Random Oracle Combiners: Merkle-Damgård Style

Yevgeniy Dodis*, Eli Goldin†, and Peter Hall‡

New York University

April 3, 2025

Abstract

A *Random Oracle Combiner* (ROC), introduced by Dodis et al. (CRYPTO '22), takes two hash functions h_1, h_2 from m bits to n bits and outputs a new hash function C from m' to n' bits. This function C is guaranteed to be indifferentiable from a fresh random oracle as long as one of h_1 and h_2 (say, h_1) is a random oracle, while the other h_2 can "arbitrarily depend" on h_1.

The work of Dodis et al. also built the first *length-preserving* ROC, where $n' = n$. Unfortunately, despite this feasibility result, this construction has several deficiencies. From the practical perspective, it could not be directly applied to existing Merkle-Damgård-based hash functions, such as SHA2 or SHA3. From the theoretical perspective, it required h_1 and h_2 to have input length $m > 3\lambda$, where λ is the security parameter.

To overcome these limitations, Dodis et al. conjectured — and left as the main open question — that the following (salted) construction is a length-preserving ROC:

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

where $\mathcal{Z}_1, \mathcal{Z}_2$ are random salts of appropriate length, and f^* denotes the Merkle-Damgård-extension of a given compression function f.

As our main result, we *resolve this conjecture in the affirmative*. For practical use, this makes the resulting combiner applicable to existing, Merkle-Damgård-based hash functions. On the theory side, it shows the existence of ROCs only requiring optimal input length $m = \lambda + O(1)$.

Keywords: hash functions, combiners, random oracle model, indifferentiability framework.

1 Introduction

Cryptographic combiners [20, 19] take two implementations of a given primitive, into a new instantiation of this primitive which is provably secure as long as one of the two input instantiations is secure, and the other could be arbitrary. A long line of work on the subject [3, 18, 19, 20, 23] studied combiners for various cryptographic primitives, with particular attention given to hash function combiners [6, 8, 26, 27]. The reason for this interest is twofold. First, the question of choosing a

*Email: dodis@cs.nyu.edu

†Email: eli.goldin@nyu.edu. Eli Goldin is supported by an NSF Graduate Research Fellowship.

‡Email: pf2184@nyu.edu.

given hash function between mutually untrusted parties is quite common. For example, [12] mention an application when one party is a software company trusting one hash function h_1, but the potential client must use another function h_2 (say, per client's country different hash standard). Hash combiners, when simple and efficient, provide a perfect solution to this dilemma.

Second, there is a particular technical issue in building such hash combiners: output length. Concretely, the most natural and common way to formalizing security of hash combiners is through their collision-resistance (CR) property, as this is typically the minimal security property required from a hash function. And there is a trivial CR-combiner C: concatenation, defined as $C^{h_1,h_2}(M) = h_1(M)\|h_2(M)$. While simple, for many applications doubling the output length is extremely inconvenient. For example, in hash-then-sign signatures (such as FDH [4] or BLS [7]), this forces one to use much heavier public-key cryptography, and similar considerations apply for virtually any real-world application we can think of. Unfortunately, the works of [6, 26, 27] showed that this limitation is inherent, even if the combiner is allowed to be salted (with the salt chosen independently of h_1 and h_2).

Random Oracle Combiners. Fortunately, extending an earlier "cryptophia" proposal by Mittelbach [25], Dodis et al. [12] recently proposed a different solution to this unfortunate state of affairs, while simultaneously providing a possibly better formalization to "hash-combiners" than CR-combiners. They noticed that many applications of hash functions anyway assume stronger/different properties than collision-resistance, and are analyzed in the Random Oracle (RO) model. As a result, it seems natural to define an build *random oracle combiners* (ROCs). While starting with the stronger assumption of random oracle than CR, they also reach a similarly stronger conclusion. Which is anyway what is needed for most applications.

Moreover, ROCs no longer have the unfortunate CR-combiners lower bound on the output length! Indeed, Dodis et al. [12] build a simple and efficient *length-preserving* ROC C, as follows:

$$C^{h_1,h_2}_{\mathcal{Z}_1,\mathcal{Z}_2}(M) := h_1(M,\mathcal{Z}_1) \oplus h_2(M,\mathcal{Z}_2), \tag{1}$$

where $\mathcal{Z}_1, \mathcal{Z}_2$ are sufficiently long random salts.[1] At first, this appears to almost settle the question of building practical hash combiners. The combiner C is quite efficient, making only one call to the underlying hash functions h_1 and h_2. Also, although it turns out that the lengths of the salts $\mathcal{Z}_1$ and $\mathcal{Z}_2$ must be slightly longer than the length of the message M, this roughly loses "only" a factor of 2 in terms of bandwidth for long enough messages M.

Unfortunately, as was already noticed by [12], and as we discuss next, the construction above was not really applicable to combining existing hash functions such as SHA2 and SHA3. Additionally, even ignoring practicality and applicability constraints, the combiner does not achieve theoretically optimal feasibility of ROCs. We explain both of these deficiencies separately.

For simplicity of exposition and by symmetry, below we will always assume that h_1 is the true random oracle, while h_2 is the adversarial one.

The Practical View of ROCs. Imagine we want to apply a given ROC — e.g., the one in Equation (1) — to an existing "good" hash function F, such as SHA2 or SHA3. (We do not care about the "bad" hash function for now.) The most obvious way of doing it would be to model F as the good RO h_1 in Equation (1). Unfortunately, all existing cryptographic hash function

[1] Dodis et al. [12] showed that no deterministic ROCs are possible, so the existence of salt is necessary.

F are not "monolithic", and have a lot of lower-level structure to it. In particular, virtually all of them are based of iterating some *compression function* f using the classical Merkle-Damgård transform. Ignoring some small variations regarding the last block of the input,[2] the Merkle-Damgård transform of a given compression function f splits the long input M into appropriate-length blocks $M = (m_1, \ldots, m_\ell)$ and outputs:

$$f^*(M) = f(\ldots f(f(0, m_1), m_2), \ldots), m_\ell)$$

In particular, as first advocated by Coron et al [9] and many follow-up works, it is very dangerous to ignore the Merkle-Damgård structure of existing functions F, and assume that the entire $F = f^*$ is a monolithic random oracle. Instead, the approach advocated by [9], and followed by virtually all the follow-up work, is to only model the underlying *compression function f as ideal*, thereby accounting for the Merkle-Damgård structure of F. In the world of ROCs, this means that:

One should model h_1 as the compression function f,
and not as the overall hash function $F = f^*$

This leaves three avenues to proceed, if we still want to apply the ROC in Equation (1) to real-world hash functions F.

1. The first one is to attempt using the indifferentiability composition framework of [21], which is a formal way to say that any (single-stage) security game secure assuming one ideal object, is still secure if this object is replaced by a construction from other (say, smaller) ideal objects. Indeed, Coron et al. [9] showed that many Merkle-Damgård-like hash functions, — including plain Merkle-Damgård $F(M) = f^*(M)$ for *fixed lengths inputs* M, — yield a construction which is indifferentiable from a "monolithic" RO. This seems to suggest that the ROC from Equation (1) would be secure if $h_1 = f^*$, and f is modeled as RO. Unfortunately, as shown by [12], this reasoning is false. The indifferentiability-style ROC definition of [12] is a "two-stage" security game [28], to which the composition theorem of [21] does *not* apply. Moreover, this was not a hypothetical mismatch, as [12] provided a convincing counter-example. Thus, this approach does not work *generically*: the resulting construction is either insecure, or a new, direct proof of its security *must be provided*. We will come back to this point shortly.

2. The second avenue is to indeed directly apply the ROC from Equation (1) to the compression function f; i.e., set $h_1 = f$. This has two big issues. First, the entire input $(M, \mathcal{Z}_1)$ has to fit inside the compression function $f = h_1$. And since the salt $\mathcal{Z}_1$ in the analysis of [12] has to be at least security parameter λ longer than the message M, the largest block we can insert inside C would be $(m - \lambda)/2$, where m is the length of the compression function. For SHA2, $m = 512$, and setting $\lambda = 128$ leaves less than 200 bits left. Which is probably not enough for any practical use. For SHA3, $m = 1600$, which leaves more than 700 bits for the message M. This is more reasonable, but still not enough to be directly used in most applications of the combiner (e.g., to hash-and-sign signatures, and others).

   This brings us to the second issue with this approach: *huge inconvenience*. Instead of using a given "good" hash function like SHA2 or SHA3 as a *black-box*, we will be forced to explicitly

[2] The subtle variations with the last blocks are needed to prevent so called "extension attacks" for *variable-length* inputs. Since variable-length inputs are not super-relevant for the rest of the Intoduction, and since the "plain" Merkle-Damgård transform is secure [9] for *fixed-length* inputs, we will not worry about low-level details concerning the last block, until relevant much later in the paper.

extract their corresponding compression functions f, apply Equation (1) to f (and another "bad" compression function) to build a new compression function C, and finally directly implement some Merkle-Damgård-like mode of operation to this new compression function C. In sum, this approach is either inapplicable or completely impractical.

3. The final option, and indeed one advocated by [12], is to directly build a ROC $\widetilde{C}^{h_1,h_2}(M)$ which: (a) only treats a fixed-length compression function $h_1 = f$ as its "good" random oracle in its analysis; but (b) only makes black-box calls to the *iterated* function $F = f^*$ (e.g., SHA2 or SHA3, so it does not need to "yank out" anything from the internals of such hash functions). And, of course, (c) is capable of supporting "long-enough" inputs M for the final applications of the combiner. Unfortunately, we have seen that the *combiner in Equation (1) does not meet these properties (a)-(c).* Looking ahead, our main result will resolve this issue.

The Theoretical View of ROCs. We have concluded that properties (a)-(c) above are important for practical applicability of ROCs. But we can also momentarily ignore all these practical considerations, and instead focus on the theoretical feasibility only. From this perspective, and assuming λ is a security parameter, we are asking about the minimal input length m for the "good" hash function h_1 that is sufficient to build a good ROC. Notice, the output length n of h_1 is not important in terms of feasibility, as even 1-bit output h_1 can be converted to $n = \text{poly}(\lambda)$ at the expense of "only" $\log n = O(\log \lambda)$ bits in the input (and $n = \text{poly}(\lambda)$ slowdown in efficiency, which is fine from this perspective).

Also, we clearly need $m \geq \lambda$, if λ is the security parameter. Moreover, since we mentioned that we cannot apply the composition theorem of the indiffentiability framework to the input function h_1, we cannot start with a poor choice of $m \gg \lambda$, and then "reduce" it to the $m \approx \lambda$ using some domain-extension technique for random oracles (i.e., the powerful beyond-birthday domain extension result of Maurer and Tessaro [22]). Thus, we need to *minimize* m directly. Finally, for simplicity we will also require the ROC C to support at least inputs of length $m' \geq \lambda$, as otherwise we cannot even achieve security $\text{poly}(\lambda)/2^{\lambda}$ for the final application, against polynomial time attackers.[3]

We can now evaluate the minimal length m for the ROC in Equation (1) to work. As we already mentioned, the salt $\mathcal{Z}_1$ in the analysis of [12] has to be at least security parameter λ longer than the message M, which has to be at least λ bits long. This means that $m = |M| + |\mathcal{Z}_1| \geq 2|M| + \lambda \geq 3\lambda$.

In contrast, we shall see that our ROC will only require optimal $m = \lambda + \tilde{O}(1)$, in addition to satisfying the desired properties (a)-(c) described in the "Practical View" above.

The Conjecture of Dodis et al. As we mentioned, the original work of [12] noticed the limitations of the ROC construction in Equation (1). To overcome these limitations, they directly revisited the "failed" composition attempt of building a "monolithic" $h_1 = f^*$ in Equation (1) from a smaller compression function f, modeled as a fixed-length random oracle. In fact, renaming this new compression function by h_1 itself, — as required by ROC's syntax, — we arrive at the following ROC candidate proposed by [12]:

$$\widetilde{C}^{h_1,h_2}_{\mathcal{Z}_1,\mathcal{Z}_2}(M) := h_1^*(M, \mathcal{Z}_1) \oplus h_2^*(M, \mathcal{Z}_2), \tag{2}$$

[3] As we will see, our final construction in Equation (2) will naturally work for any $m' = \text{poly}(\lambda)$, but for the perspective of comparing to the construction in Equation (1), we will just settle on minimal acceptable $m' \geq \lambda$.

