf{TypeSp} \times \{0,1\}^6 \times [0,2^{t-32}-1]$ and define $g\colon\ D \to \mathcal{X}^+$ via

Algorithm $g(\mathsf{M}, \mathsf{type}, \mathsf{lvl}, \mathsf{startl})$
$(M, T_1, \ldots, T_m) \leftarrow \mathsf{Encode}(\mathsf{M}, \mathsf{type}, \mathsf{lvl}, \mathsf{startl})$
For $i = 1, \ldots, m$ do $M_i^* \leftarrow (T_i, M[i])$
$M^* \leftarrow (M_1^*, \ldots, M_m^*)$
Return M^*

Then for all $(\mathsf{M}, \mathsf{type}, \mathsf{lvl}, \mathsf{startl}) \in D$ we have

$$\mathsf{F}(\mathsf{M}, \mathsf{type}, \mathsf{lvl}, \mathsf{startl}) \;=\; \mathsf{Comp}^*(K, g(\mathsf{M}, \mathsf{type}, \mathsf{lvl}, \mathsf{startl}))\ .$$

But Lemma 4.2 says that g is suffix-free. The lemma now follows from Lemma 7.1. ∎

7.4 Indifferentiability for SSkein

Fix throughout this subsection a key $\mathsf{K} \in \{0,1\}^b$ to be used as an initial vector. Let

$$\begin{aligned} D &= \mathsf{OutLens} \times ((\mathsf{TypeSp} - \{\mathsf{TyCfg}, \mathsf{TyOut}\}) \times \mathsf{MsgSp})^* \\ \mathcal{X} &= \mathsf{MsgSp} \times (\mathsf{TypeSp} - \{\mathsf{TyOut}\}) \times \{0,1\}^7 \times [0, 2^{t-32} - 1] \\ \mathsf{TwSp}_2 &= \{\mathsf{MkTw}(1, 1, \mathsf{TyOut}, 0, 0^7, 8)\} \\ \mathsf{TwSp}_1 &= \{0,1\}^t \backslash \mathsf{TwSp}_2\ . \end{aligned}$$

Define $\mathsf{g}\colon\ D \to \mathcal{X}^+$ by

Algorithm $\mathsf{g}((\mathsf{N}, ((\mathsf{type}_1, \mathsf{M}_1), \ldots, (\mathsf{type}_r, \mathsf{M}_r))))$
$(\mathsf{type}_0, \mathsf{M}_0) \leftarrow \mathsf{SMkConfig}(\mathsf{N})$
For $i = 0, \ldots, r$ do $x_i \leftarrow (\mathsf{M}_i, \mathsf{type}_i, 0^7, 0)$
Return $(x_0, \ldots, x_r)$

Let $P_i\colon\ \{0,1\}^b \times \mathsf{TwSp}_i \times \{0,1\}^b \to \{0,1\}^b$ be ideal tweakable block ciphers and let $\mathsf{E}_i = P_i(1, (\cdot, \cdot, \cdot))$ for $i = 1, 2$. Let $\mathsf{TComp}_i\colon\ \{0,1\}^b \times \mathsf{TwSp}_i \times \{0,1\}^b \to \{0,1\}^b$ be defined by

$$\mathsf{TComp}_i(\mathsf{K}, \mathsf{T}, W) \;=\; \mathsf{E}_i(\mathsf{K}, \mathsf{T}, W) \oplus W\ .$$

Let $\mathsf{f}_i\colon\ \{0,1\}^b \times \mathcal{X} \to \{0,1\}^b$ be obtained by applying the UBI transform to TComp_i. Let $\mathsf{f}_i^*\colon\ \{0,1\}^b \times \mathcal{X}^+ \to \{0,1\}^b$ be the cascade of f_i. Define $\mathsf{h}\colon\ D \to \{0,1\}^b$ by

$$\mathsf{h}((\mathsf{N}, ((\mathsf{type}_1, \mathsf{M}_1), \ldots, (\mathsf{type}_r, \mathsf{M}_r)))) = \mathsf{f}_1^*(\mathsf{K}, \mathsf{g}((\mathsf{N}, ((\mathsf{type}_1, \mathsf{M}_1), \ldots, (\mathsf{type}_r, \mathsf{M}_r))))))\ . \qquad (7)$$

