Random Oracle Combiners: Breaking the Concatenation Barrier for Collision-Resistance

Yevgeniy Dodis[1], Niels Ferguson[2], Eli Goldin[1], Peter Hall[1], and Krzysztof Pietrzak[3]

[1]New York University (dodis@cs.nyu.edu,eg3293@nyu.edu,pf2184@nyu.edu)
[2]Microsoft (niels@microsoft.com)
[3]IST Austria (krzysztof.pietrzak@ist.ac.at)

Abstract

Suppose two parties have hash functions h_1 and h_2 respectively, but each only trusts the security of their own. We wish to build a *hash combiner* C^{h_1,h_2} which is secure so long as either one of the underlying hash functions is. This question has been well-studied in the regime of *collision resistance*. In this case, concatenating the two hash outputs clearly works. Unfortunately, a long series of works (Boneh and Boyen, CRYPTO'06; Pietrzak, Eurocrypt'07; Pietrzak, CRYPTO'08) showed no (noticeably) shorter combiner for collision resistance is possible.

We revisit this pessimistic state of affairs, motivated by the observation that collision-resistance is insufficient for many applications of cryptographic hash functions anyway. We argue the right formulation of the "hash combiner" is what we call *random oracle (RO) combiners*.

Indeed, we circumvent the previous lower bounds for collision resistance by constructing a simple length-preserving RO combiner

$$\widetilde{C}^{h_1,h_2}_{\mathcal{Z}_1,\mathcal{Z}_2}(M) = h_1(M,\mathcal{Z}_1) \oplus h_2(M,\mathcal{Z}_2),$$

where $\mathcal{Z}_1, \mathcal{Z}_2$ are random salts of appropriate length. We show that this extra randomness is necessary for RO combiners, and indeed our construction is somewhat tight with this lower bound.

On the negative side, we show that one cannot generically apply the composition theorem to further replace "monolithic" hashes h_1 and h_2 by some simpler indifferentiable construction (such as the *Merkle-Damgård transformation*) from smaller components, such as fixed-length compression functions. Despite this issue, we directly prove collision resistance of the Merkle-Damgård variant of our combiner, where h_1 and h_2 are replaced by iterative Merkle-Damgård hashes applied to fixed-length compression functions. Thus, we can still subvert the concatenation barrier for collision-resistance combiners using practically small components.

Keywords: hash functions, combiners, random oracle model, indifferentiability framework, collision resistance

1 Introduction

Imagine you are a major software company and wrote some important cryptographic code utilizing a "cryptographic hash function". You believe that your favorite function h_1 is the best such function to use. One of your clients would like to buy your software. However, the regulations of their country mandates they use a different cryptographic hash function h_2, and they are not allowed to use a different hash function. Your company does not believe h_2 is as secure as h_1, and you do not wish to substitute h_1 by h_2. For example, in the case where h_2 gets broken, you do not want any bad publicity that your software application could suddenly become insecure, even though you wanted to use the "secure" hash function h_1. But you also want to make a sale, as the customer is a big client.

Hash combiners provide a practical solution to this dilemma. Imagine your software will utilize some combined hash function $C = C^{h_1,h_2}$, which depends in a black-box way on both h_1 and h_2, and which is secure as long as either h_1 or h_2 is secure. In this case you can convince the customer to still buy your software, as the security of the hash function C is at least as good as that of h_2. But you are not worried of a future attack, since C is also as secure as your trusted hash function h_1.

How can we build such a combiner $C = C^{h_1,h_2}$? That depends on the notion of security that we want. So far, most of the literature of hash combiners focused on the notion of *collision-resistance* as a target. In this case, the following simple *concatenation combiner* works:

$$C^{h_1,h_2}(M) = h_1(M)\|h_2(M)$$

Indeed, a collision for C clearly implies a collision for both h_1 and h_2. Unfortunately, this construction also doubles the output size of the hash function (assuming, for simplicity, that both hash outputs are the same length). As such, this restriction severely limits the applicability of combiner in practice. For example, in hash-then-sign signatures (such as FDH [3] or BLS [5]), this forces one to use much heavier public-key cryptography, and similar considerations apply for virtually any real-world application we can think off.

Perhaps we can do better? Unfortunately, a series of works [4, 21, 22] showed that this is not the case, even if the combiner is allowed to be randomized (i.e., keyed, with the key chosen independently of h_1 and h_2).

Is There a Way Out? At first, it seems like we are stuck, since collision-resistance is one of the most basic security properties of hash function that we will need to preserve. Fortunately, as was first observed by Mittelbach [19] (see also [18]) and continued in this work, this conclusion is perhaps overly pessimistic. Namely, most applications of "cryptographic hash functions" already require more than collision-resistance. For example, hash-then-sign signatures [3, 5] are typically proven in the random oracle model (ROM), and it is in fact unlikely that a standard-model property of hash functions will be enough to instantiate these widely-used schemes (see, e.g., [9, 20, 8]).

One potential solution, then, is to design a *multi-property preserving* combiner [11], which simultaneously preserves multiple (or possibly all?) required security properties of hash functions. In this setting, the concatenation combiner would fail even more miserably — for example, if one of the input hash functions is not pseudorandom, the concatenation combiner will not be pseudorandom regardless of whether the other is. Hence, it does not solve our original problem.

A better approach, studied by [19, 18], is to try to prove collision-resistance (and other individual properties) of the combiner in the random oracle model, where one of the hash functions is modeled as a *random oracle* (RO), and the other may adversarially query the random oracle. While very useful (see Section 1.2 for more discussion), this still does not address the issue that most applications of hash functions might need stronger properties than collision resistance. It is also a bit underwhelming to start with a random oracle but only achieve weaker security properties.

Thus, in this work, we take this approach one step beyond, and introduce the following, even better solution to the problem:

<u>*Random Oracle Combiners!*</u>

Ignoring for a second that this notion was never defined prior to this work — something we will fix shortly — this solution seems to be a much better fit for our application. Indeed, it addresses the fact that most application of cryptographic hash function anyway require something stronger than collision-resistance; in most settings, a random oracle usually suffices. Even more, perhaps when one of the hash functions h_1 or h_2 is a true random oracle, the lower bound for collision-resistance combiners mentioned above would no longer hold. Intuitively, a stronger assumption (RO rather than collision-resistance) on the hash function might yield a similarly stronger conclusion on the combiner, while being much more efficient. Indeed, this is precisely what we will show in this work: *There does exist simple length-preserving random oracle combiners.*

1.1 Our Results

We now describe our results in more detail.

Defining RO-combiners. Our first contribution is a formal definition of an RO-combiner. We follow the *indifferentiability framework* of Maurer, Renner, and Holenstein [16] similarly to Coron et al. [7] to say that the combiner C should be indifferentiable from a random oracle provided h_1 or h_2 is an RO.

In our case, we must be careful as to how h_1 and h_2 may depend on each other so as to accurately model real use-cases. For concreteness, suppose h_1 is a random oracle. We would like our combiner C^{h_1,h_2} to be indifferentiable from a random oracle even if h_2 *arbitrarily depends on* h_1. If h_2 was limited to not depend on h_1 at all, trivial constructions such as $C^{h_1,h_2}(M) = h_1(M) \oplus h_2(M)$ would suffice. No reasonable practitioner would be happy with such a combiner, though, as even the reasonable choice of $h_2 = h_1$ results in $C^{h_1,h_2} = 0$ on all inputs. On the other hand, if h_2 can depend on h_1 in some "exponentially-complex" way, this seems to give the adversarial hash function too much power in a way that does not model the real world.

Thus, we settle on the following compromise, which we feel adequately balances real-world threats of the attacker. We allow h_2 to implement an arbitrary (even unbounded-size) circuit g, but on every input, g is allowed to make only a large but bounded number of black-box calls to h_1. This rules out, for example, h_2 being able to access the entire truth table of the random oracle on a single function call. We refer to Definition 2.4 (and further discussion of our notion in Section 2.2) to formally model this *adversarial implementation.*

To see that this notion is non-trivial, we show that no deterministic RO-combiners exist. The formal proof of this result is rather involved (see Theorem 4.4), but the intuition is as follows: For any deterministic combiner C (which outputs just one bit) we will bias one of the hash functions in a way that $C(0)$ will be biased. This is accomplished if either $C(0)$ does not "meaningfully depend" on (say) h_2, in which case our task is easily accomplished by controlling h_1, or if both h_1 and h_2 have an effect on the output of the combiner, in which case we will program h_2 to bias the output. Here "meaningfully depend" roughly corresponds to choosing h_1 at random, and seeing if further choosing h_2 at random affect the value of $C(0)$. Otherwise, we will do "rejection sampling" inside the adversarial function h_2 that we design. The adversarial hash h_2 will attempt to evaluate $C(0)$ using its oracle gates to h_1, and will keep randomly changing its answers to h_2 until the result of the $C(0)$ evaluation is 0.

A Length-Preserving RO-combiner. Having developed some intuition why building RO-combiners is non-trivial, and requires some randomness, we proceed with our main positive result. Namely, we show that the following *length-preserving* (and, necessarily, randomized) RO-combiner is secure:

$$\widetilde{C}^{h_1,h_2}_{\mathcal{Z}_1,\mathcal{Z}_2}(M) = h_1(M, \mathcal{Z}_1) \oplus h_2(M, \mathcal{Z}_2),$$

where $\mathcal{Z}_1, \mathcal{Z}_2$ are sufficiently long random salts. We refer to Theorem 4.1.

Specifically, the lengths of $\mathcal{Z}_1$ and $\mathcal{Z}_2$ are slightly longer (roughly, by security parameter) than the length of the hash message M. As a high-level intuition, the key to the security of this RO-combiner comes from the fact that no evaluation of $\widetilde{C}(M)$ can completely call $h_1(*, \mathcal{Z}_1)$ inside $h_2(M, \mathcal{Z}_2)$, and vice versa. This is because the message M (which can be chosen by the attacker *after* it learns $\mathcal{Z}_1, \mathcal{Z}_2$) is not long enough to encode the entirety of $\mathcal{Z}_1$ or $\mathcal{Z}_2$. Thus, our indifferentiability simulator (run in the ideal world, where C is now a true random oracle) does not get stuck when adapting one of the hash functions h_1 or h_2 to be consistent with RO C.

We also show that the lower bound on $|\mathcal{Z}_1|, |\mathcal{Z}_2|$ required by our security analysis is tight, by presenting a simple attack on any $\widetilde{C}^{h_1,h_2}_{\mathcal{Z}_1,\mathcal{Z}_2}$ when using significantly shorter salts. For more details, see Theorem 4.3.

Can We Use Short Compression Functions? While our main positive result is length-preserving and already illustrates the usefulness of our approach of modeling "hash function combiners" as RO-combiners, our particular construction $\widetilde{C}^{h_1,h_2}_{\mathcal{Z}_1,\mathcal{Z}_2}$ is not yet applicable to practical cases in the following sense: Even for reasonably-sized inputs M, the underlying hash calls $h_1(M, \mathcal{Z}_1)$ and $h_2(M, \mathcal{Z}_2)$ are over inputs which do not fit into a typical block of existing cryptographic hash functions, such as SHA-2 or SHA-3 (in part, because the salts $\mathcal{Z}_1$ and $\mathcal{Z}_2$ are relatively long).

Instead, practical hash functions typically iterate a given compression function using some mode of operation, such as the Merkle-Damgård transformation. At first, it is tempting to attempt simply apply the composition theorem in the indifferentiability framework [16] — Instead of assuming h_1 and h_2 are monolithic hash functions on "long inputs" $(M, \mathcal{Z})$, imagine they are modeled as fixed-length compression functions on "short-inputs", and we appropriately redefine our RO-combiner to be

$$\mathrm{SufC}^{h_1,h_2}_{\mathcal{Z}_1,\mathcal{Z}_2}(M) = h_1^*(M, \mathcal{Z}_1) \oplus h_2^*(M, \mathcal{Z}_2),$$

where the notation $f^*(m_1, \ldots m_t) = f(m_t, f(m_{t-1}, \ldots, f(m_1, 0) \ldots))$ corresponds to the Merkle-Damgård (MD) evaluation of the compression function f.

Now, we could use the earlier indifferentiability result of [7] that applying the MD evaluation a fixed number of times on a compression function modeled as a fixed-length random oracle is by itself indifferentiable from a (longer-input) monolithic random oracle. Combined with our previous result, this would imply that SufC is still a good RO-combiner, but now with much more practical parameters for h_1 and h_2.