Namely, it simply replaced "monolithic" h_1 and h_2 by their Merkle-Damgård-implementations which will anyway occur in practice. We can immediately see that this construction, if secure, would simultaneously achieve all the desired properties mentioned so far, including:

- *Merkle-Damgård-friendliness:* $\widetilde{C}$ only makes a single black-box call to the *iterated* functions h_1^* and h_2^*. This means the combiner does not need to "yank out" anything from the internals of such hash functions, and use existing libraries, for example.

- *Length-Preservation:* The output of $\widetilde{C}$ is the same as that of the given hash functions h_1 and h_2, once again overcoming the "concatenation barrier" for collision resistant combiners [6, 26, 27].

- *Large Combiner Input:* The input M for the combiner can be made effectively as large as needed, as Merkle-Damgård iteration can handle arbitrary long inputs, by chopping them into smaller blocks.

- *Feasibility-Friendliness:* Since h_1 and h_2 operate via Merkle-Damgård, there is no issue in needing to fit the salts $\mathcal{Z}_1$ and $\mathcal{Z}_2$ inside the compression function. Thus, at least in principle (see below for our results), this ROC candidates has the potential of supporting arbitrary compression functions h_1 and h_2 from m bits to n bits, satisfying $m > n \geq \lambda$. In particular, if efficiency is not an issue,[4] the input m might be as short as theoretically optimal $m = n+1 = \lambda + 1$.

The authors of [12] also realized that, while their proof for the simpler ROC in Equation (1) does not translate to that of ROC in Equation (2), there is no explicit attack on this variant, as long as the salts are *appended to the input* M.[5] Indeed, they conjectured that the ROC $\widetilde{C}$ in Equation (2) is secure. As partial evidence, they directly showed that $\widetilde{C}$ is at least collision-resistant. Which is already very interesting, as this overcomes the "concatenation barrier" for collision-resistant combiners, at the cost of making a much stronger (random oracle) assumption on the "good" compression function h_1.

Unfortunately, the collision-resistant proof, while giving some confidence in the security of $\widetilde{C}$, is a much weaker properly than indifferentiability from random oracle, required by ROC. Even syntactically, while the collision-resistance proof "only" has to show resilience to a particular type of collision-finding attacker, the indiffentiability proof must define a general "indifferentiability simulator", and then formally argue that no distinguisher can tell apart the real world from the ideal world (the latter with the simulator). We expand on this in Section 1.1, but mention that we did not manage to meaningfully extend the collision-resistant proof from [12] to a full indifferentiability analysis (e.g., define the simulator, analyze its termination, or do the actual indifferentiability proof).

[4] Recall, each Merkle-Damgård call to the compression function is processing $\delta = m - n$ bits of M, and the number of such calls is $|M|/\delta$. Thus, if one makes $n = \lambda$ and δ as small as 1, the number of compression calls increases to $|M|$, compared to at most $|M|/\lambda$, when the compression function is at least length-doubling (as is the case in the real-world).

[5] They had attacks on the prepend version, which they used to demonstrate the lack of composition properties for ROCs.

Our Main Result. Nevertheless, our main result resolves this conjecture in the affirmative: *the Random Oracle Combiner in Equation (2) is secure.*

The result is formally stated in Theorem 3.1, but here is an informal version, more precisely accounting for salt lengths:

Theorem 1.1 (Informal version of Theorem 3.1)**.** *Let $h_1, h_2 : \{0,1\}^{n+\delta} \to \{0,1\}^n$ be two oracles. Then, as long as $|\mathcal{Z}_1| = |\mathcal{Z}_2| > |M| + \lambda$, the following construction $\widetilde{C}^{h_1,h_2}$ is a secure random oracle combiner:*

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

We notice that this construction is simultaneously satisfying all the attractive properties mentioned above. In particular, it gives an efficient hash combiner both for practical use (output preservation, Merkle-Damgård-friendliness, large input, good exact security), as well as in theory (the minimal input length m of h_1 can be as low as $\lambda + O(1)$).

1.1 Technical Overview

We briefly present the idea of our proof, and the difficulties we had to overcome.

The Definitional Framework. We first recall the indifferentiability framework for hash functions, due to Coron et al. [9]. The high-level objective is to build a simulator Sim in the "ideal world" which can fool any polynomial-query distinguisher $\mathcal{D}$ into thinking that $\mathcal{D}$ is in the "real world". These worlds are defined as follows: In the real world, $\mathcal{D}$ has access to a true random oracle h_1, and the real-world implementation of the combiner $\widetilde{C}^{h_1,h_2}$. In the ideal world, the combiner is replaced by a true $|M|$-bit random oracle H, while the compression function h_1 must be "faked" by the simulator Sim, so that the outputs of Sim make the fake h_1 "consistent" with H, *as if H was also following the combiner construction.* For the lack of better name, we call this task of the simulator as *combiner-consistency.*

In the world of ROCs [12], there is an additional complication for the simulator, having to do with the adversarial hash function h_2. In the preprocessing step, we allow the distinguisher to select an arbitrary oracle circuit $g^{(\cdot)}$ which is allowed to make a bounded number of oracle calls to a true random oracle h_1. In the real world, we then define $h_2 = g^{h_1}$, while in the ideal world the simulator gets $g^{(\cdot)}$, but now has to define fake h_1 so that combiner-consistency of a true random function holds w.r.t. $h_1 = \mathsf{Sim}$ and $h_2 = g^{\mathsf{Sim}}$.

This is quite ambitious, as every time Sim defines h_1 on some input, this could force the definition of $h_2 = g^{h_1}$, making it hard to respect the combiner-consistency. In fact, Dodis et al. [12] showed that this is *impossible* to do deterministically. This is where the salts $\mathcal{Z} = (\mathcal{Z}_1, \mathcal{Z}_2)$ come in. $\mathcal{D}$ has to commit to g before the salts are chosen. Afterwards, all the parties (including $\mathcal{D}$ and Sim) get the salts, with the hope that it is too late for the circuit g to "mess up" the combiner-consistency property, despite the fact that $\mathcal{D}(\mathcal{Z})$ can freely evaluate $g^{\mathsf{Sim}}(M)$ on many inputs M which *depend* on the salt $\mathcal{Z}$.

Easy Case: The "Monolithic" Combiner. For the construction of [12] given in Equation (1), this simulator task turns out to be simple. This is because the entire long salt $\mathcal{Z}_1$ can fit inside the monolithic input to h_1, making the salted random oracle $H'(M) = h_1(M, \mathcal{Z}_1)$ completely

independent from the adversarial hash function g (which was chosen before $\mathcal{Z}_1$ was known). In particular, the simulator can define

$$h_1(M, \mathcal{Z}_1) = H(M) \oplus g^{h_1}(M, \mathcal{Z}_2)$$

to ensure combiner-consistency, provided $g^{h_1}(M, \mathcal{Z}_2)$ never called h_1 on *any input* ending in $\mathcal{Z}_1$, even if the input M above was chosen by the distinguisher *as a function of the salt* $\mathcal{Z}_1$. Fortunately, this is easy to ensure in the real-world (meaning the simulator never needs to worry about this in the ideal world). Indeed, all one needs to do is make the salt-length $|\mathcal{Z}_1| \geq |M| + \lambda$, and a simple union bound does the trick: the input M is too short to encode enough information about $\mathcal{Z}_1$ to cause $g^{h_1}(M, \mathcal{Z}_2)$ to ever evaluate $h_1(\cdot, \mathcal{Z}_1)$.

Hard Case: The "Merkle-Damgård" Combiner. Unfortunately, the situation is considerably more involved for the Merkle-Damgård-friendly construction given in Equation (2). The key challenge is that it is no longer the case that $h_1(M, \mathcal{Z}_1)$ is independent of the adversarial hash function g. In fact, it is possible for the (real-world) distinguisher to find some pair of messages M, M' such that $(g^{h_1})^*(M', \mathcal{Z}_2) = h_1^*(M, \mathcal{Z}_1)$.

In particular, consider $M' = (h_1^*(M, \mathcal{Z}_1^{(1)}, \ldots, \mathcal{Z}_1^{(k-1)}), \mathcal{Z}_1^{(k)})$, where k is the block-length of $\mathcal{Z}_1$. That is, M' is the value of computing all but the last step of $h_1^*(M, \mathcal{Z}_1)$, followed by the last block of $\mathcal{Z}_1$. By definition of Merkle-Damgård, $h_1^*(M, \mathcal{Z}_1) = h_1(h_1^*(M, \mathcal{Z}_1^{(1)}, \ldots, \mathcal{Z}_1^{(k-1)}), \mathcal{Z}_1^{(k)}) = h_1(M')$, and so it is not difficult to define g such that $(g^{h_1})^*(M', \mathcal{Z}_2) = h_1(M') = h_1^*(M, \mathcal{Z}_1)$.

Defining a simulator. In order to achieve combiner-consistency for $\widetilde{C}$, we will need the simulator controlling h_1 to define it to satisy this equation:

$$h_1^*(M, \mathcal{Z}_1) = H(M) \oplus (g^{h_1})^*(M, \mathcal{Z}_2)$$

The first challenge we need to resolve is to figure out how the simulator can recognize when the last query needed to compute $h_1^*(M, \mathcal{Z}_1)$ is made. The idea here is the same idea that is used in most other length-extension indifferentiability proofs [9, 2]. Namely, the simulator will simply track all queries made to it. If a query $h_1(a, b)$ is really equivalent to completing $h_1^*(M, \mathcal{Z}_1)$, then there should be a record in the simulator of queries to all the previous rounds. That is, for each query (a, b), the simulator will track the path of messages $m_1, \ldots, m_t$ such that $h_1(a, b) = h_1^*(m_1, \ldots, m_t)$. With some work, we can show that this tracking is sufficient to recognize all queries of the form $h_1^*(M, \mathcal{Z}_1)$.

Bounding the runtime of the simulator. The second major problem is that our simulator is recursive. It is not clear that it will terminate quickly. The main challenge of this argument is to give a runtime bound on our simulator. We will show that the number of recursive calls to the simulator can be bounded by the number of queries made to the distinguisher.

A weaker separation property. In order to achieve this bound, we will show a weaker separation property than the claim that $(g^{h_1})^*(M, \mathcal{Z}_2)$ never computes $h_1^*(M', \mathcal{Z}_1)$. In particular, we will show that if you can find an M such that $(g^{h_1})^*(M, \mathcal{Z}_2)$ queries $h_1^*(M', \mathcal{Z}_1)$, then you must have queried $h_1^*(M')$ directly to the simulator when figuring out M. Note that this means that calling

$(g^{h_1})^*(M, \mathcal{Z}_2)$ cannot recurse more times than queries made to h_1, and so the simulator we defined earlier terminates quickly.

Looking at the real world, where h_1 is a true random oracle, this property is not too hard to see. Assume towards contradiction that M is such that $(g^{h_1})^*(M, \mathcal{Z}_2)$ computes $h_1^*(M', \mathcal{Z}_1)$, but M is chosen without querying $h_1^*(M', \mathcal{Z}_1)$. We can observe that the entire process of choosing M and running g must query each round of $h_1^*(M', \mathcal{Z}_1)$ in order. But since M is chosen without querying $h_1^*(M', \mathcal{Z}_1)$, this means that $(g^{h_1})^*(M, \mathcal{Z}_2)$ queries the last k rounds of $h_1^*(M', \mathcal{Z}_1)$, where k is the "block-length" of $\mathcal{Z}_1$. And so looking at the queries made by $(g^{h_1})^*(M, \mathcal{Z}_2)$ completely reveals M'. This means that M alone is enough information to describe $\mathcal{Z}_1$. But this is impossible, since $|M| \ll |\mathcal{Z}_1|$.