Also define $\mathsf{Out}\colon\ \mathsf{OutLens} \times \{0,1\}^b \to \{0,1\}^*$ by

Algorithm $\mathsf{Out}(\mathsf{N}, h)$
$y \leftarrow \varepsilon\ ;\ d \leftarrow \lceil \mathsf{N}/b \rceil\ ;\ T_1 \leftarrow \mathsf{MkTw}(1, 1, \mathsf{TyOut}, 0, 0^7, 8)$
For $i = 1, \ldots, d$ do
  $M_i \leftarrow \mathsf{IntToStr}_{64}(i) \| 0^{b-64}$
  $y_i \leftarrow \mathsf{TComp}_2(h, T_1, M_i)$
  $y \leftarrow y \| y_i$
Return $y[1 \ldots \mathsf{N}]$

Define $\mathsf{H}\colon\ \mathsf{OutLens} \times ((\mathsf{TypeSp} - \{\mathsf{TyCfg}, \mathsf{TyOut}\}) \times \mathsf{MsgSp})^* \to \{0,1\}^*$ by

$$\mathsf{H}(\mathsf{N}, ((\mathsf{type}_1, \mathsf{M}_1), \ldots, (\mathsf{type}_r, \mathsf{M}_r))) = \mathsf{Out}(\mathsf{N}, \mathsf{h}((\mathsf{N}, ((\mathsf{type}_1, \mathsf{M}_1), \ldots, (\mathsf{type}_r, \mathsf{M}_r))))) \ . \qquad (8)$$

Now observe that H is exactly Skein with fixed IV K, so our goal is to prove that H is a PRO. The proof has three steps.

The first step is to show that Out is a VOL PRO assuming P_2 is an ideal tweakable block cipher. This would follow easily if TComp_2 was a PRO under this assumption, but this unfortunately is *not* true. What saves us here is that the M_i input to TComp_2 in Out takes on a few constant values and is not under adversarial control. For each fixed i the function $h \to \mathsf{TComp}_2(h, T_1, M_i)$ is a PRO and it follows that Out is a VOL PRO. The definition of indifferentiability then implies that we can replace Out in Equation (8) by a VOL RO. Now, Lemma 7.2 says that H is a PRO as long as h is PrA.

Finally, Equation (7) and Lemma 7.1 imply that h is PrA as long as f_1 is PrA and g is suffix free. The first is implied by Lemma 7.4 and the second is true by the rules on types.

Both h and Out are based on the same ideal primitive, namely E. However, Out only uses E with the TyOut type that is never used by h, so the two are independent. This is used crucially above. Our framework captures it by considering separate ideal tweakable block ciphers P_1, P_2 on disjoint tweak spaces that together define the ideal tweakable block cipher on the full tweak space.

References

[1] M. Bellare. New proofs for NMAC and HMAC: Security without collision-resistance. In C. Dwork, editor, *Advances in Cryptology – CRYPTO 2006*, volume 4117 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, Germany, 2006.

[2] M. Bellare, R. Canetti, and H. Krawczyk. Keying hash functions for message authentication. In N. Koblitz, editor, *Advances in Cryptology – CRYPTO'96*, volume 1109 of *Lecture Notes in Computer Science*, pages 1–15, Santa Barbara, CA, USA, Aug. 18–22, 1996. Springer-Verlag, Berlin, Germany.

[3] M. Bellare, R. Canetti, and H. Krawczyk. Pseudorandom functions revisited: The cascade construction and its concrete security. In *37th Annual Symposium on Foundations of Computer Science*, pages 514–523, Oct. 14–16,, Burlington, Vermont 1996. IEEE Computer Society Press.

[4] M. Bellare, J. Kilian, and P. Rogaway. The security of the cipher block chaining message authentication code. In Y. Desmedt, editor, *Advances in Cryptology – CRYPTO'94*, volume 839 of *Lecture Notes in Computer Science*, pages 341–358, Santa Barbara, CA, USA, Aug. 21–25, 1994. Springer-Verlag, Berlin, Germany.

[5] M. Bellare and T. Ristenpart. Multi-property-preserving hash domain extension and the emd transform. In *Advances in Cryptology – ASIACRYPT 2006*, Lecture Notes in Computer Science. Springer-Verlag, Berlin, Germany, Dec. 2006.