Unfortunately, this reasoning turns out to be false for a rather subtle reason. Our indifferentiability notion for RO-combiners is technically a two stage game [23], as explained in detail in Section 3.1. As such, standard composition theorem cannot be applied to argue security of such game. Indeed, to see this explicitly, we show that a similar-looking combiner

$$\mathrm{PreC}^{h_1,h_2}_{\mathcal{Z}_1,\mathcal{Z}_2}(M) = h_1^*(\mathcal{Z}_1, M) \oplus h_2^*(\mathcal{Z}_2, M).$$

is completely insecure as an RO-combiner on fixed-length compression functions. Namely, the difference between SufC and PreC is whether the salts $\mathcal{Z}_1, \mathcal{Z}_2$ are appended or prepended to the actual message M. For the "monolithic" combiner $\widetilde{C}^{h_1,h_2}_{\mathcal{Z}_1,\mathcal{Z}_2}(M) = h_1(M,\mathcal{Z}_1) \oplus h_2(M,\mathcal{Z}_2)$, this clearly does not matter. Yet, the prepended construction is insecure when instantiated with the Merkle-Damgård based variant. Intuitively, all the useful information about the salts $\mathcal{Z}_1, \mathcal{Z}_2$ is hashed into compact descriptions $h_1^*(\mathcal{Z}_1)$ and $h_2^*(\mathcal{Z}_2)$, respectively, which may then be, e.g., included within M. See Theorem 5.4 for more details.

Direct Collision-Resistance. While the composition theorem does not immediately imply security of $\mathrm{SufC}(M) = h_1^*(M,\mathcal{Z}_1) \oplus h_2^*(M,\mathcal{Z}_2)$, one could still attempt a direct security proof that this is a secure RO-combiner. We leave this as a great open question. However, as an initial step in this direction, we show that $\mathrm{SufC}(M)$ is at least collision-resistant in our model (Theorem 6.3). This gives another meaningful (and provably secure) way to circumvent the concatenation barrier for collision-resistant combiners [4, 21, 22], albeit leveraging stronger assumptions.

We note that our construction is basically a "cryptophia short combiner" as considered in [19, 18]. However, we achieve better efficiency and our result is more general in several senses. We discuss this comparison further below.

1.2 Related Work

Combiners for cryptographic primitives were first considered by [15] and [14], followed by numerous works including [2, 13, 14, 15, 17]. Combiners for collision-resistance have been of particular interest. In this space, the concatenation combiner belongs to folklore, and proving its optimality (in terms of output length) was established by a series of works [4, 6, 21, 22]. So-called multi-property preserving combiners have been defined and constructed by [11, 12]. As these combiners are require preserving collision-resistance, they necessarily have long output length, in line with the lower bounds above. We mention that they also preserve indifferentiability from a random oracle, but in a much weaker model than our model, where the "adversarial random oracle" cannot depend on the good one. In particular, in this model even the XOR function is a good combiner. Thus, even ignoring the output length, the combiners of [11, 12] would not be secure random oracle combiners according to our definition.

Mittelbach [19] considered a similar notion to RO-combiners called "cryptophia short combiners" (additionally, a follow up work [18] identifies and fixes a flaw in the original paper). Mittelbach [19] was the first to show that the lower bounds on the output length of collision-resistance combiners (that is, nearly double the length of the outputs of the input hash functions [4, 21, 22]) can be overcome by assuming one of the two functions is ideal, even under the strong assumption that the other hash function can arbitrarily depend on the ideal one. Our work follows in this direction, making the following improvements:

Security: While [19] shows that the combined function achieves particular security properties (collision-resistance, pseudorandomness, one-wayness), we show indifferentiability from a random oracle, which is strictly stronger.

Efficiency: Our combiner is conceptually simpler and more efficient: To hash a message of length $\ell = k \cdot n$, we need to hash $2\ell + n = (2k+1)n$ bits with each of the hash functions. In [18] it's $5k \cdot n$ bits.

Completeness of Assumptions: The Cryptophia combiner is defined for arbitrarily compressing hash functions. This is justified (in the Remark of Section 3.3 of [19]) by stating their definition is one-stage, allowing them to replace the large-input hash by something indifferentiable from a random oracle, such as the Merkle-Damgard composition of a compression function. However, their model (similarly to ours) allows the adversarial hash function to make oracle queries to the honest one, and so their security game is two stage. Thus, just as with our results from Section 3, their security proof does not hold when their combiner is instantiated with "real-world" hash functions. We get around this issue by providing a direct proof of collision-resistance when our combiner is instantiated with Merkle-Damgard hash functions (Theorem 6.3).

2 Preliminaries

Definition 2.1. *A random oracle $\mathcal{H} : \{0,1\}^m \to \{0,1\}^n$ is an oracle giving access to a function $\{0,1\}^m \to \{0,1\}^n$ chosen uniformly at random during initialization.*

We will make frequent use of the following theorem, which we will call a "compression argument." The proof of this theorem is folklore, but is easy to show using the pigeonhole principle.

Theorem 2.2. *Let $\mathsf{Enc} : \mathcal{S} \to \mathcal{T}$ and $\mathsf{Dec} : \mathcal{T} \to \mathcal{S}$ be two algorithms such that*

$$\Pr_{s \xleftarrow{\$} \mathcal{S}} [\mathsf{Dec}(\mathsf{Enc}(s)) = s] \geq \epsilon.$$

Then $\epsilon \cdot |\mathcal{S}| \leq |\mathcal{T}|$.

2.1 Hash Combiners

We will use the term hash combiner to refer to the following syntax:

Definition 2.3. *A hash combiner $C_{\mathcal{Z}}^{h_1,h_2} : \{0,1\}^\ell \to \{0,1\}^s$ is an algorithm, keyed by randomness $\mathcal{Z}$ with $|\mathcal{Z}| = k$, with oracle access to two functions $h_1, h_2 : \{0,1\}^m \to \{0,1\}^n$.*

We also will refer to adversarial implementations of hash functions. When defining security of combiners, we will want security to hold against all adversarial implementations.

Definition 2.4. *An adversarial implementation of a hash function is an oracle circuit $g^{\mathcal{O}} : \{0,1\}^m \to \{0,1\}^n$. We typically bound the number of queries made by g to its oracle by T_g.*

In particular, for a hash combiner $C_{\mathcal{Z}}^{h_1,h_2} : \{0,1\}^\ell \to \{0,1\}^s$ to be secure, we will want $C_{\mathcal{Z}}^{h,g^h}$ and $C_{\mathcal{Z}}^{g^h,h}$ to be secure for any adversarial implementation g when h is implemented as an honest hash function. In this work, we will consider the honest hash function to always be implemented as a random oracle.

2.2 Alternative Models

We could consider the model where the adversarial hash function $g^{\mathcal{O}} : \{0,1\}^m \to \{0,1\}^n$ is given access to a common reference string of length r. However, if we bound the number of queries made to $\mathcal{O}$ to q, these models are equivalent up to a $\frac{r}{n}\frac{q}{2^m}$ additive factor. This is because a deterministic g^h can simulate the common reference string by non-uniformly fixing some prefix c of length $m - \log(r/n)$ and using $h(c,0), \ldots, h(c, r/n)$ as its internal randomness.

We may wish to consider the model where the circuit g is chosen efficiently by the distinguisher instead of being fixed as a separate nonuniform adversary. Note, however, that this model is strictly weaker than the one described in the previous paragraph. Thus, as our positive results hold in a stronger model, they will hold in this model as well. For our negative results, our constructions will indeed be able to be efficiently generated.

Finally, we may wish to consider a one-stage model, where the adversarial hash function is not given oracle access to the honest random oracle. This model is described in more detail in Section 7.2, but we do not analyze this model in detail in this work.

2.3 Notation

Throughout this work, we will use certain conventions to make comparisons between constructions and attacks simpler. In general, for a (randomized) hash function h, m will refer to the input length, n will refer to the output length, and k will refer to the length of the randomness. For combiners, we will use ℓ to refer to the input length. Any other notation will be detailed in the section in which it is relevant.

3 Indifferentiability and Hash Combiners

We first introduce the notion of indifferentiability. Indifferentiability was first defined by Maurer, Renner, and Holenstein [16] as a way of arguing (among other things) a notion that some idealized object could be "replaced" by some system or protocol without sacrificing any of the properties of the idealized object. More specifically, in this work, we desire indifferentiability of a hash function combiner with the random oracle. Note that this notion is strictly stronger than collision-resistance. We present the definitions necessary for this notion below, starting with that of indifferentiability itself.

Definition 3.1 (Maurer, Renner, Holenstein 2003)**.** *Let $\mathcal{F}, \mathcal{G}$ be two ideal primitives, and let $\mathcal{S}_{\mathcal{Z}}^{\mathcal{F}}$ be a family of constructions of $\mathcal{G}$ from $\mathcal{F}$. We say that $\mathcal{S}$ is (statistically) $(T_{\mathsf{Sim}}, \epsilon)$-indifferentiable from $\mathcal{G}$ if there exists a simulator $\mathsf{Sim}^{\mathcal{O}}$ making at most T_{Sim} queries to its oracle such that the following holds:*
For all (unbounded) oracle algorithms $\mathcal{D}$,

$$\left| \Pr_{\mathcal{Z} \xleftarrow{\$} \mathcal{U}} [\mathcal{D}(\mathcal{Z}, \mathcal{F}, \mathcal{S}_{\mathcal{Z}}^{\mathcal{F}}) \to 1] - \Pr_{\mathcal{Z} \xleftarrow{\$} \mathcal{U}} [\mathcal{D}(\mathcal{Z}, \mathsf{Sim}^{\mathcal{G}}(\mathcal{Z}), \mathcal{G}) \to 1] \right| \leq \epsilon$$

where $\mathcal{U}$ is the uniform distribution.

Theorem 3.2. *(Informal) If $\mathcal{S}^{\mathcal{F}}$ is ϵ-indifferentiable from $\mathcal{G}$, then for any single-stage cryptographic application using $\mathcal{G}$, replacing $\mathcal{G}$ with $\mathcal{S}^{\mathcal{F}}$ will have at most ϵ security loss.*

That is, if something is indifferentiable from a random oracle, then it can be used as a random oracle in almost all applications. Note there are some restrictions on the applications of indifferentiability. In particular, for cryptographic systems designed against multiple adversaries who cannot communicate, replacing an ideal model with something indifferentiable may not preserve security [23].

As a random oracle is in some sense the best possible hash function, an ideal hash combiner would be indifferentiable from a random oracle if one of the underlying hash functions is a random oracle. More formally, we give the following definition.

Definition 3.3. *Let $C_{\mathcal{Z}}^{h_1,h_2}$ be a combiner. Let $\mathcal{H},\mathcal{G}$ be two random oracles. We say that C is* $\underline{(T_g, T_{\mathsf{Sim}}, \epsilon)\text{-random-oracle-secure}}$ *if, for all oracle circuits $g^{\mathcal{O}}$ making at most T_g oracle queries, both $C_{\mathcal{Z}}^{\mathcal{H},g^{\mathcal{H}}}$ and $C_{\mathcal{Z}}^{g^{\mathcal{H}},\mathcal{H}}$ are $(T_{\mathsf{Sim}}, \epsilon)$-indifferentiable from $\mathcal{G}$. An equivalent formulation is illustrated in Figure 1.*

Note in particular that we allow our combiner to be keyed. That is, the combiner has access to some public randomness chosen after the adversarial hash function is determined. In the next section, we will show that this is indeed necessary in order for our definition to be achievable.

3.1 On Composability of Random-Oracle-Security

One weakness of our definition of random-oracle-security is that this definition is itself a two-stage game. That is, an adversary against random-oracle-security is composed of both the adversarial implementation $g^{\mathcal{O}}$ as well as the distinguisher $\mathcal{D}$, and neither party is allowed to communicate. Therefore, if one were to instantiate the honest oracle for a random-oracle-combiner with a hash function which is indifferentiable to a random oracle but is not a random oracle itself, the combiner may not preserve security. That is, a random-oracle-secure combiner should be instantiated with candidate hash functions and not transformations of candidate hash functions. We go into more detail on this issue in Section 5.

One could consider a stronger definition of hash combiner, which requires indifferentiability from a random oracle when instantiated with a function indifferentiable from a random oracle. We leave it an open question as to whether such a combiner can be achieved, but we suspect that the parameters required for such a transformation are likely to be worse for practitioners.

However, note that this is only an issue when precomposing our definition with indifferentiability. Postcomposing will present no issues aside from those standard for indifferentiability. That is, since our definition requires that the combiner (when instaniated properly) be indifferentiable from a random oracle, Theorem 3.2 still applies. Any one-stage security property secure when instantiated with a random oracle will also be secure when instantiated with a random-oracle-secure combiner as long as one of the underlying hash functions used by the combiner is itself a random oracle.

As most reasonable security properties one would expect from a hash function are indeed designed for one-stage adversaries, a random-oracle-secure combiner can be used as a random oracle for the vast majority of applications. In particular, when instantiated properly, a random-oracle-secure combiner will be collision resistant, one-way, pseudorandom, and second preimage resistant.