However, the situations gets significantly more complicated when operating in the ideal world. In the ideal world, whenever $h_1^*(M, \mathcal{Z}_1)$ is queried, $(g^{h_1})^*(M, \mathcal{Z}_2)$ is called, which may itself compute $h_1^*(M')$ without revealing it to h_1. Thus, the argument before breaks down, since the process of computing M may implicitly compute $h_1^*(M')$ before querying $(g^{h_1})^*(M, \mathcal{Z}_2)$. Furthermore, adversaries in the ideal world also have oracle access to H as well as the simulator h_1. In particular, this means that whenever the adversary makes a query computing $h_1^*(M, \mathcal{Z}_1)$, they get the following equation for free:

$$(g^{h_1})^*(M, \mathcal{Z}_2) = h_1^*(M, \mathcal{Z}_1) \oplus H(M)$$

For simplicity, we will first consider only adversaries in the real world which only have oracle access to the simulator h_1, but not H. We will show that every recursive call made by the simulator must be made on a message M for which $h_1^*(M)$ has already been queried.

Coloring queries. To illustrate this claim, we will color all queries made to the simulator h_1. Any query made directly we will color *blue*, and any query made indirectly via a call to $(g^{h_1})^*(M, \mathcal{Z}_2)$ we will color *orange*. When we consider a particular recursive call, we will color queries made during that call *green*. For an example, see Figure 1.

The outline of the argument is that we will prove three properties about the coloring of the queries made to the simulator, and these three properties will together imply the result. We represent each of these three properties visually in Figure 2.

First, no blue query can ever follow an orange query. The reason this is the case is intuitively because we can generate the results of all blue queries without making any orange queries. Thus, the only way for a blue query to follow an orange query is if the random output of some orange query collides with one of the choices of input for a blue query.

Second, let us consider any recursive call made for a message M, where M has a path marked entirely in blue in the state of the simulator. Let us mark all queries made directly by that recursive call green. Then no green query can ever follow an orange query. The reasoning is the same as before. Since the recursive call is made from a blue query, the corresponding message M can be revealed by just looking at the blue queries. Thus, we can discover the queries made by $(g^{\mathsf{Sim}})^*(M, \mathcal{Z}_2)$ without making any orange queries.

Finally, we cannot have a sequence of green queries which correspond to the last k blocks of $h_1^*(M', \mathcal{Z}_1)$. We show this using a compression argument. Let M be the message queried to $(g^{\mathsf{Sim}})^*(M, \mathcal{Z}_1)$ which triggered this sequence of green queries. Observe that if we know M, we can recover the entire list of green queries by running $(g^{h_1})^*(M, \mathcal{Z}_2)$. If in addition we know the index of the last query made for $h_1^*(M', \mathcal{Z}_1)$, we can follow the trail backwards to get the indices for all queries of the last k blocks of $h_1^*(M', \mathcal{Z}_1)$. But the description of M as well as the index of the final

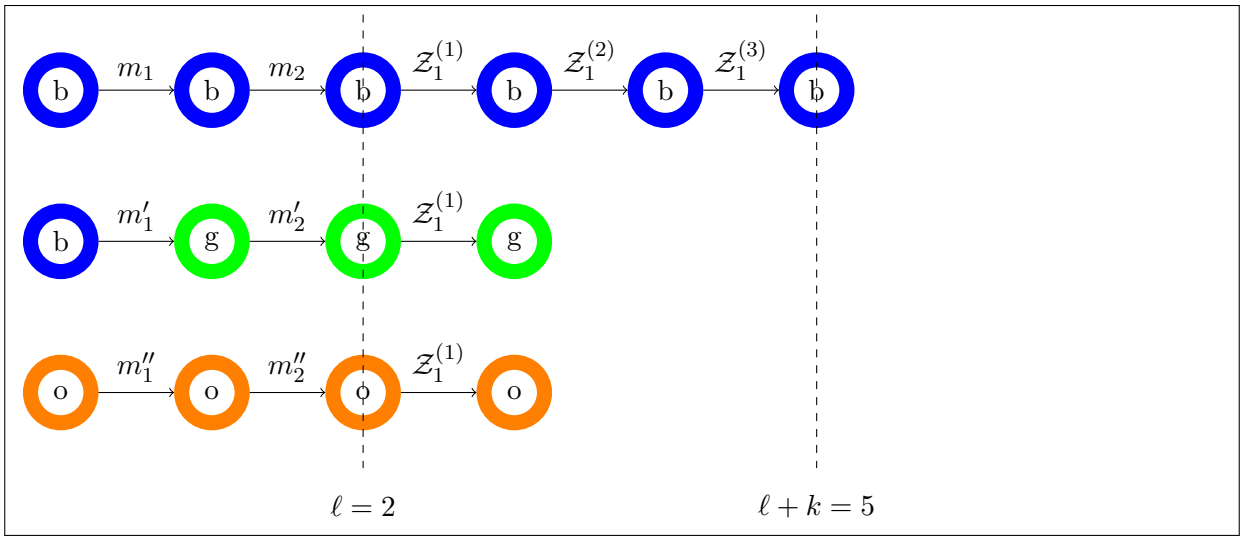

Figure 1: An example illustration of the simulator state with $\ell = 2, k = 3$. Blue nodes refer to direct queries, orange nodes refer to recursive queries, and green nodes refer to queries made in the current recursive call. Here, the distinguisher has called the simulator directly on $h_1(0, m_1')$ as well as $h_1^*(m_1, m_2, \mathcal{Z}_1)$, which has triggered $(g^{h_1})^*(m_1, m_2, \mathcal{Z}_2)$ to query $h_1^*(m_1', m_2', \mathcal{Z}_1^{(1)})$. Earlier, some other recursive call has queried $h_1^*(m_1'', m_2'', \mathcal{Z}_1^{(1)})$.

query is of length $|M| + \log_2(T_{g^*})$, where T_{g^*} is the number of h_1 oracle calls inside g^*. This is less than $|\mathcal{Z}_1|$, meaning that such situation is impossible.

We now can see that if a recursive call $(g^{h_1})^*(M, \mathcal{Z}_2)$ is triggered on a message M, then M must have a path marked entirely in blue. If not, consider the first such recursive call. If it is triggered by a blue query, then by the first property the entire path corresponding to $(M, \mathcal{Z}_1)$ must be blue, and so we are good. If it is triggered by a green query, then by the second and third properties, there must be some blue query along the last k nodes of the path corresponding to $(M, \mathcal{Z}_1)$. And so by the first property, M must have a path marked entirely in blue. Since the number of blue messages is bounded by the number of queries made by the distinguisher, we can bound the number of recursive calls by the number of queries made by the distinguisher.

Extending to distinguishers with access to H. The key idea here is that by construction, $H(M) = \mathsf{Sim}^*(M, \mathcal{Z}_1) \oplus (g^{h_1})^*(M, \mathcal{Z}_2)$. Thus, we can take any distinguisher with oracle access to H, and simulate its oracle queries $H(M)$ by querying $h_1^*(M, \mathcal{Z}_1) \oplus (g^{h_1})^*(M, \mathcal{Z}_2)$. Thus, since we know a distinguisher only querying h_1 can't cause h_1 to make more recursive queries than direct queries, we also get a bound on distinguishers with oracle access to both h_1 and H.

1.2 Related Work

As mentioned before, our work directly answers open questions from the work of [12] on random oracle combiners. This also puts us in a long line of work on combiners, initiated by [20] and [19], and continuing through works such as [3, 18, 19, 20, 23]. In particular, combiners for collision-resistance were studied by [6, 8, 26, 27], including their limitations of doubling the output length. Multi-property preserving combiners, have been defined and constructed by [16, 17]. However, these

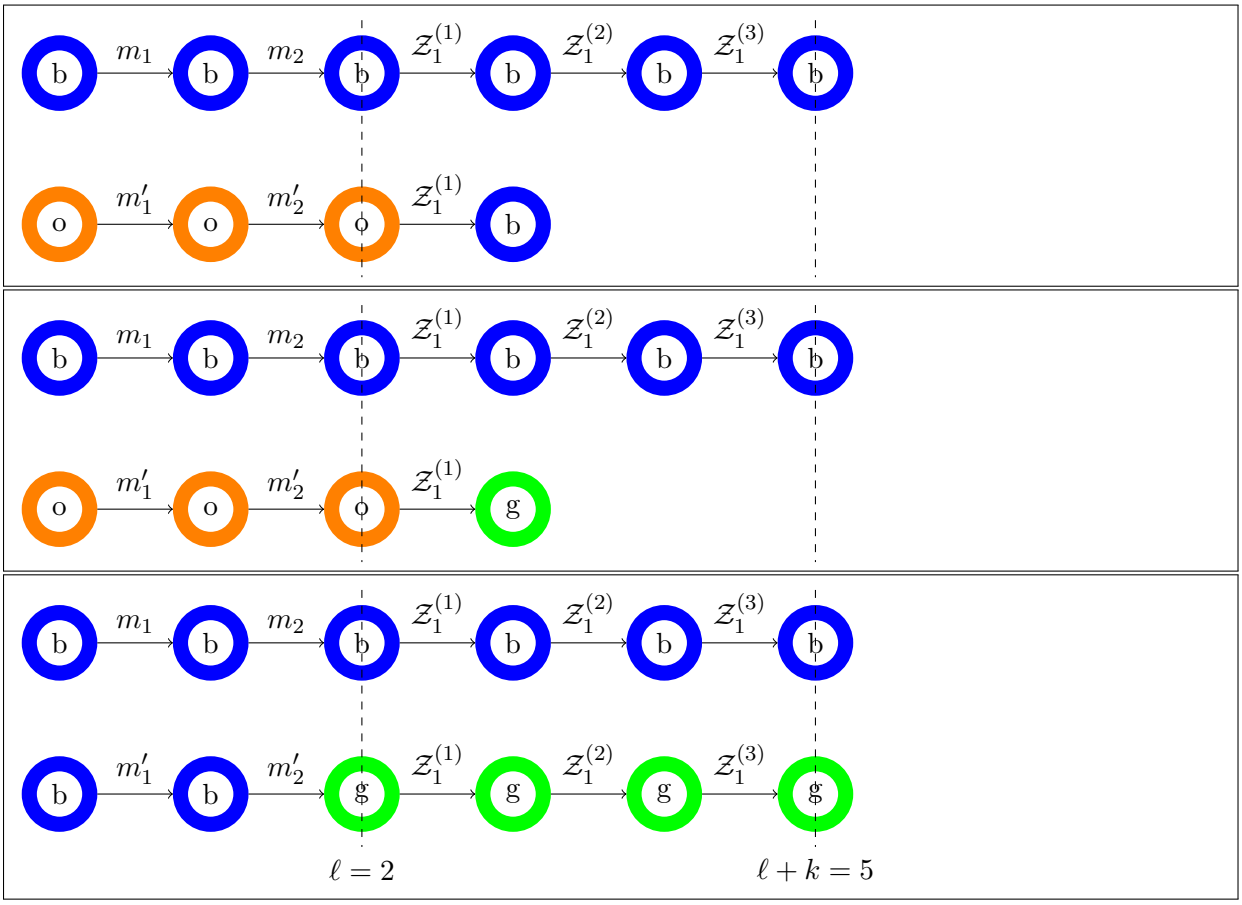

Figure 2: Illustration of the three impossible situations for the simulator state. The first represents no blue query following an orange query. The second represents no green query coming from a message M whose path is entirely blue following an orange query. The third represents no sequence of green queries containing all of $\mathcal{Z}_1$.

also face the same lower bounds as [6, 27] (one of the multi-property being collision-resistance), so they necessarily have long output length. They also preserve indifferentiability from a random oracle, but in a much weaker model than the one from [12].

Mittelbach [25] considered a similar notion to ROCs called "cryptophia short combiners" (additionally, a follow up work [24] identifies and fixes a flaw in the original paper). Mittelbach [25] 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 [6, 26, 27]) 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.

Finally, we mention several papers on proving indifferentiability of various constructions [10, 14, 1, 13, 11], where complex combinatorial and probabilistic arguments needed to be made in order to ensure the termination and/or security of a given indifferentiability simulator.