[6] M. Bellare and P. Rogaway. Code-based game-playing proofs and the security of triple encryption. In *Advances in Cryptology – EUROCRYPT 2006*, Lecture Notes in Computer Science. Springer-Verlag, Berlin, Germany, 2006.

[7] M. Bellare and B. Yee. Forward security in private key cryptography. In M. Joye, editor, *Topics in Cryptology – CT-RSA 2003*, volume 2612 of *Lecture Notes in Computer Science*, pages 1–18, San Francisco, CA, USA, Apr. 13–17, 2003. Springer-Verlag, Berlin, Germany.

[8] J. Black, P. Rogaway, and T. Shrimpton. Black-box analysis of the block-cipher-based hash-function constructions from PGV. In M. Yung, editor, *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, Germany, 2002.

[9] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudorandom bits. *SIAM Journal on Computing*, 13(4):850–864, 1984.

[10] J.-S. Coron, Y. Dodis, C. Malinaud, and P. Puniya. Merkle-Damgård revisited: How to construct a hash function. In V. Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, Germany, 2005.

[11] I. Damgård. A design principle for hash functions. In G. Brassard, editor, *Advances in Cryptology – CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, Santa Barbara, CA, USA, Aug. 20–24, 1990. Springer-Verlag, Berlin, Germany.

[12] Y. Dodis, T. Ristenpart, and T. Shrimpton. Salvaging merkle-damgard for practical applications. In *Advances in Cryptology – EUROCRYPT 2009*, Lecture Notes in Computer Science, Santa Barbara, CA, USA, 2009. Springer-Verlag, Berlin, Germany.

[13] N. Ferguson, S. Lucks, B. Schneier, D. Whiting, M. Bellare, T. Kohno, J. Callas, and J. Walker. The Skein hash function family. Submission to the NIST Cryptographic Hash Algorithm Competition, October 2008.

[14] N. Ferguson, S. Lucks, B. Schneier, D. Whiting, M. Bellare, T. Kohno, J. Callas, and J. Walker. The skein hash function family. Submission to the NIST Cryptographic Hash Algorithm Competition, October 2008.

[15] Keyed hash message authentication code. American Bankers Association, ANSI X9.71, 2000.

[16] The keyed-hash message authentication code (hmac). National Institute of Standards and Technology, NIST FIPS PUB 198, U.S. Department of Commerce, Mar. 2002.

[17] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-hashing for message authentication. IETF Internet Request for Comments 2104, Feb. 1997.

[18] M. Liskov, R. Rivest, and D. Wagner. Tweakable block ciphers. In M. Yung, editor, *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 31–46. Springer-Verlag, Berlin, Germany, 2002.

[19] S. Matyas, C. Meyer, and J. Oseas. Generating strong one-way functions with cryptographic algorithm. *IBM Technical Disclosure Bulletin*, 27(10A), 1985.

[20] U. M. Maurer, R. Renner, and C. Holenstein. Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In M. Naor, editor, *TCC 2004: 1st Theory of Cryptography Conference*, volume 2951 of *Lecture Notes in Computer Science*, pages 21–39, Cambridge, MA, USA, Feb. 19–21, 2004. Springer-Verlag, Berlin, Germany.

[21] R. C. Merkle. One way hash functions and DES. In G. Brassard, editor, *Advances in Cryptology – CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 428–446, Santa Barbara, CA, USA, Aug. 20–24, 1990. Springer-Verlag, Berlin, Germany.

[22] C. Mitchell, F. Piper, and P. Wild. Digital signatures. In G. J. Simmons, editor, *Contemporary Cryptology: The Science of Information Integrity*, 1991.

[23] J. J. Quisquater and M. Girault. 2n-bit hash-functions using n-bit symmetric block cipher algorithms. In J.-J. Quisquater and J. Vandewalle, editors, *Advances in Cryptology – EUROCRYPT'89*, volume 434 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, Germany, Apr. 10–13, 1990.

[24] A. C. Yao. Theory and applications of trapdoor functions. In *23rd Annual Symposium on Foundations of Computer Science*, pages 80–91, Chicago, Illinois, Nov. 3–5, 1982. IEEE Computer Society Press.