4 Random-Oracle-Secure Hash Combiner Construction

With this definition in mind, we present a construction of a hash function combiner which is indifferentiable from a random oracle. Let $h_1, h_2 : \{0,1\}^m \to \{0,1\}^n$ be two hash functions, and let $\mathcal{Z}_1, \mathcal{Z}_2 \xleftarrow{\$} \{0,1\}^k$ for some $k < m$. Then, our construction is simply the xor of the two hashes evaluated on the input concatenated with $\mathcal{Z}_1$, $\mathcal{Z}_2$ respectively. That is, we define our combiner $C_{\mathcal{Z}_1,\mathcal{Z}_2}^{h_1,h_2} : \{0,1\}^\ell \to \{0,1\}^n$ as

$$C_{\mathcal{Z}_1,\mathcal{Z}_2}^{h_1,h_2}(M) = h_1(M, \mathcal{Z}_1) \oplus h_2(M, \mathcal{Z}_2). \tag{1}$$

```
$\underline{REAL - b}$:
Sample $\mathcal{Z} \xleftarrow{\$} \{0,1\}^k$
Sample $h : \{0,1\}^m \to \{0,1\}^n$ u.a.r.
Run $\mathcal{D}^{\mathcal{O}_1,\mathcal{O}_2}(\mathcal{Z}) \to b'$
Output $b'$.

$\mathcal{O}_1(x)$:
Output $h(x)$

$\mathcal{O}_2(M)$:
If $b = 0$, output $C_{\mathcal{Z}}^{\mathcal{O}_1, g^{\mathcal{O}_1}}(M)$.
If $b = 1$, output $C_{\mathcal{Z}}^{g^{\mathcal{O}_1}, \mathcal{O}_1}(M)$.
```

```
$\underline{IDEAL}$:
Sample $\mathcal{Z} \xleftarrow{\$} \{0,1\}^k$
Sample $H : \{0,1\}^\ell \to \{0,1\}^s$ u.a.r.
Run $\mathcal{D}^{\mathcal{O}_1,\mathcal{O}_2}(\mathcal{Z}) \to b'$
Output $b'$.

$\mathcal{O}_1(x)$:
Output $\mathsf{Sim}^{\mathcal{O}_2}(x, \mathcal{Z})$

$\mathcal{O}_2(M)$:
Output $H(M)$.
```

Figure 1: We say that the combiner is secure if for all b, $\mathcal{D}^{\mathcal{O}_1,\mathcal{O}_2}$ not necessarily efficient, $g^{\mathcal{O}}$ a query-bounded circuit, there exists a simulator Sim such that $|\Pr[\mathcal{D}(REAL - b) \to 1] - \Pr[\mathcal{D}(IDEAL) \to 1]| \leq \epsilon$.

This construction relies on k being at least $\frac{m + \log T + \lambda}{2}$, where m is the input length of the hash functions and T is the number of honest hash function h_1 queries the adversarial hash function h_2 may make.

Note that, as m is the length of our hash function input space, this only supports message length at most $\ell = m - k$. In Section 5, we will discuss the difficulties of extending this fixed length to arbitrary length.

Theorem 4.1. *Let $|\mathcal{Z}_1| = |\mathcal{Z}_2| = k$, $h_1, h_2 : \{0,1\}^m \to \{0,1\}^n$. We define a hash combiner $\widetilde{C}_{\mathcal{Z}_1,\mathcal{Z}_2}^{h_1,h_2} : \{0,1\}^\ell \to \{0,1\}^n$ by*

$$\widetilde{C}_{\mathcal{Z}_1,\mathcal{Z}_2}^{h_1,h_2}(M) = h_1(M, \mathcal{Z}_1) \oplus h_2(M, \mathcal{Z}_2).$$

Then, for all T, $\widetilde{C}_{\mathcal{Z}_1,\mathcal{Z}_2}^{h_1,h_2}$ is $\left(T, 1, \frac{T}{2^{k-\ell}}\right)$-random-oracle-secure.

To prove Theorem 4.1, We will rely on the following key lemma.

Lemma 4.2. *Let $g^{\mathcal{O}} : \{0,1\}^m \to \{0,1\}^n$ a circuit making at most T oracle calls. Let $\mathcal{Z}_1, \mathcal{Z}_2 \xleftarrow{\$} \{0,1\}^k$ and $h \xleftarrow{\$} \{f : \{0,1\}^m \to \{0,1\}^n\}$ be random variables. Define BAD to be the event that there exists $x, x' \in \{0,1\}^{\ell := m-k}$ such that $g^h(x, \mathcal{Z}_2)$ queries $h(x', \mathcal{Z}_1)$. Let $\epsilon = \Pr_{\mathcal{Z}_1,\mathcal{Z}_2,h}[BAD]$. Then,*

$$\epsilon \leq \frac{T}{2^{k-\ell}}$$

Proof. We write

$$\epsilon_{\mathcal{Z}_2',h'} := \Pr_{\mathcal{Z}_1,\mathcal{Z}_2,h}[BAD | \mathcal{Z}_2 = \mathcal{Z}_2', h = h'] = \Pr_{\mathcal{Z}_1}[\exists\, x, x' : g^h(x, \mathcal{Z}_2') \text{ queries } h'(x', \mathcal{Z}_1)].$$

By an averaging argument, there must exist some $\mathcal{Z}_2^*, h^*$ such that $\epsilon_{\mathcal{Z}_2^*,h^*} \geq \epsilon$. We will build a compressor $(\mathsf{Enc}, \mathsf{Dec})$ for $\mathcal{Z}_1$ as follows:

We define $\mathsf{Enc}(\mathcal{Z}_1)$:

```
$\underline{Sim^{\mathcal{G}}_{\mathcal{Z}_1,\mathcal{Z}_2}(x)}$:
If $x = (M, \mathcal{Z}_1)$ for some $M$:
-Run $y \leftarrow g^{lazy}(M, \mathcal{Z}_2)$.
-Output $\mathcal{G}(M) \oplus y$.
Else:
-Output $lazy(x)$.

Subroutine $lazy(x)$:
-If $D[x] = \perp$, set $D[x] \xleftarrow{\$} \{0,1\}^n$.
-Output $D[x]$.
```

Figure 2: The simulator.

1. Find the first lexicographic x such that $g^{h^*}(x, \mathcal{Z}_2^*)$ queries $h^*(x', \mathcal{Z}_1)$ for some x'. If no such x exists, output $\perp$. Otherwise, let t be the index that this query occurs at.

2. Output (x, t).

We also define $\mathsf{Dec}(x, t)$:

1. Let $(x', \mathcal{Z})$ be the t-th query made by $g^{h^*}(x', \mathcal{Z}_1)$.

2. Output $\mathcal{Z}$.

Note that as long as Enc doesn't output $\perp$, it is clear that $\mathsf{Dec}(\mathsf{Enc}(\mathcal{Z}_1)) = \mathcal{Z}_1$. But the probability that Enc fails is $\epsilon_{\mathcal{Z}_2^*,h^*} \geq \epsilon$. Thus, Enc compresses a set of size $\epsilon_{\mathcal{Z}_2^*,h^*}2^k$ into a set of size $2^\ell \cdot T$. Theorem 2.2 then gives us

$$\epsilon \leq \epsilon_{\mathcal{Z}_2^*,h^*} \leq \frac{T}{2^{k-\ell}},$$

which completes the proof. □

We now move to the proof of Theorem 4.1. Note that, since our scheme is symmetric, it suffices to prove $\widetilde{C}^{g^{\mathcal{H}},\mathcal{H}}_{\mathcal{Z}}$ is ϵ-indifferentiable from $\mathcal{G}$ for $\epsilon = \frac{T}{2^{k-\ell}}$.

Proof. We first define our simulator Sim as in Figure 2. Let $REAL$ and $IDEAL$ be the games in Figure 3. We simply need to show that for any (unbounded) $\mathcal{D}$, $|\Pr[REAL \to 1] - \Pr[IDEAL \to 1]| \leq \epsilon$.

To achieve this, we rely on a series of hybrid games, $REAL, G1, G2, G3, G4, IDEAL$ defined in Figures 3, 4, and 5.

Claim 1: $\Pr[REAL \to 1] = \Pr[G1 \to 1]$.
In G1, since f and h are sampled uniformly at random, $\widetilde{h}$ is a uniformly random function. Thus, since the only difference between these two games is that h (a uniformly random function) is replaced with $\widetilde{h}$, $REAL$ and $G1$ are identically distributed.

Claim 2: $|\Pr[G1 \to 1] - \Pr[G2 \to 1]| \leq \epsilon$.
Conditioned on the event that there does not exist any x such that $g^h(x, \mathcal{Z}_2)$ queries h on $(M, \mathcal{Z}_2)$, $G2$ is identically distributed to $G1$. But by Lemma 4.2, this occurs with probability at most ϵ, and so the claim follows.

Claim 3: $\Pr[G2 \to 1] = \Pr[G3 \to 1]$.
Note that the function $g^h(x, \mathcal{Z}_2)$ is independent of f, and so as f is uniformly random, $F(x) = f(x) \oplus g^h(x, \mathcal{Z}_2)$ is also uniformly random. Thus, since the only difference between $G3$ and $G2$ is that f is replaced by F, the two games are identically distributed.

Claim 4: $\Pr[G3 \to 1] = \Pr[G4 \to 1]$.
Note that $G4$ is just $G3$ with algebraic terms reorganized. All values returned by all oracles are identically distributed in both games. The claim trivially follows.

Claim 5: $\Pr[IDEAL \to 1] = \Pr[G4 \to 1]$.
The only difference between $IDEAL$ and $G4$ is that h is replaced by lazy sampling. The claim trivially follows.

Putting the above claims together, we get that $\Pr[REAL \to 1] = \Pr[G1 \to 1]$ and $\Pr[G2 \to 1] = \Pr[IDEAL \to 1]$, and so

$$|\Pr[REAL \to 1] - \Pr[IDEAL \to 1]| \leq \epsilon$$

$\square$

4.1 Remarks on Parameters

Given two $\{0,1\}^m \to \{0,1\}^n$ hash functions, our construction gives a $\{0,1\}^\ell \to \{0,1\}^n$ hash function with security loss $\frac{T}{2^{k-\ell}}$. Note that here, $\ell + k = m$, and so this security loss can be rewritten as $\frac{T}{2^{2k-m}}$. In practice, hash functions are expected to have so called "birthday bound" security for various security properties (such as collision resistance). That is, any attack making q queries to the hash function should have probability of success $\frac{q}{2^\lambda}$, where λ is the security parameter.

As our notion of indifferentiability applies to unbounded attackers, the security loss from our construction does not at all depend on the number of queries made by the distinguisher. Thus, to achieve birthday security using our construction, it is sufficient to set

$$k = \frac{m + \log T + \lambda}{2}.$$

Of course, this means our input space is limited to length $\ell = (m - \log T - \lambda)/2$.

```
REAL:
Sample $Z_1, Z_2 \overset{\$}{\leftarrow} \{0,1\}^k$
Sample $h : \{0,1\}^m \to \{0,1\}^n$ u.a.r.
Run $\mathcal{D}^{\mathcal{O}_1,\mathcal{O}_2}(\mathcal{Z}_1, \mathcal{Z}_2) \to b'$
Output $b'$.

$\mathcal{O}_1(x)$:
Output $h(x)$

$\mathcal{O}_2(M)$:
Output $\mathcal{O}_1(M, \mathcal{Z}_1) \oplus g^{\mathcal{O}_1}(M, \mathcal{Z}_2)$.
```

(a) Real Game, where $\mathcal{O}_1$ refers to the random oracle and $\mathcal{O}_2$ refers to our combiner construction.

```
IDEAL:
Sample $Z_1, Z_2 \overset{\$}{\leftarrow} \{0,1\}^k$
Sample $H : \{0,1\}^\ell \to \{0,1\}^n$ u.a.r.
Run $\mathcal{D}^{\mathcal{O}_1,\mathcal{O}_2}(\mathcal{Z}_1, \mathcal{Z}_2) \to b'$
Output $b'$.

$\mathcal{O}_1(x)$:
If $x = (M, \mathcal{Z}_1)$:
- Compute $y = g^{lazy}(M, \mathcal{Z}_2)$. If $g$ ever queries $(M', \mathcal{Z}_1)$ for any $M'$, fail.
- Output $\mathcal{O}_2(M) + y$
Else: output $lazy(x)$

$\mathcal{O}_2(M)$:
Output $F(M)$.

Subroutine $lazy(M)$:
If $D[x] = \perp$: $D[x] \overset{\$}{\leftarrow} \{0,1\}^n$.
Output $D[x]$.
```

(b) Ideal Game, where $\mathcal{O}_1$ refers to our h_1 simulator and $\mathcal{O}_2$ refers to our idealized combiner.