2 Preliminaries

2.1 Random Oracles and Indifferentiability

This work is concerned with proving indifferentiability from a random oracle, using the indifferentiability framework due to Maurer, Renner, and Holenstein.

Definition 2.1 (Maurer, Renner, Holenstein 2003)**.** *Let $\mathcal{F}, \mathcal{G}$ be two ideal primitives, and let $\mathcal{S}^{\mathcal{F}}_{\mathcal{Z}}$ be a family of constructions of $\mathcal{G}$ from $\mathcal{F}$. We say that $\mathcal{S}$ is $(T_1, T_2, T_{\mathsf{Sim}}, \epsilon)$-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 oracle algorithms $\mathcal{D}^{\mathcal{O}_1,\mathcal{O}_2}$ making at most T_1, T_2 queries respectively to each of its oracles,

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

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

We give the definition of a random oracle, along with some useful facts about the random oracle that we will make use of.

Definition 2.2. *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.*

Lemma 2.3 (Birthday Bound)**.** *Let $h : \{0,1\}^{n+\delta} \to \{0,1\}^n$ be a random oracle. For any algorithm $\mathcal{D}$ making at most T queries to h, the probability that $\mathcal{D}$ produces two inputs (x_1, x_2) such that $h(x_1) = h(x_2)$ is $\leq \frac{T^2}{2^n}$.*

Lemma 2.4 (Guessing Lemma)**.** *Let $h : \{0,1\}^{n+\delta} \to \{0,1\}^n$ be a random oracle. For any algorithm $\mathcal{D}$ making at most T queries to h, the probability that $\mathcal{D}$ produces an input output pair (x, y) such that $h(x) = y$ without querying $h(x)$ is $\leq \frac{1}{2^n}$.*

Proof. This immediately follows from the lazy sampling technique. If $\mathcal{D}$ does not query $h(x)$, then the value of $h(x)$ is uniformly random, and so the probability that $\mathcal{D}$ guesses it correctly is 2^{-n}. □ □

$\underline{REAL_b}$:
Sample $\mathcal{Z} \xleftarrow{\$} \{0,1\}^k$
Sample $h : \{0,1\}^{n+\delta} \to \{0,1\}^n$ 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 $C_{\mathcal{Z}}^{g^{\mathcal{O}_1},\mathcal{O}_1}(M)$.

$\underline{IDEAL(C)}$:
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 3: We say that the combiner C is $(T_1, T_2, T_g, T_{\mathsf{Sim}})$-secure if for all $b \in \{0,1\}$, circuits $\mathcal{D}^{\mathcal{O}_1,\mathcal{O}_2}, g^{\mathcal{O}_1}$ with $\mathcal{D}$ making at most T_1 queries to $\mathcal{O}_1$, T_2 queries to T_2 and g making at most T_g queries to $\mathcal{O}_1$, there exists a simulator $\mathsf{Sim}^{\mathcal{O}_2}$ such that $|\Pr[\mathcal{D}(REAL_b) \to 1] - \Pr[\mathcal{D}(IDEAL) \to 1]| \leq \epsilon$. (Reproduced from [12])

Finally, we present the indifferentiability-based definition of Random Oracle combiners, as they appeared in [12].

Definition 2.5 (Definition 3.3 from [12])**.** *Let $C_{\mathcal{Z}}^{h_1,h_2} : \{0,1\}^m \to \{0,1\}^n$ be a PPT algorithm taking in an input M, a seed $\mathcal{Z}$, and two oracles h_1 and h_2. Let $\mathcal{H} : \{0,1\}^{n+\delta} \to \{0,1\}^n, \mathcal{G} : \{0,1\}^m \to \{0,1\}^n$ be two random oracles. We say that C is a $\underline{(T_g, T_1, T_2, T_{\mathsf{Sim}}, \epsilon)\text{-random oracle combiner}}$ 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 $(T_1, T_2, T_{\mathsf{Sim}}, \epsilon)$-indifferentiable from $\mathcal{G}$.*

An equivalent, game-based formulation is given in Figure 3.

2.2 The Merkle-Damgård Transform

We present the Merkle-Damgård Transform, which allows one to convert a fixed-length hash function into one which takes inputs of arbitrary size by processing the input block by block.

Definition 2.6 (Merkle-Damgård Transform)**.** *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,

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

We will use f^* to refer to Merkle-Damgård applied to f. We will slightly abuse notation and use the same notation even when f is some stateful oracle. That is, if $\mathcal{O}$ is an oracle, we will define

$$\mathcal{O}^*(x) \coloneqq \mathcal{O}(x_\ell, \ldots, \mathcal{O}(x_1, 0) \ldots),$$

where $\mathcal{O}(x_1, 0)$ is performed before $\mathcal{O}(x_2, \mathcal{O}(x_1, 0))$, and so on.

3 Our Main Result

We now have the background to present the main result of the work, Theorem 3.1.

Theorem 3.1. *Let $h_1, h_2 : \{0,1\}^{n+\delta} \to \{0,1\}^n$ be two oracles. Let $k > \ell$ be integers and let $C^{h_1,h_2} : \{0,1\}^{(\ell+k)\delta} \to \{0,1\}^n$ be defined as follows:*

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

Then for all T_g, T_1, T_2, C is a $(T_g, T_1, T_2, T_{\mathsf{Sim}}, \epsilon)$ random oracle combiner with

$$\epsilon \leq O\left(\frac{(T_1 T_g(\ell+k) + T_2 T_g^2(\ell+k)^2)^2}{2^n} + \frac{T_g(\ell+k)}{2^{(k-\ell)\delta}}\right)$$

$$T_{\mathsf{Sim}} \leq O\left(T_1 T_g(\ell+k) + T_2 T_g^2(\ell+k)^2\right)$$

To prove this theorem, it suffices to show that for $g^{\mathcal{O}}$ any oracle circuit and $h_1 : \{0,1\}^{n+\delta} \to \{0,1\}^n$ a random oracle, $C^{h_1,g^{h_1}}$ is indifferentiable from a bigger random oracle $H : \{0,1\}^{(\ell+k)\delta} \to \{0,1\}^n$. Since the construction is symmetric, we then immediately get that $C^{g^h_1,h_1}$ is also indifferentiable from H.

Description of the simulator We define a simulator for the indifferentiability game in Figure 4. The simulator tracks all queries made to it in T. The purpose of this is so that if the adversary queries

$$\mathsf{Sim}^*(X) = \mathsf{Sim}(\mathsf{Sim}(\ldots \mathsf{Sim}(\mathsf{Sim}(0, x_1), x_2), \ldots), x_\ell)$$

then the simulator should be able to recover X from the input to the last query $(\mathsf{Sim}(\ldots \mathsf{Sim}(\mathsf{Sim}(0, x_1), x_2), \ldots), x_\ell)$. This is formalized through the subroutine $TPath$. In particular, if (a, b) is the last query made in computing $\mathsf{Sim}^*(X)$, then it should be the case that $TPath(X) = (a, b)$.

Recall that for consistency the simulator should satisfy the property

$$H(M) = (\mathsf{Sim}^H)^*(M,\mathcal{Z}_1) \oplus (g^{\mathsf{Sim}^H})^*(M,\mathcal{Z}_2)$$

for messages M which are ℓ blocks long. In other words, we need

$$(\mathsf{Sim}^H)^*(M,\mathcal{Z}_1) = H(M) \oplus (g^{\mathsf{Sim}^H})^*(M,\mathcal{Z}_2)$$

So there are two cases the simulator needs to handle.

On input (a, b) such that $TPath(a, b) \neq (M, \mathcal{Z}_1)$ for an ℓ-block long M, $\mathsf{Sim}^H(a, b)$ should just act like a random oracle. Thus, in this case the simulator outputs a random string (and stores it in T to maintain consistency).

On input (a, b) such that $TPath(a, b) = (M, \mathcal{Z}_1)$ for an ℓ-block long M, $\mathsf{Sim}^H(a, b)$ explicitly computes $H(M) \oplus (g^{\mathsf{Sim}^H})^*(M, \mathcal{Z}_2)$ and outputs that. Note that this may cause recursive queries, and so we also include a safeguard limiting the number of recursive calls to some value t_{max}, which we will set later.

$\underline{\mathsf{Sim} - INIT}$:
Initialize $T = []$.
Initialize $rcount = 0$.

$\underline{\mathsf{Sim}^{\mathcal{O}}(a, b)}$:
If $(a, b) \in T$, output $T[a, b]$.
$rcount \leftarrow rcount + 1$.
If $rcount > t_{max}$, terminate.
Set $X \leftarrow Tpath(a, b)$.
If $X = (M, \mathcal{Z}_1)$ with $|M| = \ell \cdot \delta$:
-Set $T[a, b] \leftarrow H(M) \oplus (g^{\mathcal{O}_1})^*(M, \mathcal{Z}_2)$.
Else: set $T[a, b] \xleftarrow{\$} \{0, 1\}^n$.
Output $T[a, b]$.

Subroutine $\underline{Tpath(a, b)}$:
Let $(a_1, b_1), \ldots, (a_s, b_s)$ be the longest sequence in T such that
$-a_1 = 0$
$-a_i = T[a_{i-1}, b_{i-1}]$ for $i > 1$
$-a = T[a_s, b_s]$
Output $b_1 || \ldots || b_s || b$

Figure 4: Our simulator for the indifferentiability game.

3.1 The hybrid argument

For the ease of notation, throughout the rest of this paper T_{g^*} will be $(\ell + k)T_g$, a bound on the number of queries made by g during any call to $(g^{(\cdot)})^*$.

We will prove this indifferentiability through a sequence of hybrids. In particular, for each indifferentiability adversary $\mathcal{D}^{\mathcal{O}_1, \mathcal{O}_2}$, we define a sequence of games $G0, G1, \ldots, G8, Gf$. $G0$ will be REAL the "real" game from Theorem 2.5, where $\mathcal{O}_1$ is a random oracle h and $\mathcal{O}_2$ is the construction $C^{h, g^h}_{\mathcal{Z}_1, \mathcal{Z}_2} = h^*(M, \mathcal{Z}_1) \oplus (g^h)^*(M, \mathcal{Z}_2)$. Gf will be IDEAL, the "ideal" game from Theorem 2.5 with $\mathcal{O}_2$ a random oracle H and $\mathcal{O}_1$ the simulator described above Sim^H. To complete the argument, we will bound for each i,

$$|\Pr[Gi \to 1] - \Pr[G(i-1) \to 1]| .$$

These hybrids are detailed in Figures 5, 6 and 8. Note that in each hybrid, $TPath$ refers to the subroutine defined in Figure 4.

Lemma 3.2. $\mathsf{REAL} \equiv G1$

Proof. Note that $G1$ acts identically to REAL, but performs extra computations which are not returned by the oracles. Because of the recursion counter, $G1$ can never enter an infinite loop. Thus, it is clear that the oracles behave identically in both games. □ □

Lemma 3.3. $|\Pr[G1 \to 1] - \Pr[G2 \to 1]| \leq \Pr[G2 \to \perp] \leq \frac{t_{max}(t_{max}+1)}{2^n}$.

REAL:

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

$\mathcal{O}_1(a, b)$:
Output $h(a, b)$

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

(a) The real game, where $\mathcal{O}_1$ is a wrapper for the small random oracle and $\mathcal{O}_2$ is a wrapper for our combiner construction.

G1:

Sample $Z_1, Z_2 \xleftarrow{\$} \{0,1\}^{k\delta}$
Sample $h : \{0,1\}^{n+\delta} \to \{0,1\}^n$ u.a.r.
Initialize $T = []$, $rcount = 0$.
Run $\mathcal{D}^{\mathcal{O}_1,\mathcal{O}_2}(\mathcal{Z}_1, \mathcal{Z}_2) \to b'$
Output b'.

$\mathcal{O}_1(a, b)$:
If $(a, b) \in T$, output $T[a, b]$.
$rcount \leftarrow rcount + 1$.
Set $X \leftarrow Tpath(a, b)$.
If $X = (M, \mathcal{Z}_1)$ with $|M| = \ell \cdot \delta$:
  -If $rcount \leq t_{max}$, call $(g^{\mathcal{O}_1})^*(M, \mathcal{Z}_2)$.
Set $T[a, b] \leftarrow h(a, b)$.
Output $T[a, b]$.

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

(b) The first hybrid, where we track queries made to $\mathcal{O}_1$ in T. When a path is of the form $(M, \mathcal{Z}_1)$ for an ℓ block long M, we additionally call $(g^{\mathcal{O}_1})^*(M, \mathcal{Z}_2)$, although we do nothing with it. We also add a safeguard so that this does not infinitely recurse.

G2:

Sample $Z_1, Z_2 \xleftarrow{\$} \{0,1\}^{k\delta}$
Sample $h : \{0,1\}^{n+\delta} \to \{0,1\}^n$ u.a.r.
Initialize $T = []$, $rcount = 0$.
Run $\mathcal{D}^{\mathcal{O}_1,\mathcal{O}_2}(\mathcal{Z}_1, \mathcal{Z}_2) \to b'$
If flag $FAIL$ is set, output $\perp$.
Output b'.

$\mathcal{O}_1(a, b)$:
If $(a, b) \in T$, output $T[a, b]$.
$rcount \leftarrow rcount + 1$.
Set $X \leftarrow Tpath(a, b)$.
If $X = (M, \mathcal{Z}_1)$ with $|M| = \ell \cdot \delta$:
  -If $rcount \leq t_{max}$, call $(g^{\mathcal{O}_1})^*(M, \mathcal{Z}_2)$.
Set $T[a, b] \leftarrow h(a, b)$.
If $rcount \leq t_{max}$:
  -If $T[a, b] = 0$, set flag $FAIL$.
  -If there exists $(a', b') \in T$ such that $T[a, b] = a'$ or $T[a, b] = T[a', b']$, set flag $FAIL$.
Output $T[a, b]$.

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

(c) The second hybrid, where we add consistency checks to T. If any checks fail, we force the game to output $\perp$.

G3:

Sample $Z_1, Z_2 \xleftarrow{\$} \{0,1\}^{k\delta}$
Sample $h : \{0,1\}^{n+\delta} \to \{0,1\}^n$ u.a.r.
Initialize $T = []$, $rcount = 0$.
Run $\mathcal{D}^{\mathcal{O}_1,\mathcal{O}_2}(\mathcal{Z}_1, \mathcal{Z}_2) \to b'$
If flag $FAIL$ is set, output $\perp$.
Output b'.

$\mathcal{O}_1(a, b)$:
If $(a, b) \in T$, output $T[a, b]$.
$rcount \leftarrow rcount + 1$.
If $rcount > t_{max}$, terminate and set flag $FAIL$.
Set $X \leftarrow Tpath(a, b)$.
If $X = (M, \mathcal{Z}_1)$ with $|M| = \ell \cdot \delta$:
  -Call $(g^{\mathcal{O}_1})^*(M, \mathcal{Z}_2)$.
Set $T[a, b] \leftarrow h(a, b)$.
If $T[a, b] = 0$, set flag $FAIL$.
If there exists $(a', b') \in T$ such that $T[a, b] = a'$ or $T[a, b] = T[a', b']$, set flag $FAIL$.
Output $T[a, b]$.

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

(d) The third hybrid, where we output $\perp$ if the recursion counter ever exceeds its limits.

Figure 5: The REAL game and Hybrids $G1$, $G2$, and $G3$.

G4:
Sample $Z_1, Z_2 \xleftarrow{\$} \{0,1\}^{k\delta}$
Sample $h : \{0,1\}^{n+\delta} \to \{0,1\}^n$ u.a.r.
Initialize $T = []$, $rcount = 0$.
Run $\mathcal{D}^{\mathcal{O}_1,\mathcal{O}_2}(\mathcal{Z}_1, \mathcal{Z}_2) \to b'$
If flag $FAIL$ is set, output $\perp$, else b'.

$\mathcal{O}_1(a, b)$:
If $(a, b) \in T$, output $T[a, b]$.
$rcount \leftarrow rcount + 1$.
If $rcount > t_{max}$, terminate and set flag $FAIL$.
Set $X \leftarrow Tpath(a, b)$.
If $X = (M, \mathcal{Z}_1)$ with $|M| = \ell \cdot \delta$:
-Call $(g^{\mathcal{O}_1})^*(M, \mathcal{Z}_2)$.
-Sample $y \xleftarrow{\$} \{0,1\}^n$.
-Set $T[a, b] \leftarrow y$.
Else: set $T[a, b] \leftarrow h(a, b)$.
If $T[a, b] = 0$, set flag $FAIL$.
If there exists $(a', b') \in T$ such that $T[a, b] = a'$ or $T[a, b] = T[a', b']$, set flag $FAIL$, else output $T[a, b]$.

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

(a) The fourth hybrid, where we sample the output of $\mathcal{O}_1$ uniformly (and store it in T).

G5:
Sample $Z_1, Z_2 \xleftarrow{\$} \{0,1\}^{k\delta}$
Sample $h : \{0,1\}^{n+\delta} \to \{0,1\}^n$ u.a.r.
Initialize $T = []$, $rcount = 0$.
Run $\mathcal{D}^{\mathcal{O}_1,\mathcal{O}_2}(\mathcal{Z}_1, \mathcal{Z}_2) \to b'$
If flag $FAIL$ is set, output $\perp$, else b'.

$\mathcal{O}_1(a, b)$:
If $(a, b) \in T$, output $T[a, b]$.
$rcount \leftarrow rcount + 1$.
If $rcount > t_{max}$, terminate and set flag $FAIL$.
Set $X \leftarrow Tpath(a, b)$.
If $X = (M, \mathcal{Z}_1)$ with $|M| = \ell \cdot \delta$:
-Sample $y \xleftarrow{\$} \{0,1\}^n$.
-Set $T[a, b] \leftarrow y \oplus (g^{\mathcal{O}_1})^*(M, \mathcal{Z}_2)$.
Else: set $T[a, b] \leftarrow h(a, b)$.
If $T[a, b] = 0$, set flag $FAIL$.
If there exists $(a', b') \in T$ such that $T[a, b] = a'$ or $T[a, b] = T[a', b']$, set flag $FAIL$, else output $T[a, b]$.

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

(b) The fifth hybrid, where we $\oplus$ the output of recursive calls with $(g^{\mathcal{O}_1})^*(M, \mathcal{Z}_2)$.

Figure 6: Hybrids $G4$ and $G5$.

G6:
Sample $Z_1, Z_2 \xleftarrow{\$} \{0,1\}^{k\delta}$
Sample $h : \{0,1\}^{n+\delta} \to \{0,1\}^n$ u.a.r.
Sample $H : \{0,1\}^{\ell\cdot\delta} \to \{0,1\}^n$ u.a.r.
Initialize $T = []$, $rcount = 0$.
Run $\mathcal{D}^{\mathcal{O}_1,\mathcal{O}_2}(\mathcal{Z}_1, \mathcal{Z}_2) \to b'$
If flag $FAIL$ is set, output $\perp$, else b'.

$\mathcal{O}_1(a, b)$:
If $(a, b) \in T$, output $T[a, b]$.
$rcount \leftarrow rcount + 1$.
If $rcount > t_{max}$, terminate and set flag $FAIL$.
Set $X \leftarrow Tpath(a, b)$.
If $X = (M, \mathcal{Z}_1)$ with $|M| = \ell \cdot \delta$:
-Set $T[a, b] \leftarrow H(M) \oplus (g^{\mathcal{O}_1})^*(M, \mathcal{Z}_2)$.
Else: set $T[a, b] \leftarrow h(a, b)$.
If $T[a, b] = 0$, set flag $FAIL$.
If there exists $(a', b') \in T$ such that $T[a, b] = a'$ or $T[a, b] = T[a', b']$, set flag $FAIL$, else output $T[a, b]$.

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

(a) The sixth hybrid, where we sample all the randomness used for recursive calls in the beginning as the function H.

G7:
Sample $Z_1, Z_2 \xleftarrow{\$} \{0,1\}^{k\delta}$
Sample $h : \{0,1\}^{n+\delta} \to \{0,1\}^n$ u.a.r.
Sample $H : \{0,1\}^{\ell\cdot\delta} \to \{0,1\}^n$ u.a.r.
Initialize $T = []$, $rcount = 0$.
Run $\mathcal{D}^{\mathcal{O}_1,\mathcal{O}_2}(\mathcal{Z}_1, \mathcal{Z}_2) \to b'$
If flag $FAIL$ is set, output $\perp$, else b'.

$\mathcal{O}_1(a, b)$:
If $(a, b) \in T$, output $T[a, b]$.
$rcount \leftarrow rcount + 1$.
If $rcount > t_{max}$, terminate and set flag $FAIL$.
Set $X \leftarrow Tpath(a, b)$.
If $X = (M, \mathcal{Z}_1)$ with $|M| = \ell \cdot \delta$:
-Set $T[a, b] \leftarrow H(M) \oplus (g^{\mathcal{O}_1})^*(M, \mathcal{Z}_2)$.
Else: set $T[a, b] \leftarrow h(a, b)$.
If $T[a, b] = 0$, set flag $FAIL$.
If there exists $(a', b') \in T$ such that $T[a, b] = a'$ or $T[a, b] = T[a', b']$, set flag $FAIL$, else output $T[a, b]$.

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

(b) The seventh hybrid, where $\mathcal{O}_2$ ignores the results of its queries and outputs $H(M)$ directly.

Figure 7: Hybrids $G6$ and $G7$.

G8:
Sample $Z_1, Z_2 \xleftarrow{\$} \{0,1\}^{k\delta}$
Sample $h : \{0,1\}^{n+\delta} \to \{0,1\}^n$ u.a.r.
Sample $H : \{0,1\}^{\ell\cdot\delta} \to \{0,1\}^n$ u.a.r.
Initialize $T = []$, $rcount = 0$.
Run $\mathcal{D}^{\mathcal{O}_1,\mathcal{O}_2}(\mathcal{Z}_1, \mathcal{Z}_2) \to b'$
If flag $FAIL$ is set, output $\perp$, else b'.

$\mathcal{O}_1(a, b)$:
If $(a, b) \in T$, output $T[a, b]$.
$rcount \leftarrow rcount + 1$.
If $rcount > t_{max}$, terminate and set flag $FAIL$.
Set $X \leftarrow Tpath(a, b)$.
If $X = (M, \mathcal{Z}_1)$ with $|M| = \ell \cdot \delta$:
-Set $T[a, b] \leftarrow H(M) \oplus (g^{\mathcal{O}_1})^*(M, \mathcal{Z}_2)$.
Else: set $T[a, b] \leftarrow h(a, b)$.
If $T[a, b] = 0$, set flag $FAIL$.
If there exists $(a', b') \in T$ such that $T[a, b] = a'$ or $T[a, b] = T[a', b']$, set flag $FAIL$, else output $T[a, b]$.

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

(a) The eighth hybrid, where $\mathcal{O}_2$ no longer queries $\mathcal{O}_1^*(M, \mathcal{Z}_1) \oplus (g^{\mathcal{O}_1})^*(M, \mathcal{Z}_2)$.

IDEAL:
Sample $Z_1, Z_2 \xleftarrow{\$} \{0,1\}^{k\delta}$
Sample $H : \{0,1\}^{\ell\cdot\delta} \to \{0,1\}^n$ u.a.r.
$\mathsf{Sim} - INIT$. Run $\mathcal{D}^{\mathcal{O}_1,\mathcal{O}_2}(\mathcal{Z}_1, \mathcal{Z}_2) \to b'$
If flag $FAIL$ is set, output $\perp$, else b'.
$\mathcal{O}_1(a, b)$:
Output $\mathsf{Sim}^{\mathcal{O}_2}(a, b)$.

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

(b) The ideal game, where $\mathcal{O}_1$ is a wrapper for the simulator and $\mathcal{O}_2$ is a wrapper for the big random oracle H.