Figure 3: The real and ideal games for Theorem 4.1.

```
G1:
Sample $Z_1, Z_2 \xleftarrow{\$} \{0,1\}^k$
Sample $h : \{0,1\}^m \to \{0,1\}^n$ u.a.r.
Sample $f : \{0,1\}^\ell \to \{0,1\}^n$ u.a.r.
Define $\widetilde{h}(x) = \begin{cases} f(M) & x = (M, \mathcal{Z}_1) \text{ for some } M \\ h(x) & o.w. \end{cases}$
Run $\mathcal{D}^{\mathcal{O}_1,\mathcal{O}_2}(\mathcal{Z}_1, \mathcal{Z}_2) \to b'$
Output $b'$.

$\mathcal{O}_1(x)$:
Output $\widetilde{h}(x)$

$\mathcal{O}_2(M)$:
Output $\widetilde{h}(M, \mathcal{Z}_1) \oplus g^{\widetilde{h}}(M, \mathcal{Z}_2)$.
```

The same as the real game, but with the honest hash function divided into two parts.

```
G2:
Sample $Z_1, Z_2 \xleftarrow{\$} \{0,1\}^k$
Sample $h : \{0,1\}^m \to \{0,1\}^n$ u.a.r.
Sample $f : \{0,1\}^\ell \to \{0,1\}^n$ u.a.r.
Define $\widetilde{h}(x) = \begin{cases} f(M) & x = (M, \mathcal{Z}_1) \text{ for some } M \\ h(x) & o.w. \end{cases}$
Run $\mathcal{D}^{\mathcal{O}_1,\mathcal{O}_2}(\mathcal{Z}_1, \mathcal{Z}_2) \to b'$
Output $b'$.

$\mathcal{O}_1(x)$:
Output $\widetilde{h}(x)$

$\mathcal{O}_2(M)$:
Output $f(M) \oplus g^h(M, \mathcal{Z}_2)$.
```

The same as G1, but with $g^{\widetilde{h}}$ replaced with g^h. That is, the compromised hash function's queries to $(M, \mathcal{Z}_1)$ are replaced with random values.

Figure 4: The first two hybrids used for Theorem 4.1.

```
G3:
Sample $Z_1, Z_2 \xleftarrow{\$} \{0,1\}^k$
Sample $h : \{0,1\}^m \to \{0,1\}^n$ u.a.r.
Sample $f : \{0,1\}^\ell \to \{0,1\}^n$ u.a.r.
Define $F(x) := f(x) \oplus g^h(x, \mathcal{Z}_2)$.
Define $\widetilde{h}(x) = \begin{cases} F(M) & x = (M, \mathcal{Z}_1) \text{ for some } M \\ h(x) & o.w. \end{cases}$
Run $\mathcal{D}^{\mathcal{O}_1,\mathcal{O}_2}(\mathcal{Z}_1, \mathcal{Z}_2) \to b'$
Output $b'$.

$\mathcal{O}_1(x)$:
Output $\widetilde{h}(x)$

$\mathcal{O}_2(M)$:
Output $F(M) \oplus g^h(M, \mathcal{Z}_2)$.
```

The same as G2, but with $f(x)$ replaced by $f(x) \oplus g^h(x, \mathcal{Z}_2)$. That is, the honest hash functions outputs are shifted by $g^h(x, \mathcal{Z}_2)$.

```
G4:
Sample $Z_1, Z_2 \xleftarrow{\$} \{0,1\}^k$
Sample $h : \{0,1\}^m \to \{0,1\}^n$ u.a.r.
Sample $f : \{0,1\}^\ell \to \{0,1\}^n$ u.a.r.
Run $\mathcal{D}^{\mathcal{O}_1,\mathcal{O}_2}(\mathcal{Z}_1, \mathcal{Z}_2) \to b'$
Output $b'$.

$\mathcal{O}_1(x)$:
If $x = (M, \mathcal{Z}_1)$, output $\mathcal{O}_2(M) \oplus g^h(M, \mathcal{Z}_2)$.
Else: output $h(x)$.

$\mathcal{O}_2(M)$:
Output $f(M)$.
```

The same as G3, but with values reorganized so as to be similar to the simulator.

Figure 5: The last two hybrids used for Theorem 4.1.

4.2 Our Security Proof is Tight

Note that although our proof achieves birthday security for $k = \frac{m+\log T+\lambda}{2}$, one may wonder whether Theorem 4.1 holds even for smaller values of k. That is, are the parameters required by our security proof optimal for our construction? In this section, we show that the parameters are optimal up to a constant factor.

Theorem 4.3. *Let $k = \ell$ and let $\widetilde{C}^{h_1,h_2}_{\mathcal{Z}_1,\mathcal{Z}_2} : \{0,1\}^\ell \to \{0,1\}^n$ be as in Equation (1). Then $\widetilde{C}$ is not $\left(1, 1 - \frac{1}{2^n}\right)$-random oracle secure.*

Proof. Define $g^{\mathcal{H}}(x,y) := h(x,x)$. Then observe that

$$\widetilde{C}^{h,g^h}_{\mathcal{Z}_1,\mathcal{Z}_2}(\mathcal{Z}_1) = h(\mathcal{Z}_1,\mathcal{Z}_1) + g^h(\mathcal{Z}_1,\mathcal{Z}_2) = 0.$$

Thus, we can distinguish $\widetilde{C}$ from a random oracle simply by evaluating on $\mathcal{Z}_1$. This will succeed with probability

$$\left|\Pr[\widetilde{C}(\mathcal{Z}_1) = 0] - \Pr[\mathcal{H}(\mathcal{Z}_1) = 0]\right| = 1 - \frac{1}{2^n}$$

□

In particular, this means that our proof of security for our construction is close to tight. That is, our construction admits an attack when $k \leq \ell$. Since our proof requires that $k \geq \ell + \log T + \lambda$, this means that our randomness requirement is tight up to a $\log T + \lambda$ additive factor. If $\ell = c\lambda$ and $T \leq 2^\lambda$, our proof of security will be tight for our construction up to a $(c+2)$ multiplicative factor.

4.3 Random-Oracle-Secure Combiners Need Randomness

Our construction requires randomness linear in m and the security parameter λ. We now show the dependence on λ is necessary.

In particular, we first prove that no deterministic random-oracle-secure hash combiners can exist — randomness is always needed. This contrasts with the notion of combiners for collision resistance, in which setting concatenation functions as an effective deterministic combiner. We further demonstrate a lower bound on the amount of randomness required for any hash combiner to be random-oracle-secure, and we argue that in order to achieve "brute-force security", λ bits of randomness are required.

First, we present our main theorem, showing that random-oracle-secure combiners require some randomness.

Theorem 4.4. *Let C^{h_1,h_2} be a <u>deterministic</u> hash combiner making at most T_C queries to its underlying hash function. Then, for any value of T_{Sim}, C is not $(7T_C, T_{\mathsf{Sim}}, 0.25)$-random oracle secure.*

Theorem 4.5. *Let $C^{h_1,h_2}_{\mathcal{Z}}$ be a hash combiner making T_C oracle queries and utilizing k bits of randomness (i.e. $|\mathcal{Z}| = k$). Then, for any value of T_{Sim}, C is not $\left(7T_C, T_{\mathsf{Sim}}, \frac{0.25}{2^k}\right)$-secure, even when only considering attacks which can be generated in time $O(2^k)$.*

Theorem 4.5 means that any combiner must use superlogarithmic randomness in order to achieve negligible security loss. In practice, this means that if one hopes to achieve "brute-force" security, (i.e. no attacker should be able to succeed with probability $\geq \frac{1}{2^\lambda}$), then it is necessary that $k \geq \lambda - 2$. That is, there should be at least as much randomness used by the combiner as desired bits of security.

In particular, we note that the adversarial hash functions used in the proofs of these theorems can be produced in time $O(1)$ and $O((2^k)^2)$ respectively. The distinguisher in both theorems runs in time $O(1)$.

The proofs of both theorems are a direct result of the following lemma:

Lemma 4.6. *Let $C_{\mathcal{Z}}^{h_1,h_2} : \{0,1\}^\ell \to \{0,1\}^s$ be a hash combiner with $|\mathcal{Z}| = k$ making at most T_C queries to its oracles. Let $\epsilon > 0$, $\rho \in (0,1)$, $d \in \mathbb{N}$ such that*

$$\rho\left(1 - \left(\frac{1}{2}+\epsilon\right)^d\right) - \left(\frac{1}{2}+\epsilon\right) > 0$$

Then, C is not $(dT_C, \infty, \frac{\epsilon}{2^k})$-random oracle secure.

In particular, Theorem 4.5 results from setting $\rho = 0.9$, $\epsilon = 0.25$, and $d = 7$. Theorem 4.4 results from setting $k = 1$ in Theorem 4.5.

4.4 Proof of Lemma 4.6

Claim 1. *let $C^{h_1,h_2} : \{0,1\}^\ell \to \{0,1\}^s$ be any deterministic oracle function making at most T_C queries to its oracles. Let $\mathcal{F} = \{f : \{0,1\}^m \to \{0,1\}^n\}$. Suppose that*

$$\Pr_{h_1 \xleftarrow{\$} \mathcal{F}}\left[\Pr_{h_2 \xleftarrow{\$} \mathcal{F}}[C^{h_1,h_2}(0) = 0] \geq \frac{1}{2} - \epsilon\right] \geq \rho.$$

Then, for all d, there exists an $\widetilde{h}_r^{h_1}$ making at most $d \cdot T_C$ queries to h_1 such that

$$\Pr_r[\Pr_{h_1}[C^{h_1,\widetilde{h}_r^{h_1}}(0) = 0] \geq \frac{1}{2} + \epsilon] \geq \rho\left(1 - \left(\frac{1}{2}+\epsilon\right)^d\right) - \left(\frac{1}{2}+\epsilon\right)$$

Proof. We will first define a stateful algorithm $lazy_r(x)$:
On initialization, $lazy_r$ sets $D = \{\}$, $ctr = 1$, and parses $r = w_1, \ldots, w_{T_C}$.
On input x, $lazy_r(x)$ does the following:
- If $x \notin D$, set $D[x] = w_{ctr}$ and increase ctr by 1.
- Output $D[x]$.

We will define $\widetilde{h}_r^{h_1}(x)$ as follows:
-Parse $r = r_1, \ldots, r_d$.
-For each $i = 1, \ldots, d$: run $C^{h_1, lazy_{r_i}}(0) \to c_i$, keeping track of D_i the database computed by $lazy_{r_i}$.
-If $c_i = 0$ for any i, set i^* to be the smallest such i.
-Output $D_{i^*}[x]$ if $x \in D_{i^*}$. Otherwise output 0.

We make two key observations. First note that if $\widetilde{h}_r$ finds i^* for any (and thus all) of its inputs, then $C^{h_1, lazy_{r_{i^*}}}(0) = C^{\widetilde{h}_r, h_2}(0) = 0$. Second, note that for uniform w, $C^{h_1, lazy_w}(0)$ is identically distributed to $C^{h_1,h_2}(0)$.
Let h_1 be such that $\Pr_{h_2}[C^{h_1,h_2}(0) = 0] \geq \left(\frac{1}{2} - \epsilon\right)$. Then

$$\Pr_w[C^{h_1, lazy_w}(0) = 0] \geq \left(\frac{1}{2} - \epsilon\right)$$

$$\Pr_{w_1,\dots,w_d}[C^{h_1, lazy_w}(0) = 0 \text{ for some } i] \geq 1 - \left(\frac{1}{2} + \epsilon\right)^d$$

$$\Pr_r[C^{h_1, \widetilde{h}_r^{h_1}}(0) = 0] \geq 1 - \left(\frac{1}{2} + \epsilon\right)^d$$

Putting this together with the assumption in the lemma, we get

$$\Pr_{h_1}[\Pr_r[C^{h_1, \widetilde{h}_r}(0) = 0] \geq 1 - \left(\frac{1}{2} + \epsilon\right)^d] \geq \rho$$

and so

$$\Pr_{r,h_1}[C^{h_1, \widetilde{h}_r}(0) = 0] \geq \rho\left(1 - \left(\frac{1}{2} + \epsilon\right)^d\right)$$

Using basic probability, we get our lemma. □

Claim 2. *Let C^{h_1,h_2} be such that*

$$\Pr_{h_1 \xleftarrow{\$} \mathcal{F}}[\Pr_{h_2 \xleftarrow{\$} \mathcal{F}}[C^{h_1,h_2}(0) = 0] \geq \frac{1}{2} - \epsilon] < \rho$$

Then there exists an efficient $\widetilde{h}_r^{h_2}$ making at most T_C queries to h_2 such that

$$\Pr_r[\Pr_{h_2}[C^{\overline{h}_r, h_2}(0) = 1] \geq \frac{1}{2} + \epsilon] \geq 1 - \rho$$

Proof. We simply define $\overline{h}_r$ to lazily sample the random oracle. In particular, $\overline{h}_r(x)$ is defined by:
-Run $C^{lazy_r, h_2}(0)$ to get database D.
-If $x \in D$, output $D[x]$, otherwise output 0.