Figure 8: The IDEAL game and the last hybrid $G8$.

Proof. Consider the situation where $h(a,b)$ is sampled uniformly at random each time it is called for the first time. Then, each time $\mathcal{O}_1$ is called, the probability that $h(a,b) = 0$ or there exists $(a',b') \in T$ such that $h(a,b) = a'$ is $\leq \frac{|T|+1}{2^n}$ by a union bound.

Since this check can be made at most t_{max} times, and for each of these checks, $|T| \leq t_{max}$, by a union bound we have

$$\Pr[G2 \to \perp] \leq \frac{t_{max}(t_{max}+1)}{2^n}.$$

The claim follows from the fact that $G1$ and $G2$ are identical conditioned on $G2$ not outputting $\perp$. □ □

Lemma 3.4. *For $t_{max} \geq T_1 + T_2(\ell + k + T_{g^*})$,*

$$\Pr[G3 \to \perp] \leq \Pr[G2 \to \perp] + \frac{3t_{max}^2}{2^n} + \frac{T_{g^*}}{2^{(\ell-k)\delta}}$$

and

$$|\Pr[G2 \to 1] - \Pr[G3 \to 1]| \leq \frac{3t_{max}^2}{2^n} + \frac{T_{g^*}}{2^{(\ell-k)\delta}}$$

Note that as long as $rcount \leq t_{max}$, the oracles in both games are identical. Thus, it remains to bound $\Pr[G3 \to \perp]$. In particular, this step is essentially equivalent to showing that the simulator terminates before $rcount \leq t_{max}$. This argument goes along the same lines as described in the technical overview. Since this hybrid is formally very technically involved, we delay its proof to Section 3.2, see Theorem 3.13.

Lemma 3.5. $G3 \equiv G4$.

Proof. Note that in $G3$, $h(a,b)$ will only be called exactly once the first time $\mathcal{O}_1(a,b)$ is called. Thus, replacing some of these calls to $h(a,b)$ with uniformly random strings leads to the exact same distributions of oracles. □ □

Lemma 3.6. $G5 \equiv G4$.

Proof. Since XORing by any value is a permutation, sampling a random string is equivalently distributed to XORing a random string to any value. The claim follows. □ □

Lemma 3.7. $G6 \equiv G5$.

Proof. By the definition of $Tpath$, it is clear that for any table T and any message X, there is at most one $(a,b) \in T$ such that $Tpath(a,b) = X$. Thus, $H(M)$ will be used at most once in $G6$, and so is identically distributed to a random value. As the only difference between $G5$ and $G6$ is that a random value is replaced by $H(M)$, the two games are identically distributed. □ □

Lemma 3.8. *Consider any call to $G6$ in which $\perp$ is not set. Let $(a_1,b_1),\ldots,$ $(a_{\ell+k},b_{\ell+k})$ be any path in T such that $a_1 = 0$ and $a_i = T[a_{i-1},b_{i-1}]$ for $i > 1$ Then the first query made to $\mathcal{O}_1(a_{\ell+k},b_{\ell+k})$ triggered the recursive condition with $X = b_1||\ldots b_{\ell+k}$.*

Proof. First, we need to show that at the time $\mathcal{O}_1(a_{\ell+k},b_{\ell+k})$ is queried, $T[a_i,b_i]$ has already been set for $i < \ell + k$. If this is not the case, then there is some i such that $T[a_i,b_i]$ is set before $T[a_{i-1},b_{i-1}]$. But then when $T[a_{i-1},b_{i-1}]$ is set, since $T[a_{i-1},b_{i-1}] = a_i$, the flag $FAIL$ would have been set.

Next, we need to show that there is no other path of length $\geq \ell + k$ ending at $(a_{\ell+k},b_{\ell+k})$. Say there is some other path $(a'_1,b'_1),\ldots,(a'_t,b'_t)$ with

1. $a'_1 = 0$
2. $a'_i = T[a'_{i-1}, b'_{i-1}]$ for $i > 1$
3. $a'_t = a_{\ell+k}$ and $b'_t = b_{\ell+k}$

Since the path is different, there must be some point at which they diverge. That is, there should be some i, j such that

1. $T[a'_i, b'_i] = a_j$
2. Either $j = 1$ or $(a'_i, b'_i) \neq (a_{j-1}, b_{j-1})$

If $j = 1$, then since $a_1 = 0$, we have $T[a'_i, b'_i] = 0$, which would cause $G6$ to fail. If $j \neq 0$, then we have $T[a'_i, b'_i] = T[a_j, b_j]$ with $(a'_i, b'_i) \neq (a_j, b_j)$, which would also cause $G6$ to fail.

Thus, when $\mathcal{O}_1(a_{\ell+k}, b_{\ell+k})$ is called for the first time, the longest path corresponding to $(a_{\ell+k}, b_{\ell+k})$ is $(a_1, b_1), \ldots, (a_{\ell+k-1}, b_{\ell+k-1})$, and so
$Tpath(a_{\ell+k}, b_{\ell+k}) = b_1 || \ldots b_{\ell+k}$. □ □

Lemma 3.9. $|\Pr[G6 \to 1] - \Pr[G7 \to 1]| \leq \Pr[G6 \to \perp]$ *and*
$\Pr[G7 \to \perp] \leq \Pr[G6 \to \perp]$.

Proof. It is sufficient to show that as long as $G6$ does not output $\perp$, every query to $\mathcal{O}_2(M)$ outputs $H(M)$. If this is the case, then as long as $G6$ does not output $\perp$, $G6$ and $G7$ behave identically.

Consider any query to $\mathcal{O}_2(M)$ in $G6$ where $FAIL$ is not set after the end of the query. During this query $\mathcal{O}_1^*(M, \mathcal{Z}_1)$ calls $\mathcal{O}_1$ directly on some inputs $(a_1, b_1), \ldots, (a_{\ell+k}, b_{\ell+k})$ with $a_i = \mathcal{O}_1(a_{i-1}, b_{i-1}) = T[a_{i-1}, b_{i-1}]$ and $a_1 = 0$. Here $b_1 || \ldots || b_{\ell+k} = (M, \mathcal{Z}_1)$

Thus, by Theorem 3.8, $\mathcal{O}_1 * (M, \mathcal{Z}_1) = \mathcal{O}_1(a_{i-1}, b_{i-1})$ triggers the recursive condition with $X = (M, \mathcal{Z}_1)$. And so $\mathcal{O}_1^*(M, \mathcal{Z}_1) = H(M) \oplus (g^{\mathcal{O}_1})^*(M, \mathcal{Z}_2)$.

Since results from $\mathcal{O}_1$ are cached, the second call to $(g^{\mathcal{O}_1})^*(M, \mathcal{Z}_2)$ gives the same response. And so $\mathcal{O}_2(M) = H(M) \oplus (g^{\mathcal{O}_1})^*(M, \mathcal{Z}_2) \oplus (g^{\mathcal{O}_1})^*(M, \mathcal{Z}_2) = H(M)$. □ □

Lemma 3.10. $|\Pr[G7 \to 1] - \Pr[G8 \to 1]| \leq \Pr[G7 \to \perp] + \frac{2t_{max}}{2^n}$ *and*
$\Pr[G8 \to \perp] \leq \Pr[G7 \to \perp] + \frac{2t_{max}}{2^n}$

The intuition behind this hybrid is that by induction, with high probability the responses to queries in $G8$ are the same as those in $G7$. In particular, the set T in $G8$ should be a subset of the set T in $G7$ since strictly fewer calls are made to $\mathcal{O}_1$. However, making this fully formal is fairly technically involved, and so we defer the proof to Appendix A.

Lemma 3.11. $|\Pr[\mathsf{IDEAL} \to 1] - \Pr[G8 \to 1]| \leq \Pr[G8 \to \perp]$.

Proof. The only differences between $G8$ and IDEAL are the following:

1. $h(a, b)$ is sampled only when needed, instead of all at the beginning.
2. IDEAL can not output $\perp$.

Runtime:
Sample $Z_1, Z_2 \xleftarrow{\$} \{0,1\}^{k\delta}$
Sample $h : \{0,1\}^{n+\delta} \to \{0,1\}^n$ u.a.r.
Initialize $T = []$, $rcount = 0$.
Run $\mathcal{D}^{\mathcal{O}}(\mathcal{Z}_1, \mathcal{Z}_2)$
Output 0.

$\mathcal{O}(a, b)$:
If $(a, b) \in T$, output $T[a, b]$.
$rcount \leftarrow rcount + 1$.
If $rcount > t_{max}$, end the game with output 1.
Set $X \leftarrow Tpath(a, b)$.
If $X = (M, \mathcal{Z}_1)$ with $|M| = \ell \cdot \delta$:
-Call $(g^{\mathcal{O}})^*(M, \mathcal{Z}_2)$.
Set $T[a, b] \leftarrow h(a, b)$.
If $T[a, b] = 0$, end the game and output 0.
If there exists $(a', b') \in T$ such that $T[a, b] = a'$ or $T[a, b] = T[a', b']$, end the game and output 0, else output $T[a, b]$.

Subroutine $Tpath(a, b)$:
Let $(a_1, b_1), \ldots, (a_s, b_s)$ be the longest sequence in T such that
$-a_1 = 0$
$-a_i = T[a_{i-1}, b_{i-1}]$ for $i > 1$
$-a = T[a_s, b_s]$
Output $b_1 || \ldots || b_s || b$

Figure 9: The game for which we will bound runtime.

Note that the first difference clearly leads to identical distributions, since it doesn't matter when the values of h are sampled as long as $h(a, b)$ is sampled by the time (a, b) is queried.

Thus, the games are equivalently distributed conditioned on $G8$ not outputting $\perp$. But we know that

$$\Pr[G8 \to \perp] \leq \frac{t_{max}(t_{max}+1)}{2^n} + \frac{3t_{max}^2}{2^n} + \frac{T_{g^*}}{2^{(\ell-k)\delta}} + \frac{2t_{max}}{2^n}$$

And so the claim follows. □ □

Putting this all together, we get that for $t_{max} \geq (T_1 + T_2(\ell + k + T_{g^*})) \cdot (T_{g^*} + 1)$

$$|\Pr[\mathsf{REAL} \to 1] - \Pr[\mathsf{IDEAL} \to 1]| = O\left(\frac{t_{max}^2}{2^n} + \frac{T_{g^*}}{2^{(\ell-k)\delta}}\right)$$

which concludes the proof of Theorem 3.1.

3.2 Runtime Proof

Our goal is to show that for all $\mathcal{D}^{\mathcal{O}_1, \mathcal{O}_2}$, with all but negligible probability, $G3$ from Figure 5d never overflows its recursion counter. To show this, we will consider a slightly simplified game, defined in Figure 9. In the simplified game, we remove $\mathcal{O}_2$ and we say that the game outputs 1 if and only if the recursion counter overflows.

To describe the runtime game directly, we give $\mathcal{D}$ oracle access to a random function h through a wrapper oracle $\mathcal{O}$. $\mathcal{O}$ will then keep track of all queries made to it. Whenever there is a path in the table of queries corresponding to a message $X = M||\mathcal{Z}_1$, then we run $(g^{\mathcal{O}})^*(M, \mathcal{Z}_2)$, allowing g to call $\mathcal{O}$ (which may then recursively call $g^{\mathcal{O}}$ again). In addition, we keep a recursion counter to track how many times $\mathcal{O}$ is called.

We will prove that the probability the recursion counter overflows in this simplified game is negligible.

Lemma 3.12 (Key Lemma)**.** *For all circuits g and adversaries $\mathcal{D}$ making at most T_1 queries to $\mathcal{O}$, as long as $t_{max} \geq T_1 \cdot (T_{g^*} + 1)$,*

$$\Pr[\mathsf{Runtime} \to 1] \leq \frac{t_{max}^2}{2^n} + \frac{2 \cdot T_1 \cdot t_{max} \cdot T_{g^*}}{2^n} + \frac{T_{g^*}}{2^{(\ell-k)\delta}},$$

where Runtime *is as in Figure 9.*

Corollary 3.13 follows naturally.