It is clear that $C^{lazy_r, h_2}(0)$ is identically distributed to $C^{h_1,h_2}(0)$, so we conclude

$$\Pr_r[\Pr_{h_2}[C^{\overline{h}_r, h_2}(0) = 0] \geq \frac{1}{2} - \epsilon] < \rho.$$

□

The proof of Lemma 4.6 comes from the two claims as follows:

Proof. Let ϵ, ρ, and d be as in the theorem statement. We will first consider the case where $\mathcal{Z} = 0$. By the claims, when ρ and d are set appropriately, either

$$\Pr_r[\Pr_{h_1}[C_0^{h_1, \widetilde{h}_r}(0) = 0] \geq \frac{1}{2} + \epsilon] > 0$$

or

$$\Pr_r[\Pr_{h_2}[C_0^{\overline{h_1}_r,h_2}(0)=1]\geq\frac{1}{2}+\epsilon]>0$$

We will define an attack for the first case, and the attack in the second case will be symmetric.

Observe that in the first case, there must exist some r^* such that $\Pr_{h_1}[C_0^{h_1,\widetilde{h}_{r^*}}(0)=0]\geq\frac{1}{2}+\epsilon$. Thus, we define $g=\widetilde{h}_{r^*}$. For $\mathcal{H},\mathcal{G}$ random oracles, We will develop an attacker $\mathcal{D}$ for the indifferentiability game between $C_{\mathcal{Z}}^{\mathcal{H},g^{\mathcal{H}}}$ from $\mathcal{G}$.

$\mathcal{D}(\mathcal{Z},\mathcal{O}_1,\mathcal{O}_2)$:
-If $\mathcal{Z}\neq 0$: flip a coin $b\xleftarrow{\$}\{0,1\}$ and output b.
-If $\mathcal{Z}=0$: output 1 if and only if $\mathcal{O}_2(0)$.

We can then compute the advantage of $\mathcal{D}$ as follows:

$$\begin{aligned}
&\Pr[\mathcal{D}(\mathcal{H},C_{\mathcal{Z}}^{\mathcal{H},g^{\mathcal{H}}})=1]\\
&=\Pr[\mathcal{Z}=0]\Pr[\mathcal{D}(C_{\mathcal{Z}}^{\mathcal{H},g^{\mathcal{H}}}=0|\mathcal{Z}=0]+\Pr[\mathcal{Z}\neq 0]\Pr[\mathcal{D}(C_{\mathcal{Z}}^{\mathcal{H},g^{\mathcal{H}}}=0|\mathcal{Z}\neq 0]\\
&=\frac{1}{2^k}\Pr_{h_1}[C_0^{h_1,\widetilde{h}_{r^*}^{h_1}}(0)=0]+\left(1-\frac{1}{2^k}\right)\frac{1}{2}\\
&\geq\frac{1}{2}+\frac{1}{2^k}\left(\left(\frac{1}{2}+\epsilon\right)-\frac{1}{2}\right)=\frac{1}{2}+\frac{\epsilon}{2^k}
\end{aligned}$$

But for any simulator Sim,

$$\Pr[\mathcal{D}(\mathsf{Sim}^{\mathcal{G}},\mathcal{G})=1]=\frac{1}{2}$$

and so $C_{\mathcal{Z}}^{\mathcal{H},g^{\mathcal{H}}}$ is not $\frac{1}{2^k}(1-\left(\frac{1}{2}-\epsilon\right))$-indifferentiable from $\mathcal{G}$. Thus, since in either case g will make at most dT_C queries to its oracle, we get that C^{h_1,h_2} is not $\left(dT_C,\frac{\epsilon}{2^k}\right)$-random oracle secure.

□

A few remarks on efficiency: We remark that the probability of a random r being sufficient for our purposes is

$$\alpha:=\min\left(1-\rho,\rho\left(1-\left(\frac{1}{2}+\epsilon\right)^d\right)-\left(\frac{1}{2}+\epsilon\right)\right)$$

and we can test whether some r is sufficient by estimating

$$\Pr_{h_1}[C_0^{h_1,\widetilde{h}_r}(0)=0]=\Pr_{r'}[C_0^{lazy_{r'},\widetilde{h}_r}(0)=0]$$

or

$$\Pr_{h_2}[C_0^{\overline{h}_r,h_2}(0)=0]=\Pr_{r'}[C_0^{\overline{h}_r,lazy_{r'}}(0)=0]$$

and seeing if it is larger than $\frac{1}{2}-\frac{1}{2}\frac{\epsilon}{2^k}$. This test can be performed in time $O\left(\left(\frac{2^k}{\epsilon}\right)^2\right)$, and a good r can be found in $O\left(\frac{1}{\alpha}\right)$ trials. Thus, we can produce a g which biases C in time $O\left(\frac{1}{\alpha}\left(\frac{2^k}{\epsilon}\right)^2\right)$.

5 Extensions to Arbitrary Length

In Sections 3 and 4, we considered a regime where the two input hash functions go from m-bit long strings to n-bit long strings.

Note that there are a number of constructions [7], mostly designed around the Merkle-Damgård transform (Section 5.1), which when instantiated on a compressing random oracle, are indifferentiable from a random oracle. In Section 5.2, we show how to compose such a construction with a combiner to produce a combiner for arbitrary length input. However, in Section 5.3, we see that the parameters of our construction imply that it is not advisable to employ this approach for hash functions used in practice.

One would hope to instead perform a transformation from [7] on the original compression function, and then applying our combiner to the results. However, as we remarked in Section 3.1, this construction may not be secure. In fact, in Section 5.4, we demonstrate an attack against this approach.

To remedy this issue, in Section 6, we explicitly consider the security of precomposing with a Merkle-Damgård transformation instead of relying on the modular approach. In particular, we show that this approach produces a combiner satisfying collision-resistance. We conjecture that indifferentiability will hold for this construction as well.

5.1 The Merkle-Damgård Transformation

First, we define some of the terms necessary to understand the Merkle-Damgård transformation on hash functions. For the remainder of this work, for any hash function h, we will denote by h^* the Merkle-Damgård transformation applied to that hash function.

Definition 5.1. *(Merkle-Damgård transformation): Let $f : \{0,1\}^{n+\Delta} \to \{0,1\}^n$ be a function. We define $f^* : \{0,1\}^* \to \{0,1\}^n$ as follows:*
On input x, write $x = (x_1, \ldots, x_\ell)$ where $|x_i| = \Delta$ (if $|x|$ is not a multiple of Δ, pad with 0s). Then,

$$f^*(x) = f(x_\ell, \ldots, f(x_2, f(x_1, 0)) \ldots)$$

Intuitively, the Merkle-Damgård transform separates the message into Δ-bit blocks and then iteratively evaluates a fixed-length hash function $h : \{0,1\}^{n+\Delta} \to \{0,1\}^n$ on each block in order to extend the length of input arbitrarily.

At times, we will want to consider partial evaluations of the Merkle-Damgård evaluation. In these events, for any hash function $h : \{0,1\}^{n+\Delta} \to \{0,1\}^n$, any message $x = (x_1, \ldots, x_\ell) \in \{0,1\}^{\ell\Delta}$, and any $1 \le i \le \ell$, we denote

$$h^{(i)}(x) := h^*(x_1, \ldots, x_i).$$

5.2 Arbitrary Length Construction

For any hash function transformation from fixed length to arbitrary length which preserves indifferentiability, we can apply this to our fixed-length random-oracle-secure combiner (Equation (1)) to produce an arbitrary length random-oracle-secure combiner. In fact, this holds for any random-oracle-secure combiner, as seen in the following theorem. Note that the parameters for indifferentiability will depend heavily on the particular construction used, and so we will only provide an informal theorem. The proof is a straightforward application of Theorem 3.2.

Theorem 5.2. *(Informal) Let $\mathcal{F} : \{0,1\}^m \to \{0,1\}^n$ and $\mathcal{G} : \{0,1\}^* \to \{0,1\}^{n'}$ be random oracles, and let $T^{\mathcal{F}}$ be indifferentiable from $\mathcal{G}$. Let $C_{\mathcal{Z}}^{h_1,h_2}$ be a random-oracle-secure combiner. Then $\overline{C}_{\mathcal{Z}}^{h_1,h_2}$ defined by*

$$\overline{C}_{\mathcal{Z}}^{h_1,h_2}(M) = T^{C_{\mathcal{Z}}^{h_1,h_2}}(M)$$

is a random-oracle-secure combiner.

Plugging in the NMAC transformation construction from [7] gives us the following corollary.

Corollary 5.3. *Let $\widetilde{C}$ be as in Theorem 4.1. Then*

$$ArbC_{\mathcal{Z}}^{h_1,h_2}(M) := \widetilde{C}(1, \widetilde{C}(0, M_\ell, \widetilde{C}(0, M_{\ell-1}, \widetilde{C}(\dots, \widetilde{C}(0, M_1, 0)\dots))))$$

is a random-oracle-secure combiner.

5.3 A Note on Parameters

Note that in order for our random-oracle-secure combiner construction to be a compression function for use in Theorem 5.2, it is necessary that $\ell > n$. But as $\ell \le m/2$, this means that the underlying hash functions must satisfy $n \le m/2$.

In practice, the compression function used for SHA2 maps 768 bits (representing the internal state and the input block) to 256 bits (representing just the internal state) [1]. Thus, for SHA2, $m/2 = \frac{768-256}{2} = 256 = n$, and so the resulting combiner is not compressing. The compression function used for SHA3 maps 2688 bits to 1600 bits [10]. Thus, our combiner cannot be instantiated with either of these hash functions without first truncating the output. However, truncation will reduce security guarantees significantly, and so is not recommended.

5.4 A Prefix Attack on Merkle-Damgård Hash Functions

In order to produce a random oracle with sufficient compression, we could imagine applying Merkle-Damgård to a concrete hash function. Thus, the resulting hash function would be indifferentiable from a random oracle, and will be sufficiently compressing. Unfortunately, we show that if we use the random-oracle-combiner as a black-box, the resulting construction may not be secure.

In particular, applying a combiner on a hash function indifferentiable from a random oracle will not necessarily preserve security.

Define

$$PreC_{\mathcal{Z}_1,\mathcal{Z}_2}^{h_1,h_2}(M) := h_1(\mathcal{Z}_1, M) \oplus h_2(\mathcal{Z}_2, M)$$

Note that the same proof as Theorem 4.1 shows that $PreC$ is $\left(T, 1, \frac{T}{2^{k-\ell}} + \frac{1}{2^k}\right)$-random-oracle-secure.

Furthermore, [7] shows that if $\mathcal{H} : \{0,1\}^m \to \{0,1\}^n$ is a random oracle, then for any fixed m', $\mathcal{H}^* : \{0,1\}^{m'} \to \{0,1\}^n$ is a random oracle.

However, we will show that composing these two constructions does not lead to a random-oracle-secure combiner. In particular, the resulting hash combiner will not even be a one-way function.

Theorem 5.4. *Let $h_1, h_2 : \{0,1\}^{2n} \to \{0,1\}^n$.*
Define

$$PreCMD_{\mathcal{Z}_1,\mathcal{Z}_2}^{h_1,h_2}(M) := h_1^*(\mathcal{Z}_1, M) \oplus h_2^*(\mathcal{Z}_2, M).$$

There exists an adversarial implementation $g^{\mathcal{O}}$ such that, if $y = PreCMD^{h,g^h}_{\mathcal{Z}_1,\mathcal{Z}_2}(x)$, there exists an efficient A such that

$$\Pr_{\substack{h \xleftarrow{\$} \{f:\{0,1\}^m \to \{0,1\}^n\} \\ \mathcal{Z}_1,\mathcal{Z}_2 \xleftarrow{\$} \{0,1\}^k, x \xleftarrow{\$} \{0,1\}^\ell}} [PreCMD^{h,g^h}_{\mathcal{Z}_1,\mathcal{Z}_2}(A(y)) = y] \geq 1 - \frac{3}{2^n}.$$

We immediately get the following corollary.