Corollary 3.13. *Let $t_{max} = (T_1 + T_2(\ell + k + T_{g^*}))(T_{g^*} + 1)$. Then,*

$$\Pr[rcount \geq t_{max} \text{ in } G3] \leq \frac{3t_{max}^2}{2^n} + \frac{T_{g^*}}{2^{(\ell-k)\delta}}$$

Proof. Let $\mathcal{D}^{\mathcal{O}_1,\mathcal{O}_2}$ be an adversary for $G3$ making T_1 queries to $\mathcal{O}_1$ and T_2 queries to $\mathcal{O}_2$. We define $\widetilde{\mathcal{D}}^{\mathcal{O}}$ to run $\mathcal{D}$ but replacing all $\mathcal{O}_1$ queries by $\mathcal{D}$ with a $\mathcal{O}$ query and replacing all $\mathcal{O}_2$ queries by $\mathcal{D}$ with $\mathcal{O}^*(M, \mathcal{Z}_1) \oplus (g^{\mathcal{O}})^*(M, \mathcal{Z}_2)$. It is clear that

$$\Pr[rcount \geq t_{max} \text{ in } G3(\mathcal{D})] \leq \Pr[\mathsf{Runtime}(\mathcal{D}') \to 1]$$

Furthermore, $\mathcal{D}'$ makes $\leq T_1 + T_2(\ell + k + T_{g^*})$ queries to $\mathcal{O}$.

The corollary follows. □ □

We then proceed to the proof of the key lemma.

To illustrate our argument, we assign colors to the truth table of $h(a, b)$ defined during the duration of the game as follows:

1. If $h(a, b)$ was not queried at all during the game, color it gray.
2. If $h(a, b)$ was queried by a non-recursive call to $\mathcal{O}$, color it blue.
3. If $h(a, b)$ was queried to $\mathcal{O}$, but only during recursive calls, color it orange.

Furthermore, we call a message X of arbitrary length blue if there exists $(a, b) \in T$ such that $Tpath(a, b) = X$ and for each (a_i, b_i) in the corresponding path, (a_i, b_i) is blue. That is, X is blue if there exists a path of only blue queries reconstructing X.

The main idea of our argument is to show that with all but negligible probability, if $(g^{\mathcal{O}})^*(M, \mathcal{Z}_2)$ were to recurse when called on a blue message M, then the recursive calls $(g^{\mathcal{O}})^*(M', \mathcal{Z}_2)$ will be made only on blue messages M'. We will then show that with all but negligible probability, an honest query to $\mathcal{O}$ will only recurse on a message M if M is blue. Together, this implies that

the only recursive calls made will be on blue messages, and so the number of total recursions is bounded by the number of blue messages.

We will define three properties of executions of Runtime, and then we will show that these properties all hold with high probability. Together, these properties will imply that the the total number of recursions is bounded by the number of blue messages.

Notation 3.14. *For ease of notation, We will call the state of* Runtime *immediately after rcount is set to t to be the state of* Runtime *at time step t.*

Notation 3.15. *We say a message M is blue if there exists a path $(a_1, b_1), \ldots, (a_\ell, b_\ell)$ corresponding to M. That is,*

1. $a_1 = 0$,
2. $a_i = T[a_{i-1}, b_{i-1}]$ *for* $i > 1$,
3. $b_1 || \ldots || b_\ell = M$.

Property 1. *If at any time step t, $(a_1, b_1), (a_2, b_2) \in T$ with $h(a_1, b_1) = a_2$ and (a_2, b_2) is blue, then (a_1, b_1) is blue at time t.*

Property 2. *Let M be any message which is blue at time t. Let M' be any recursive query $(g^{\mathcal{O}})^*(M', \mathcal{Z}_2)$ made directly by $\mathcal{O}$ in the call to $(g^{\mathcal{O}})^*(M, \mathcal{Z}_2)$. Let $(a'_1, b'_1), \ldots, (a'_{\ell+k}, b'_{\ell+k})$ be the path corresponding to $M'||\mathcal{Z}_2$. In particular, this means that $Tpath(a'_{\ell+k}, b'_{\ell+k}) = M'||\mathcal{Z}_2$. Then for all i, either*

1. *(a'_i, b'_i) is blue at time t or*
2. *(a'_i, b'_i) is queried by $(g^h)^*(M, \mathcal{Z}_2)$.*

Property 3. *Let M be any message. Let M' be any recursive query $\mathcal{O}_1^*(M', \mathcal{Z}_1)$ made directly by $\mathcal{O}$ in the call to $(g^{\mathcal{O}_1})^*(M, \mathcal{Z}_2)$. Let $(a'_1, b'_1), \ldots, (a'_{\ell+k}, b'_{\ell+k})$ be the path corresponding to $M'||\mathcal{Z}_1$. In particular, this means that $Tpath(a'_{\ell+k}, b'_{\ell+k}) = M'||\mathcal{Z}_1$. Then there exists an $\ell < i \leq \ell + k$ such that (a'_i, b'_i) is not queried directly by $(g^h)^*(M, \mathcal{Z}_2)$.*

Informally, Property 1 says that blue queries can only follow blue queries. Property 2 says that every complete sequence of Merkle-Damgard query $\mathcal{O}^*(M) = \mathcal{O}(\mathcal{O}(\ldots, \mathcal{O}(\mathcal{O}(0, m_1), m_2), \ldots), m_\ell)$ must have all of its corresponding orange queries $\mathcal{O}(\ldots, m_i)$ within a single recursive call. Property 3 says that no single recursive call can query the last k rounds of $\mathcal{O}^*(M, \mathcal{Z}_1)$ by itself. The idea is that if all three properties hold, then for all recursive queries made, at least one round in the corresponding path is blue. Thus, the number of recursive queries is bounded by the number of direct queries.

Lemma 3.16. *Consider any randomness $r = (\mathcal{Z}_1, \mathcal{Z}_2, h)$. Let* Runtime$(r)$ *be* Runtime *instantiated with the given randomness. The probability over r that* Runtime(r) *satisfies Property 1 is greater than or equal to $1 - \frac{t_{max}^2}{2^n}$.*

Proof. Let $\epsilon \coloneqq \Pr[\mathsf{Runtime}(r) \text{ does not satisfy Property 1}]$. We construct an adversary $\mathcal{D}'$ predicting $h(x)$ without querying it, violating Theorem 2.4.

We first observe that if we run $\mathsf{Runtime}(r)$, but do not make any recursive queries, then up until the point where either terminates, the input output behavior of the oracles is exactly the same as if we were making recursive queries. Thus, we can recover the blue queries made by $\mathsf{Runtime}(r)$ at any point of the execution by simply running $\mathsf{Runtime}(r)$ without making recursive queries.

Thus, if there is some blue query which continues a non-blue query in $\mathsf{Runtime}(r)$, then we can recover the input to the blue query without knowing the output of the non-blue query. Formally, define $\mathcal{D}'$ as follows:

1. Guess two indices $q_1', q_2' \xleftarrow{\$} [t_{max}]$.
2. Guess $r \xleftarrow{\$} \{0,1\}^{2k\delta} \times \{h : \{0,1\}^{n+\delta} \to \{0,1\}^n\}$.
3. Run $\mathsf{Runtime}(r)$ normally until just before query q_1' on input (a_1', b_1').
4. Run $\mathsf{Runtime}(r)$ without making recursive queries until just before query q_1' on input (a_2', b_2').
5. Output $(x = (a_1', b_1'), y = a_2')$.

If $\mathsf{Runtime}(r)$ does not satisfy Property 1, then at some point in the execution there exist two queries $(a_1, b_1), (a_2, b_2) \in T$ such that (a_2, b_2) is blue and (a_2, b_2) is not and $h(a_1, b_1) = a_2$. Say that (a_1, b_1) is made during the q_1st query and (a_2, b_2) is made during the q_2nd direct query.

Then, if $q_1' = q_1$ and $q_2' = q_2$, it is clear that when running $\mathcal{D}'$, we have $(a_1', b_1') = (a_1, b_1)$ and $(a_2', b_2') = (a_2, b_2)$. Furthermore, since (a_1, b_1) is not blue by the time (a_2, b_2) is added to T, when $\mathcal{D}'$ runs $\mathsf{Runtime}(r)$ without making recursive queries, $\mathcal{D}'$ does not query $h(a_1, b_1)$. Since $\mathcal{D}'$ terminates before querying (a_2, b_2), $\mathcal{D}'$ does not query $h(a_1, b_1)$.

But since we have $h(x) = h(a_1, b_1) = a_2 = y$, $\mathcal{D}'$ is an attacker for Theorem 2.4 with success probability $\geq \epsilon \cdot \Pr[q_1' = q_1 \text{ and } q_2' = q_2]$. Thus,

$$\epsilon \cdot \frac{1}{t_{max}^2} \leq \frac{1}{2^n}.$$

□ □

Lemma 3.17. *The probability over $r = (\mathcal{Z}_1, \mathcal{Z}_2, h)$ that $\mathsf{Runtime}(r)$ satisfies Property 2 is greater than or equal to $1 - \frac{2 \cdot T_1 \cdot t_{max} \cdot T_{g^*}}{2^n}$.*

Proof. The idea behind this proof is roughly the same as for Theorem 3.16. In particular, we can generate all blue queries without making any recursive calls. This means that we can recover M and thus generate all queries made by $(g^h)^*(M, \mathcal{Z}_2)$ without making recursive calls.

In particular, let $\epsilon := \Pr[\mathsf{Runtime}(r) \text{does not satisfy Property 2}]$. We will construct $\mathcal{D}'$ violating Theorem 2.4.

Consider any r such that $\mathsf{Runtime}(r)$ does not satisfy Property 2. Let M and M' be two messages such that M is blue at time t, $(g^{\mathcal{O}})^*(M, \mathcal{Z}_2)$ triggers a recursion $(g^{\mathcal{O}})^*(M', \mathcal{Z}_2)$ and there is a query in the path of M' which is neither blue nor queried by $(g^h)^*(M, \mathcal{Z}_2)$.

Since we know that the last query in the path of M' is queried by $(g^h)^*(M, \mathcal{Z}_2)$, there must be some pair of queries $(a_1, b_1), (a_2, b_2) \in T$ such that (a_1, b_1) is not blue at time t nor queried directly by $(g^h)^*(M, \mathcal{Z}_2)$, and (a_2, b_2) is either blue or queried directly by $(g^h)^*(M, \mathcal{Z}_2)$. Formally, define $\mathcal{D}'$ as follows:

1. Guess indices $\widetilde{q_1} \xleftarrow{\$} [t_{max}]$.

2. Guess $r \xleftarrow{\$} \{0,1\}^{2k\delta} \times \{h : \{0,1\}^{n+\delta} \to \{0,1\}^n\}$.

3. Run $\mathsf{Runtime}(r)$ normally until just before query $\widetilde{q_1}$ with input $(\widetilde{a_1}, \widetilde{b_1})$.

4. Flip a coin $b \xleftarrow{\$} \{0,1\}$.

5. If $b = 0$,

   (a) Guess index $\widetilde{q_2} \xleftarrow{\$} [T_1]$.

   (b) Run $\mathsf{Runtime}(r)$ without making recursive queries until just before query $\widetilde{q_2}$ with input $(\widetilde{a_2}, \widetilde{b_2})$.

   (c) Output $(x = (\widetilde{a_1}, \widetilde{b_1}), y = \widetilde{a_2})$.

6. If $b = 1$,

   (a) Guess indices $\widetilde{q_M} \xleftarrow{\$} [T_1], q_2 \xleftarrow{\$} [T_{g^*}]$.

   (b) Run $\mathsf{Runtime}(r)$ without making recursive queries until just before query $\widetilde{q_M}$ with input $(\widetilde{a_M}, \widetilde{b_M})$. Set $\widetilde{X} = Tpath(\widetilde{a_M}, \widetilde{b_M})$. Set $\widetilde{M} = \widetilde{X}_{\leq \ell}$ the first ℓ blocks of $\widetilde{X}$.