Corollary 5.5. *PreCMD is not $\left(1, T_{\mathsf{Sim}}, 1 - \frac{4}{2^n}\right)$-random-oracle-secure for any value of T_{Sim}.*

Proof. (of Theorem 5.4) Define the compromised hash function g^h as follows

$$g^h(x, IV) := \begin{cases} 1 & x = 0 \\ h(x,x) & IV = 1 \\ h(x, IV) \oplus x & \text{else} \end{cases}$$

Let $y \neq 0$. Define $M_y = 0|h^*(\mathcal{Z}_1, 0)|y$. We will show that with high probability over h, that if $g^*(\mathcal{Z}_2, M_y) = h^*(\mathcal{Z}_1, y)$.
We call h good if $h^*(\mathcal{Z}_1, 0) \neq 0$ and $h^*(\mathcal{Z}_1, 0, h^*(\mathcal{Z}_1, 0)) \neq 1$. In this case,

$$\begin{aligned} g^*(\mathcal{Z}_2, M_y) &= g(y, g(h^*(\mathcal{Z}_1, 0), g(0, g^*(\mathcal{Z}_2)))) \\ &= g(y, g(h^*(\mathcal{Z}_1, 0), 1)) \\ &= g(y, h(h^*(\mathcal{Z}_1, 0), h^*(\mathcal{Z}_1, 0))) \\ &= g(y, h^*(\mathcal{Z}_1, 0, h^*(\mathcal{Z}_1, 0))) \\ &= h(y, h^*(\mathcal{Z}_1, 0, h^*(\mathcal{Z}_1, 0))) \oplus y \\ &= h^*(\mathcal{Z}_1, 0, h^*(\mathcal{Z}_1, 0), y) \oplus y \\ &= h^*(\mathcal{Z}_1, M_y) \oplus y \end{aligned}$$

and so

$$PreCMD^{h,g^h}_{\mathcal{Z}_1,\mathcal{Z}_2}(M_y) = h^*(\mathcal{Z}_1, M_y) \oplus h^*(\mathcal{Z}_1, M_y) \oplus y = y$$

Define $A(y) = M_y$. For simplicity, we will write $y = PreCMD^{h,g^h}_{\mathcal{Z}_1,\mathcal{Z}_2}(x)$. Then,

$$\begin{aligned} &\Pr_{h,\mathcal{Z}_1,\mathcal{Z}_2,x}[PreCMD^{h,g^h}_{\mathcal{Z}_1,\mathcal{Z}_2}(A(y)) = y] \\ &= \Pr[y \neq 0, h^*(\mathcal{Z}_1, 0) \neq 0, h^*(\mathcal{Z}_1, 0, h^*(\mathcal{Z}_1, 0)) \neq 1] \geq 1 - \frac{3}{2^n} \end{aligned}$$

□

6 A Short Collision Resistant Hash Function Combiner

Despite the fact that indifferentiability does not precompose with random-oracle-security, we may hope that our construction is secure when instantiated specifically with Merkle-Damgård style hash functions. In particular, define

$$CMD^{h_1,h_2}_{\mathcal{Z}_1,\mathcal{Z}_2}(M) = h_1^*(M, \mathcal{Z}_1) \oplus h_2^*(M, \mathcal{Z}_2).$$

We would like to claim that $CMD^{h_1,h_2}_{\mathcal{Z}_1,\mathcal{Z}_2}$ is random-oracle-secure.

As evidence towards this result, we show that $CMD^{h_1,h_2}_{\mathcal{Z}_1,\mathcal{Z}_2}$ is collision-resistant. We leave the claim that $CMD^{h_1,h_2}_{\mathcal{Z}_1,\mathcal{Z}_2}$ is random-oracle-secure as an open question.

6.1 Collision Resistance Definitions

We present the definitions of collision resistance as well as a collision-resistant hash function combiner in the random oracle model.

Definition 6.1. *We say that a family of hash functions $\{H^{\mathcal{O}}_{\mathcal{Z}}\}_{\mathcal{Z}}$ is (T_A, ϵ)-collision-resistant in the random oracle model if for all $A^{\mathcal{O}}$ making at most T_A queries to $\mathcal{O}$*

$$\Pr_{\mathcal{Z}}[A^{\mathcal{O}}(\mathcal{Z}) \to M_0, M_1 : H^{\mathcal{O}}_{\mathcal{Z}}(M_0) = H^{\mathcal{O}}_{\mathcal{Z}}(M_1)] \leq \epsilon$$

Definition 6.2. *Let $C^{h_1,h_2}_{\mathcal{Z}}$ be a combiner. Let $\mathcal{H}, \mathcal{G}$ be two random oracles. We say that C is $\underline{(T_g, T_A, \epsilon)\text{-collision-resistant secure}}$ if, for all oracle circuits $g^{\mathcal{O}}$ making at most T_g oracle queries, both $C^{\mathcal{H},g^{\mathcal{H}}}_{\mathcal{Z}}$ and $C^{g^{\mathcal{H}},\mathcal{H}}_{\mathcal{Z}}$ are (T_A, ϵ)-collision-resistant in the random oracle model.*

6.2 Merkle-Damgård Yields Collision Resistance

We present the main theorem for the section, and the rest will be dedicated to proving this result.

Theorem 6.3. *Let $h_1 : \{0,1\}^{n+\Delta} \to \{0,1\}^n$ be a random oracle, and let $h_2 : \{0,1\}^{n+\Delta} \to \{0,1\}^n$ be an oracle circuit with at most T_{h_2} gates to h_1. Let $\mathcal{Z}_1, \mathcal{Z}_2 \xleftarrow{\$} \{0,1\}^{k\Delta}$. Define $CMD^{h_1,h_2}_{\mathcal{Z}_1,\mathcal{Z}_2} : \{0,1\}^{\ell\Delta} \to \{0,1\}^n$ as*

$$CMD^{h_1,h_2}_{\mathcal{Z}_1,\mathcal{Z}_2}(M) = h_1^*(M, \mathcal{Z}_1) \oplus h_2^*(M, \mathcal{Z}_2).$$

Then, $CMD^{h_1,h_2}_{\mathcal{Z}_1,\mathcal{Z}_2}$ is $(T_{h_2}, T_{\mathcal{A}}, \epsilon)$-collision-resistant secure for

$$\epsilon \leq \frac{T_g^{k/\Delta}}{2^{k-\ell-1}} + \frac{2T_{\mathcal{A}} + 4(\ell+k) + 5}{2^n} + \frac{3}{(T_{\mathcal{A}} + 2(\ell+k)) \cdot 2^n}$$

Note that, in particular, the output length of CMD significantly bypasses the impossibility results of [4, 21, 22].

6.3 Proof of Theorem 6.3

We now prove Theorem 6.3. The proof is rather technical, but the high-level intuition is as follows: If the combiner were not collision-resistant, then with noticeable probability we can find $M_0 \neq M_1$ such that $h_2^*(M_0, \mathcal{Z}_2) \oplus h_2^*(M_1, \mathcal{Z}_2) = h_1^*(M_0, \mathcal{Z}_1) \oplus h_1^*(M_1, \mathcal{Z}_1)$. Then, we see that either one or both of the h_2^* evaluations must query the final round of an h_1^* evaluation, or neither will.

1. If one does, then with high probability it must query every intermediate value for computing h_1^* (Lemma 6.4), and we may use these intermediate values with these queries to compress the random oracle.

```
reconstruct($h_{target}, \vec{q}$):
Let $i_1, \ldots, i_\ell$ be an increasing sequence of indices of $\vec{q}$ corresponding to queries $q_{i_1} = (s_1, IV_1), \ldots, q_{i_\ell} = (s_\ell, IV_\ell)$ satisyfing:
- $IV_1 = 0^n$,
- $\forall i < \ell, IV_{i+1} = h_1(s_i, IV_i)$.
- $h_{target} = h_1(s_\ell, IV_\ell)$. If this sequence is unique in $\vec{q}$, output $c_1 \circ c_2 \circ \ldots \circ c_\ell$.
Else, output $\perp$.
```

Figure 6: The algorithm reconstruct.

2. If neither does, then a similar process may be used to create an algorithm which outputs a random oracle output without querying it, which obviously succeeds only with negligible probability.

So, in either case, we see that a noticeable probability of success at breaking collision resistance corresponds to a noticeable probability of either (1) compressing the random oracle, or (2) learning the random oracle without querying it. Since both of these things cannot succeed with noticeable probability, we are done.

As a note, all notions of h_2 in the theorem statement and proof are h_1 oracle circuits. We denote $h_2 = h_2^{h_1}$ for simplicity.

Proof. Suppose toward contradiction that there exists some $h_2, \mathcal{A}$ which break collision resistance. In particular, $\mathcal{A}(\mathcal{Z}_1, \mathcal{Z}_2)$ outputs some $M_0 \neq M_1$ such that $CMD^{h_1,h_2}_{\mathcal{Z}_1,\mathcal{Z}_2}(M_0) = CMD^{h_1,h_2}_{\mathcal{Z}_1,\mathcal{Z}_2}(M_1)$ with probability $\epsilon \geq 1/p(\lambda)$ for some polynomial p.

We define $\mathcal{A}_{complete}$ as a PPT algorithm which on input $(\mathcal{Z}_1, \mathcal{Z}_2)$ computes $(M_0, M_1) \leftarrow \mathcal{A}(\mathcal{Z}_1, \mathcal{Z}_2)$, queries $h_1^*(M_0)$ and $h_1^*(M_1)$, and outputs (M_0, M_1). In particular, $\mathcal{A}_{complete}$ also breaks collision resistance of $CMD^{h_1,h_2}_{\mathcal{Z}_1,\mathcal{Z}_2}$ with probability ϵ, but $\mathcal{A}_{complete}$ is guaranteed to make every h_1 query needed to compute $h_1^*(M_0)$ and $h_1^*(M_1)$. Let $T_{complete} \leq T_{\mathcal{A}} + 2(\ell + k)$.

Without loss of generality, we denote by M_0 the message which $\mathcal{A}_{complete}$ queries the final round of first.

With probability ϵ, $(M_0, M_1) \leftarrow \mathcal{A}_{complete}(\mathcal{Z}_1, \mathcal{Z}_2)$ satisfies $CMD^{h_1,h_2}_{\mathcal{Z}_1,\mathcal{Z}_2}(M_0) = CMD^{h_1,h_2}_{\mathcal{Z}_1,\mathcal{Z}_2}(M_1)$. If this is true, then one of the following clearly must hold:

1. $h_2^*(M_0, \mathcal{Z}_2)$ queries h_1 on $(\mathcal{Z}_{1,k}, h^*(M_1, \mathcal{Z}_{1,1}, \ldots, \mathcal{Z}_{1,k-1}))$.
2. $h_2^*(M_1, \mathcal{Z}_2)$ queries h_1 on $(\mathcal{Z}_{1,k}, h^*(M_1, \mathcal{Z}_{1,1}, \ldots, \mathcal{Z}_{1,k-1}))$.
3. Neither $h_2^*(M_0, \mathcal{Z}_2)$ nor $h_2^*(M_1, \mathcal{Z}_2)$ query h_1 on $(\mathcal{Z}_{1,k}, h^*(M_1, \mathcal{Z}_{1,1}, \ldots, \mathcal{Z}_{1,k-1}))$.

We denote by ϵ_1 the probability of Case 1 ocurring, and likewise for ϵ_2 for Case 2 and ϵ_3 for Case 3. By a union bound, $\epsilon \leq \epsilon_1 + \epsilon_2 + \epsilon_3$. We will analyze each of these cases individually.

Before we start the case analysis, we will make heavy use of the following process and technical lemma.

In essence, reconstruct takes a set of queries as well as a target final output, and reconstructs an ℓ-block input $c = c_1 \circ \ldots \circ c_\ell$ satisfying $h_1^*(c) = h_{target}$, if one exists. This allows us to implicitly reconstruct messages which the query pattern of an algorithm indicates it evaluated.

We also present the technical Lemma 6.4, whose proof is deferred to Section 6.4.

Lemma 6.4. *Let $h : \{0,1\}^{n+\Delta} \to \{0,1\}^n$ be a random oracle, and let skip^h be a PPT algorithm which makes at most T_{skip} queries to h. Let (x, y) be the output of skip, and let ℓ be such that $|x| = \ell\Delta$. Denote by x_i the i-th block of Δ bits in x. Then, the probability that $(x, y) \leftarrow \mathsf{skip}^h$, $y = h^*(x)$, and there is some $i \leq \ell$ such that skip^h queries h on $(x_i, h^{(i)}(x))$ is at most $(q + 1)/2^n$.*

As a peek ahead, the proof of Lemma 6.4 involves using skip as a compressor for the random oracle. As each case reduces to this lemma, then, we are intrinsically hiding a compression argument inside of each case.

Case 1: We define skip' as follows:

1. Sample $\mathcal{Z}_1, \mathcal{Z}_2 \xleftarrow{\$} \{0,1\}^{k\Delta}$.
2. Sample a random index $t_0 \in [T_{complete}]$.
3. Run $\mathcal{A}_{complete}(\mathcal{Z}_1, \mathcal{Z}_2)$ up until before query t_0.
4. Let $\vec{q} = (q_1, \ldots, q_{T_{complete}})$ denote the $T_{complete}$ h_1-queries made by $\mathcal{A}_{complete}$, where queries made after t_0 are denoted implicitly according to the randomness which $\mathcal{A}_{complete}$ is using.
5. Set $\widetilde{M_0} = \mathsf{reconstruct}(q_{t_0}, \vec{q})$. If $\mathsf{reconstruct}(q_{t_0}, \vec{q}) = \perp$, output $\perp$ instead.
6. Compute $y = h_2(\widetilde{M_0}, \mathcal{Z}_2)$.
7. Output $(\widetilde{M_0} \circ \mathcal{Z}_2, y)$.

Because we are running $\mathcal{A}_{complete}$, must some index t_0 such that in the above procedure, $q_{t_0} = (M_{0,\ell}, h_1^{(\ell-1)}(M_0))$. By Lemma 6.4, then, in this event, with probability at least $1-(T_{complete}+1/2^n)$, $\mathcal{A}_{complete}$ will query all of $h_1^*(M_0)$ in order. Furthermore, if there are no queries among these queries, then these queries will be unique. In this case, it is clear that the conditions for $\mathsf{reconstruct}(q_{t_0}, \vec{q})$ will be fulfilled, and in fact $\mathsf{reconstruct}(q_{t_0}, \vec{q}) = \widetilde{M_0} = M_0$. By a union bound, then, the probability that this is the case is at least $\rho = (T_{complete} + 1)/2^n + T_{complete}^2/2^n$ (If t_0 is guessed correctly).

So, with probability at least $\epsilon_1 - \rho/T_{complete}$, skip' chooses the correct t_0 runs $h_2^*(M_0, \mathcal{Z}_2)$ as desired, which queries $h_1(\mathcal{Z}_{1,i}, h_1^*(M_0, \mathcal{Z}_{1,1}, \ldots, \mathcal{Z}_{1,i-1})$ for all i by assumption. Note that here, the probability is over any internal randomness of $A_{complete}$, the choice of h_1, and the sampling of $\mathcal{Z}_1, \mathcal{Z}_2$.

In particular, there must be some choice of $A_{complete}, h_1, \mathcal{Z}_2$ such that skip' generates $\widetilde{M_0}$ such that $h_2^*(\widetilde{M_0}, \mathcal{Z}_2)$ queries all $h_1(\mathcal{Z}_{1,i}, h_1^*(M_0, \mathcal{Z}_{1,1}, \ldots, \mathcal{Z}_{1,i-1}))$ with probability at least $\epsilon_1 - \rho/T_{complete}$ (where now the probability is only over the random sampling of $\mathcal{Z}_1$). Call these $A_{complete}^{opt}, h_1^{opt}, \mathcal{Z}_2^{opt}$, respectively. We will at last use these to compress any $\mathcal{Z}_1$.

We define $(\mathsf{Enc}, \mathsf{Dec})$ as so: First, $\mathsf{Enc}(\mathcal{Z}_1)$ is defined as so:

1. Compute $(M_0, M_1) \leftarrow A_{complete}^{opt}(\mathcal{Z}_1, \mathcal{Z}_2^{opt})$.
2. If $h_2^*(M_0, \mathcal{Z}_2^{opt})$ does not query $h_1^{opt}(\mathcal{Z}_{1,i}, (h_1^{opt})^*(M_0, \mathcal{Z}_{1,1}, \ldots, \mathcal{Z}_{1,i-1})$ for all i, output $\perp$.
3. Else, if $h_2^*(M_0, \mathcal{Z}_2^{opt})$ does query every $h_1^{opt}(\mathcal{Z}_{1,i}, (h_1^{opt})^*(M_0, \mathcal{Z}_{1,1}, \ldots, \mathcal{Z}_{1,i-1})$ during its computation, let $\widetilde{q}_1, \ldots, \widetilde{q}_k$ be these indices of these queries.

4. Output $M_0, \widetilde{q}_1, \ldots, \widetilde{q}_k$.

We define $\mathsf{Dec}(M_0, \widetilde{q}_1, \ldots, \widetilde{q}_k)$ as so:

1. Initialize s as the empty string.
2. Run $h_2^*(M_0, \mathcal{Z}_2^{opt})$. Whenever this process makes query $\widetilde{q}_i$, let (s_{q_i}, IV_{q_i}) be the query itself and set $s = s \circ s_i$.
3. Output s.

Intuitively, if every h_1 query is made in step (2) of Enc, then Enc outputs the indices of those exact queries. Then, because $\mathcal{Z}_{1,i}$ is the state of each of those queries, Dec can use those queries to recover $\mathcal{Z}_1$. Correctness of the encoding in fact follows directly from the fact that $s_{q_i} = \mathcal{Z}_{1,i}$ for all $i \in [k]$, giving us $\mathsf{Dec}(\mathsf{Enc}(\mathcal{Z}_1) = \mathcal{Z}_1$.

Because we chose our other inputs optimally, we see that this encoding is correct on at least an $(\epsilon_1 - \rho/T_{complete})$-fraction of the space of all $\mathcal{Z}_1$. The output space of Enc, meanwhile, is $2^\ell \cdot T_{h_2}^{k/\Delta}$, as each M_0 is ℓ bits long and we must choose the queries from among the k/Δ h_1^{opt}-calls of h_2, each of which makes up to T_{h_2} queries.

Theorem 2.2 then gives us $(\epsilon_1 - \rho/T_{complete})2^k \leq 2^\ell \cdot T_{h_2}^{k/\Delta}$. Substituting $\rho = (T_{complete}+1)/2^n + T_{complete}^2/2^n$ and solving for ϵ_1, we see this is satisfied by

$$\epsilon_1 \leq \frac{T_g^{k/\Delta}}{2^{k-\ell}} + \frac{T_{complete}^2 + T_{complete} + 1}{2^n T_{complete}} = \frac{T_g^{k/\Delta}}{2^{k-\ell}} + \frac{T_{complete}+1}{2^n} + \frac{1}{T_{complete}2^n}. \tag{2}$$

Case 2: We observe that Case 2 follows the exact same reasoning as in Case 1 when replacing $(M_0, \widetilde{M_0}, t_0)$ with $(M_1, \widetilde{M_1}, t_1)$ respectively. Therefore, we get a similar bound

$$\epsilon_2 \leq \frac{T_g^{k/\Delta}}{2^{k-\ell}} + \frac{T_{complete}+1}{2^n} + \frac{1}{T_{complete}2^n}. \tag{3}$$

Case 3: We define skip' as similarly as in the first case with some important adjustments:

1. Sample $\mathcal{Z}_1, \mathcal{Z}_2 \xleftarrow{\$} \{0,1\}^{k\Delta}$.
2. Sample two random indices $t_0 < t_1 \in [T_{complete}]$.
3. Run $\mathcal{A}_{complete}(\mathcal{Z}_1, \mathcal{Z}_2)$ up until before query t_1.
4. Let $\vec{q} = (q_1, \ldots, q_{T_{complete}})$ denote the $T_{complete}$ h_1-queries made by $\mathcal{A}_{complete}$, where queries made after t_0 are denoted implicitly according to the randomness which $\mathcal{A}_{complete}$ is using.
5. Set $\widetilde{M_0} = \mathsf{reconstruct}(q_{t_0}, \vec{q})$ *and* $\widetilde{M_1} = \mathsf{reconstruct}(q_{t_1}, \vec{q})$. If either call to $\mathsf{reconstruct}$ returns with $\perp$, instead immediately output $\perp$.
6. Compute $y = h_1^*(\widetilde{M_0}, \mathcal{Z}_1) \oplus h_2^*(\widetilde{M_1}, \mathcal{Z}_2) \oplus h_2^*(\widetilde{M_0}, \mathcal{Z}_2)$.
7. Output $(\widetilde{M_1} \circ \mathcal{Z}_2, y)$.

Similar to Cases 1 and 2, we see that with probability that, if both t_0 and t_1 are chosen correctly — i.e., such that $q_{t_0} = (M_{0,\ell}, h_1^{(\ell-1)}(M_0))$ and $q_{t_1} = (M_{1,\ell}, h_1^{(\ell-1)}(M_1))$ — then the probability that $\mathsf{reconstruct}(q_{t_0}, \vec{q}) = \widetilde{M_0} = M_0$ and $\mathsf{reconstruct}(q_{t_1}, \vec{q}) = \widetilde{M_1} = M_1$ is at least $\rho = (T_{comlete}+1)/2^n + T^2_{complete}/2^n$. Note that this is the exact same as in the previous cases as, assuming both t_0 and t_1 are chosen correctly, either both reconstruct will output a value or both will output $\perp$. So, we only have to account for collisions or errors in reconstruct once.

Because we must choose two indices correctly this time in order to reconstruct M_0 and M_1, we see that with probability at least $1 - \rho/T^2_{complete}$, we have chosen both t_0, t_1 such that $\widetilde{M_0} = M_0$ and $\widetilde{M_1} = M_1$. In particular, then, with probability at least $\epsilon_3 - \rho/T^2_{complete}$, we see $y = h_1^*(\widetilde{M_0}, \mathcal{Z}_1) \oplus h_2^*(\widetilde{M_1}, \mathcal{Z}_2) \oplus h_2^*(\widetilde{M_0}, \mathcal{Z}_2) = h_1^*(M_0, \mathcal{Z}_1) \oplus h_2^*(M_1, \mathcal{Z}_2) \oplus h_2^*(M_0, \mathcal{Z}_2) = h_1^*(M_1, \mathcal{Z}_1)$. By the assumption of this case, neither $h_2^*(M_1, \mathcal{Z}_2)$ nor $h_2^*(M_0, \mathcal{Z}_2)$ will query h_1 on input $(\mathcal{Z}_{1,k}, h^*(M_1, \mathcal{Z}_{1,1}, \ldots, \mathcal{Z}_{1,k-1}))$. Because $M_0 \neq M_1$, $h_1^*(\widetilde{M_0}, \mathcal{Z}_1)$ also does not make this query. However, we see $y = h_1^*(M_1, \mathcal{Z}_1) = h_1^*(\widetilde{M_1} \circ \mathcal{Z}_2)$. So, we see that skip' has output a pair $(x, h_1^*(x))$ without querying the final h_1 query of $h_1^*(x)$. Because h_1 is chosen uniformly at random, the probability that any algorithm can achieve this is at most $1/2^n$. In particular, this yields

$$\epsilon_3 - \rho/T^2_{complete} \leq \frac{1}{2^n}.$$

Substituting in $\rho = (T_{comlete}+1)/2^n + T^2_{complete}/2^n$ and solving for ϵ_3, we have

$$\epsilon_3 \leq \frac{T^2_{complete} + T_{complete} + 1}{T^2_{complete} \cdot 2^n} + \frac{1}{2^n} = \frac{1}{2^n} + \frac{1}{2^{n-1}} + \frac{1}{T^2_{complete} \cdot 2^n}. \tag{4}$$

Putting it all together: As stated, we know $\epsilon \leq \epsilon_1 + \epsilon_2 + \epsilon_3$. Substituting in the upper bounds from Equations 2,3, and 4, we get

$$\epsilon \leq \frac{T_g^{k/\Delta}}{2^{k-\ell-1}} + \frac{2T_{complete}+5}{2^n} + \frac{3}{T_{complete} \cdot 2^n} \tag{5}$$

Substituting back in $T_{complete} \leq T_{\mathcal{A}} + 2(\ell + k)$, this gives us the bound

$$\epsilon \leq \frac{T_g^{k/\Delta}}{2^{k-\ell-1}} + \frac{2T_{\mathcal{A}} + 4(\ell+k) + 5}{2^n} + \frac{3}{(T_{\mathcal{A}} + 2(\ell+k)) \cdot 2^n} \tag{6}$$

□

6.4 Proof of Lemma 6.4

Proof. Let ϵ be the probability over all h, skip that $(x,y) \leftarrow \mathsf{skip}^h$, $y = h^*(x)$, and there is some $i \leq \ell$ such that skip^h queries h on $(x_i, h^{(i)}(x))$. (That is, ϵ is the value we must bound above by $(q+1)/2^n$.)

In addition, let ϵ_{skip} be the probability that $(x,y) \leftarrow \mathsf{skip}^h$, $y = h^*(x)$, and there is some $i \leq \ell$ such that skip^h queries h on $(x_i, h^{(i)}(x))$, where the probability is over the internal randomness over h. We see by an averaging argument that there must be some skip^{opt} such that $\epsilon_{\mathsf{skip}^{opt}} \geq \epsilon$. We will use this optimal skip to compress the random oracle h.

We define (Enc, Dec) as so: First, Enc(h) is defined as so:

1. Find $(x, y) \leftarrow \mathsf{skip}^{opt}$ with $|x| = \ell\Delta$.
2. Let t be the last index such that $h(x_{t+1}, h^{(t)}(x))$ is queried before $h(x_t, h^{(t-1)}(x))$. If no such index exists, output $\perp$.
3. Let $q_t = (x_t, h^{(t-1)}(x))$.
4. Define i as the query index of skip^{opt} where $h(x_{t+1}, h^{(t)}(x))$ is queried. In the case where $t = \ell$ (and therefore the final round of $h^*(x)$ is simply not queried), set $i = -1$.
5. Let $h \setminus q_t$ be the truth table of h except the entry $h(q_t)$ removed. Output $(i, h \setminus q_t)$.

We define $\mathsf{Dec}(i, h \setminus q_t)$ as so:

1. If $i = -1$, run skip^{opt} by referring to $h \setminus q_t$ to receive (x, y). Define $h(q_t) = y$ and output $h \setminus q_t$ with this entry added.
2. Else, run skip^{opt} b referring to $h \setminus q_t$, stopping just before query i would be made. Let (s, IV) be the intended input at this query, and define $h(q_t) = IV$. Output $h \setminus q_t$ with this entry added.

We show that, if Enc does not output $\perp$, then $\mathsf{Dec}(\mathsf{Enc}(h)) = h$. First, if $i = -1$ in Enc, then in particular skip^{opt} never queries $h(x_\ell, h^{(\ell-1)}(x))$. So, we may safely run skip^{opt}. By assumption with probability at least ϵ, skip^{opt} still outputs $y = h^*(x) = h(x_\ell, h^{(\ell-1)}(x))$, and so Dec correctly inputs this value for $h(q_t)$. If $i \geq 0$, on the other hand skip^{opt} behaves equivalently with access to h and $h \setminus q_t$ up to query i. So, $(s, IV) = (x_t, h^{(t-1)}(x))$ with probability at least ϵ, and so we set $h(q_t) = h(x_t, h^{(t-1)}(x)) = h^{(t)}(x) = IV$, as desired. Note that with probability at least ϵ, we are guaranteed that skip^{opt} does query out of order by assumption, and so in these cases we will also not receive $\perp$ from Enc. This satisfies correctness of our encoding.

So, we compress an ϵ-fraction of all possible functions (from m bits to n bits) to a set of query indices with all functions with one entry removed. This means we compress a set of size at least $\epsilon \cdot 2^{n2^m}$ to a set of size $(q+1)2^{n2^m}/2^n$, as q is the maximum number of queries which skip^{opt} may make, but i may also be set to -1. So, we must have

$$\epsilon \leq \frac{q+1}{2^n}.$$

□

7 Conclusion

7.1 Recommendations for Practice

We give two recommendations for practice.

Situation 1: You have two compression functions $h_1, h_2 : \{0,1\}^m \to \{0,1\}^n$, at least one of which is believed to be a random oracle. $n \leq m/4$.

We recommend using the construction from Corollary 5.3. Note that in this Corollary, T represents the number of times the compromised hash function can evaluate the uncompromised one. That is, if h_b runs in time T_b, then (if $t_1 > t_0$) $T \leq \frac{t_1}{t_0}$. Since it is unlikely that one would wish to combine two hash functions where one of the hash functions takes a million times as long to run, we will assume that $T \leq 2^{30}$.

Recall from Section 4.1 that to achieve birthday bound security one should set

$$k \geq \frac{m + \log T + \lambda}{2} = \frac{m + 30 + \lambda}{2}.$$

Note that in order to apply this construction to an input of length s, we require $\frac{s}{m-k-n-1}$ calls to the underlying hash functions. That is, applying this construction requires

$$2 \cdot \frac{m - n}{m - k - n - 1}$$

times as many queries to the underlying hash functions than applying Merkle-Damgård directly to a trusted hash function would.

Suggestion Takeaway: For any hash function property, if $n = \lambda = 256$, $m = 1024$, this construction has birthday bound security loss compared to a random oracle and requires

$$\leq 2 \cdot \frac{768}{227} \leq 7$$

times as many hash evaluations. Furthermore, the hash size is identical to the hash size of the underlying compression functions.

Downside: Our security proof does not hold when the honest hash functions is instantiated as Merkle-Damgård applied to some random oracle. That is, this suggestion should be applied only to the underlying hash functions for SHA2/SHA3/... directly.

Situation 2: You have two hash functions $h_1, h_2 : \{0,1\}^* \to \{0,1\}^n$ both instantiated as Merkle-Damgård applied to some underlying compression functions.
We recommend sampling some $\mathcal{Z}_1, \mathcal{Z}_2 \xleftarrow{\$} \{0,1\}^k$, and using the hash function

$$C_{\mathcal{Z}_1,\mathcal{Z}_2}(M) = h_1(M, \mathcal{Z}_1) \oplus h_2(M, \mathcal{Z}_2)$$

By Theorem 6.3, this hash function will be collision resistant for k sufficiently large.

In order to achieve near birthday bound security, k should be set such that $\frac{T_g^{k/\Delta}}{2^{k-\ell-1}} + \frac{2T_{\mathcal{A}}+4(\ell+k)+5}{2^n} + \frac{3}{(T_{\mathcal{A}}+2(\ell+k))\cdot 2^n} = 1/2^\lambda$, where $\Delta = m - n$ is the compression of the hash functions underlying h_1 and h_2. As in the previous situation, we can assume $T_g \leq 2^{30}$.

That is, for an appropriately set value of n, we will want

$$\frac{T_g^{k/\Delta}}{2^{k-\ell-1}} \approx \frac{1}{2^\lambda}$$

Note that this construction requires $\frac{k}{\Delta}$ additional compression function evaluations compared to simply concatenating the underlying hash functions. In particular, to achieve near birthday bound security, we require

$$\frac{k}{\Delta} = \frac{\lambda + \ell}{\Delta - 30}$$

additional hash evaluations.

Suggestion Takeaway: To achieve a collision resistant hash function from 1024 bits to 256 bits, if $\Delta = \lambda = 256$, then this construction will require

$$\frac{k}{\Delta} \leq \frac{256 + \ell}{226} \leq 6$$

additional hash evaluations. Furthermore, the hash size is identical to the hash size of the underlying hash functions.

Downside: The security proof for this construction only holds for collision-resistance. Furthermore, the amount of extra blocks required depends linearly on the length of the input, although the linear factor of $\frac{1}{\Delta - 30}$ is quite small.

7.2 Alternative One-stage Model

As observed in Section 3.1, our random oracle combiner security definition is two-stage. One may wonder if there is a reasonable one-stage security definition, so that composability properties may be used more easily. Observe that the reason the game is two-stage is because both the distinguisher $\mathcal{D}$ and the hash function g have oracle access to the idealized model, and can thus communicate through their shared state.

We could instead consider a model where g is not allowed oracle access to the idealized model. For example, g may be a fixed (non-oracle) circuit chosen by an adversary which makes polynomial queries to h. This can then be considered a one-stage game, and so this weakened combiner security property composes with indifferentiability.

Note that this model is strictly weaker than the other, since an arbitrary circuit could run the adversary inside of itself. Thus, our positive results also hold in this model. Upon inspection of the proof, we observe that our lower bounds from Section 4.3 also apply.

This model represents the scenario where the adversarial hash function g does not "run" h. In the real world, if g were to run h, we may hope that this would be detected by comparing the two hash functions' code and running times. Nevertheless, this leaves the combiner open to some attacks, and so we consider this model to be of lesser practical consideration. For the sake of brevity, we do not analyze it in detail in this work.

7.3 Open Questions

We include a number of open questions on this topic.

Is there a random-oracle-combiner construction which provably composes with indifferentiability?

Our random-oracle-secure construction does not perfectly match the randomness requirement in our lower bound. Is it possible to improve the randomness requirement for the random-oracle-secure construction.

The random-oracle-secure combiner construction presented takes hash functions from m bits to n bits and produces a hash function mapping ℓ bits to n bits, where $\ell = m - |\mathcal{Z}|/2$. Is it possible to construct a random-oracle-secure combiner with longer input length using the same amount of randomness?

As a particular approach to both of the previous two questions, is the CMD construction from Section 6 random-oracle-secure?

Is it possible to construct more efficient combiners for weaker security notions than indifferentiability or collision-resistance? For example, is there a more efficient pseudorandom combiner? Is there a more efficient combiner in the one-stage model from Section 7.2?

References

[1] Specifications for the secure hash standard. Federal Inf. Process. Stds. (NIST FIPS), 2002.

[2] C.A. Asmuth and G.R. Blakley. An efficient algorithm for constructing a cryptosystem which is harder to break than two other cryptosystems. *Computers & Mathematics with Applications*, 7(6):447–450, 1981.

[3] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In Douglas R. Stinson, editor, *Advances in Cryptology – CRYPTO'93*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249. Springer, Heidelberg, August 1994.

[4] Dan Boneh and Xavier Boyen. On the impossibility of efficiently combining collision resistant hash functions. In Cynthia Dwork, editor, *Advances in Cryptology – CRYPTO 2006*, volume 4117 of *Lecture Notes in Computer Science*, pages 570–583. Springer, Heidelberg, August 2006.

[5] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. 17:297–319, 2004.

[6] Ran Canetti, Ronald L. Rivest, Madhu Sudan, Luca Trevisan, Salil P. Vadhan, and Hoeteck Wee. Amplifying collision resistance: A complexity-theoretic treatment. In Alfred Menezes, editor, *Advances in Cryptology – CRYPTO 2007*, volume 4622 of *Lecture Notes in Computer Science*, pages 264–283. Springer, Heidelberg, August 2007.

[7] Jean-Sébastien Coron, Yevgeniy Dodis, Cécile Malinaud, and Prashant Puniya. Merkle-damgård revisited: How to construct a hash function. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 430–448. Springer, Heidelberg, August 2005.

[8] Yevgeniy Dodis, Iftach Haitner, and Aris Tentes. On the instantiability of hash-and-sign rsa signatures. In Ronald Cramer, editor, *TCC 2012: 9th Theory of Cryptography Conference*, volume 7194 of *Lecture Notes in Computer Science*. Springer, Heidelberg, March 2012.

[9] Yevgeniy Dodis, Roberto Oliveira, and Krzysztof Pietrzak. On the generic insecurity of the full domain hash. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*. Springer, Heidelberg, August 2005.

[10] Morris Dworkin. Sha-3 standard: Permutation-based hash and extendable output functions. Federal Inf. Process. Stds. (NIST FIPS).

[11] Marc Fischlin and Anja Lehmann. Multi-property preserving combiners for hash functions. In Ran Canetti, editor, *TCC 2008: 5th Theory of Cryptography Conference*, volume 4948 of *Lecture Notes in Computer Science*, pages 375–392. Springer, Heidelberg, March 2008.

[12] Marc Fischlin, Anja Lehmann, and Krzysztof Pietrzak. Robust multi-property combiners for hash functions. *Journal of Cryptology*, 27(3):397–428, July 2014.

[13] Oded Goldreich, Yoad Lustig, and Moni Naor. On chosen ciphertext security of multiple encryptions. Cryptology ePrint Archive, Report 2002/089, 2002. `https://eprint.iacr.org/2002/089`.

[14] Danny Harnik, Joe Kilian, Moni Naor, Omer Reingold, and Alon Rosen. On robust combiners for oblivious transfer and other primitives. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 96–113. Springer, Heidelberg, May 2005.

[15] Amir Herzberg. On tolerant cryptographic constructions. In Alfred Menezes, editor, *Topics in Cryptology – CT-RSA 2005*, volume 3376 of *Lecture Notes in Computer Science*, pages 172–190. Springer, Heidelberg, February 2005.

[16] Ueli Maurer, Renato Renner, and Clemens Holenstein. Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In Moni Naor, editor, *TCC 2004: 1st Theory of Cryptography Conference*, volume 2951 of *Lecture Notes in Computer Science*, pages 21–39. Springer, Heidelberg, February 2004.

[17] Remo Meier and Bartosz Przydatek. On robust combiners for private information retrieval and other primitives. In Cynthia Dwork, editor, *Advances in Cryptology – CRYPTO 2006*, volume 4117 of *Lecture Notes in Computer Science*, pages 555–569. Springer, Heidelberg, August 2006.

[18] Bart Mennink and Bart Preneel. Breaking and fixing cryptophia's short combiner. In Dimitris Gritzalis, Aggelos Kiayias, and Ioannis G. Askoxylakis, editors, *CANS 14: 13th International Conference on Cryptology and Network Security*, volume 8813 of *Lecture Notes in Computer Science*, pages 50–63. Springer, Heidelberg, October 2014.

[19] Arno Mittelbach. Cryptophia's short combiner for collision-resistant hash functions. In Michael J. Jacobson Jr., Michael E. Locasto, Payman Mohassel, and Reihaneh Safavi-Naini, editors, *ACNS 13: 11th International Conference on Applied Cryptography and Network Security*, volume 7954 of *Lecture Notes in Computer Science*, pages 136–153. Springer, Heidelberg, June 2013.

[20] Pascal Paillier. Impossibility proofs for RSA signatures in the standard model. In Masayuki Abe, editor, *Topics in Cryptology – CT-RSA 2007*, volume 4377 of *Lecture Notes in Computer Science*, pages 31–48. Springer, Heidelberg, February 2007.

[21] Krzysztof Pietrzak. Non-trivial black-box combiners for collision-resistant hash-functions don't exist. In Moni Naor, editor, *Advances in Cryptology – EUROCRYPT 2007*, volume 4515 of *Lecture Notes in Computer Science*, pages 23–33. Springer, Heidelberg, May 2007.

[22] Krzysztof Pietrzak. Compression from collisions, or why CRHF combiners have a long output. In David Wagner, editor, *Advances in Cryptology – CRYPTO 2008*, volume 5157 of *Lecture Notes in Computer Science*, pages 413–432. Springer, Heidelberg, August 2008.

[23] Thomas Ristenpart, Hovav Shacham, and Thomas Shrimpton. Careful with composition: Limitations of the indifferentiability framework. In Kenneth G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*. Springer, Heidelberg, May 2011.