   (c) Run $(g^h)^*(\widetilde{M}, \mathcal{Z}_2)$ until just before query $\widetilde{q_2}$ with input $(\widetilde{a_2}, \widetilde{b_2})$.

   (d) Output $(x = (\widetilde{a_1}, \widetilde{b_1}), y = \widetilde{a_2})$.

Let r be such that $\mathsf{Runtime}(r)$ does not satisfy Property 2. We first consider the case where (a_2, b_2) is blue. Let q_1 be the index of the query to (a_1, b_1) and let q_2 be the index of the query to (a_2, b_2) when we run $\mathcal{D}$ without making recursive calls. If $b = 0, q_1' = q_1$, and $q_2' = q_2$, then $\mathcal{D}'$ sets $x = (a_1, b_1)$ and $y = a_2$. Since (a_1, b_1) is not blue by the time (a_2, b_2) is queried, $\mathcal{D}'$ does not query (a_1, b_1).

We now consider the case where (a_2, b_2) is queried directly by $(g^h) * (M, \mathcal{Z}_2)$. Let q_M be the index of the query (a_M, b_M) which triggered the recursive call $(g^{\mathcal{O}})^*(M, \mathcal{Z}_2)$. Let q_2 be the index of the query of (a_2, b_2) in the call $(g^h)^*(M, \mathcal{Z}_2)$. If $b = 1, q_1' = q_1, q_M' = q_M$, and $q_2' = q_2$, we see $Tpath(\widetilde{a_M}, \widetilde{b_M}) = M || \mathcal{Z}_1$ and so $\widetilde{M} = M$. Also, $(\widetilde{a_1}, \widetilde{b_1}) = (a_1, b_1)$ and $(\widetilde{a_2}, \widetilde{b_2}) = (a_2, b_2)$ and so $x = (a_1, b_1)$ and $y = a_2$. Since (a_1, b_1) is not blue at the point (a_M, b_M) is queried nor queried by $(g^h)^*(M, \mathcal{Z}_2)$, $\mathcal{D}'$ does not query (a_1, b_1).

Thus, if r is such that $\mathsf{Runtime}(r)$ does not satisfy Property 2, $\mathcal{D}'$ is an attacker for Theorem 2.4 with success probability

$$\geq \min\left(\frac{\epsilon}{2T_1 \cdot t_{max}}, \frac{\epsilon}{2T_1 \cdot t_{max} \cdot T_{g^*}}\right) \geq \frac{\epsilon}{2T_1 \cdot t_{max} \cdot T_{g^*}}.$$

The lemma follows from Property 2. □ □

Lemma 3.18. *The probability over $r = (\mathcal{Z}_1, \mathcal{Z}_2, h)$ that $\mathsf{Runtime}(r)$ satisfies Property 3 is greater than or equal to $1 - \frac{T_{g^*}}{2^{(\ell-k)\delta}}$.*

Proof. The idea behind this proof is that if the property does not hold, we can use M to compress $\mathcal{Z}_1$. More formally, we will compress $(\mathcal{Z}_1, \mathcal{Z}_2, h)$ into fewer bits than is possible.

Let $\epsilon := \Pr[\mathsf{Runtime}(r) \text{ does not satisfy Property 3}]$. Note that if $\mathsf{Runtime}(r)$ does not satisfy Property 3, then that means there exists messages M, M' such that $(g^{\mathcal{O}})^*(M, \mathcal{Z}_2)$ queries the last k rounds of the path $(a'_1, b'_1), \ldots, (a'_{\ell+k}, b'_{\ell+k})$ corresponding to $M'||\mathcal{Z}_1$.

We define COM, $DECOM$ as follows.

$COM(\mathcal{Z}_1, \mathcal{Z}_2, h)$:

1. Run $\mathsf{Runtime}(\mathcal{Z}_1, \mathcal{Z}_2, h)$.
2. If ever Property 3 is violated, set M to be the violating message. Let q_k be the index of the query $(a_{\ell+k}, b_{\ell+k})$ made by $(g^h)^*(M, \mathcal{Z}_1)$.
3. If Property 3 holds, Output $\perp$.
4. Output $(q_k, M, \mathcal{Z}_2, h)$.

$DECOM(q_k, M, \mathcal{Z}_2, h)$:

1. If input is $\perp$, fail.
2. Run $(g^h)^*(M, \mathcal{Z}_2)$.
3. Let $(\widetilde{a_{\ell+k}}, \widetilde{b_{\ell+k}})$ be the q_kth query to $(g^h)^*(M, \mathcal{Z}_2)$.
4. For i from $\ell + k - 1$ to $\ell + 1$, recursively define $(\widetilde{a_i}, \widetilde{b_i})$ to be any query made by $(g^h)^*(M, \mathcal{Z}_2)$ such that $h(\widetilde{a_i}, \widetilde{b_i}) = \widetilde{a_{i+1}}$.
5. Output $(b_{\ell+1}|| \cdots ||b_{\ell+k}, \mathcal{Z}_2, h)$.

Note that if COM does not output $\perp$, then $(\widetilde{a_{\ell+k}}, \widetilde{b_{\ell+k}}) = (a'_{\ell+k}, b'_{\ell+k})$. Since $\mathsf{Runtime}(r)$ checks if there exists a collision in T, it must be the case that $(a'_{\ell+1}, b'_{\ell+1}), \ldots, (a'_{\ell+k}, b'_{\ell+k})$ is the unique path in T satisfying $h[a_i, b_i] = a_{i+1}$ terminating at $(a'_{\ell+k}, b'_{\ell+k})$. Thus, $(\widetilde{a_i}, \widetilde{b_i}) = (a'_i, b'_i)$. So, if COM does not output $\perp$, then $DECOM(COM(\mathcal{Z}_1, \mathcal{Z}_2, h) = (\mathcal{Z}_1, \mathcal{Z}_2, h)$. By pigeonhole principle, we have:

$$\Pr[DECOM(COM(r)) = r] \leq \frac{\#\text{ of values of } (q_k, M, \mathcal{Z}_2, h)}{\#\text{ of values of } (\mathcal{Z}_1, \mathcal{Z}_2, h)} \leq \frac{T_{g^*} \cdot 2^{k\delta}}{2^{\ell\delta}} \tag{3}$$

As $\Pr[DECOM(COM(r)) = r] \geq \epsilon$, we have that ϵ is also upper bounded by this, thus completing the proof.

□ □

Lemma 3.19. *If Properties 1 to 3 all hold and $t_{max} \geq T_1 \cdot (T_{g^*} + 1)$, $\mathsf{Runtime}(r)$ outputs 0.*

Proof. We will show that if Properties 1 to 3 all hold for $\mathsf{Runtime}(r)$, then every recursive query $(g^{\mathcal{O}})^*(M, \mathcal{Z}_2)$ must be made on a blue message M. If not, there must be some first recursive query $(g^{\mathcal{O}})^*(M', \mathcal{Z}_2)$ made on a non-blue message M'. Let $(a'_1, b'_1), \ldots, (a'_{\ell+k}, b'_{\ell+k})$ be the corresponding path.

If any of $(a'_{\ell+1}, b'_{\ell+1}), \ldots, (a'_{\ell+k}, b'_{\ell+k})$ is blue, then by Property 1 the whole preceding path is blue and so M' is blue, which is a contradiction.

If none of $(a'_{\ell+1}, b'_{\ell+1}), \ldots, (a'_{\ell+k}, b'_{\ell+k})$ are blue, $(a'_{\ell+k}, b'_{\ell+k})$ must be queried by some recursive query $(g^{\mathcal{O}})^*(M, \mathcal{Z}_2)$. Since M' is the first non-blue message we recurse on, M must be blue.

Thus, by Property 2, each (a'_i, b'_i) is either blue or queried directly by $(g^h)^*(M, \mathcal{Z}_2)$. Each of $(a'_{\ell+1}, b'_{\ell+1}), \ldots, (a'_{\ell+k}, b'_{\ell+k})$ is queried directly by $(g^h)^*(M, \mathcal{Z}_2)$. Since $b'_{\ell+1}||\cdots||b'_{\ell+k} = \mathcal{Z}_1$, this contradicts Property 3.

Every recursive query $(g^{\mathcal{O}})^*(M, \mathcal{Z}_2)$ must be made on a blue message M. So, the number of recursive calls is bounded by the number of blue messages. The number of blue messages is bounded by the number of calls to $\mathcal{O}$. But $rcount \leq (\#\text{ recursive queries}) \cdot (T_{g^*} + 1) \leq T_1 \cdot (T_{g^*} + 1)$ and the lemma follows. □ □

4 Conclusion and Open Problems

In this work we resolved the main open problem of [12], and showed security for the Merkle-Damgård-friendly (salted) random oracle 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 construction has many desirable features, such as output-preservation, compatibility with existing Merkle-Damgård-based hash functions, and minimal requirements on the input length of the corresponding compression function.

The main remaining inefficiency comes from the requirements on the salt lengths $\mathcal{Z}_1$ and $\mathcal{Z}_2$; each of them must be longer than the message M of the combiner. This still allows us to use this for any application where the length of the message M is a-priori fixed, such as Fiat-Shamir signatures [15], OAEP encryption [5], RSA encryption [5], hashed ElGamal, and others.

Combiners for Longer Inputs. Some applications would require either unbounded or variable inputs M. In these cases, it may be impractical to select a-priori unbounded salts. Note we could still use our combiner above, to first build a new *fixed-length* compression function f out of h_1 and h_2, and then manually built a variable-input function F which is indifferentiable from a variable-length random oracle. E.g., one of the Merkle-Damgård-based constructions from [9]. As shown by [12], here the composition theorem for the indifferentiability framework *still holds* (unlike the opposite case of first applying it to h_1, and then using such h_1 inside our combiner). As a result, we can obtain a secure ROC with a fixed-length salt. As a negative, this would require making the number of "Merkle-Damgård calls" to h_1^* and h_2^* grow with the message length M.

Open Problems. This leaves the following main open problem: Is there a secure random oracle combiner that (a) has $O(\lambda)$ length salts *and* (b) makes a constant number of calls to h_1^* and h_2^*, independent of message length? Conversely, show that such a combiner satisfying (a)+(b) cannot exist; in this case, it would be great to find the trade-off in salt length vs. number of "Merkle-Damgård-calls".

While our analysis of $\widetilde{C}$ critically uses the fact that $|\mathcal{Z}_i| > |M| + \lambda$, we do not have any attacks against this construction, once the $|\mathcal{Z}_i|$ is a constant number of blocks, independent of $|M|$. In particular, it is possible our construction $\widetilde{C}$ provides a solution to this main open problem! Thus, it would be very interesting to either show that $\widetilde{C}$ is insecure unless $|\mathcal{Z}_i| > |M|$, or find a supporting proof of security for much shorter salts (ideally, constant number of blocks).

References

[1] Elena Andreeva, Andrey Bogdanov, Yevgeniy Dodis, Bart Mennink, and John Steinberger. On the indifferentiability of key-alternating ciphers. In *Advances in Cryptology - CRYPTO 2013*, 2013.

[2] Elena Andreeva, Andrey Bogdanov, Yevgeniy Dodis, Bart Mennink, and John P. Steinberger. On the indifferentiability of key-alternating ciphers. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 531–550. Springer, Heidelberg, August 2013.

[3] 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.

[4] 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.

[5] Mihir Bellare and Phillip Rogaway. Optimal asymmetric encryption — how to encrypt with rsa. In *Advances in Cryptology - CRYPTO 1994*, 1994.

[6] 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.

[7] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. *Journal of Cryptology*, 17(4):297–319, September 2004.

[8] 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.

[9] 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.

[10] Jean-Sébastien Coron, Thomas Holenstein, Robin Künzler, Jacques Patarin, Yannick Seurin, and Stefano Tessaro. How to build an ideal cipher: The indifferentiability of the feistel construction. *J. Cryptol.*, 29(1):61–114, 2016.

[11] Yuanxi Dai, Yannick Seurin, John Steinberger, and Aishwarya Thiruvengadam. Indifferentiability of iterated even-mansour ciphers with non-idealized key-schedules: Five rounds are necessary and sufficient. In *Advances in Cryptology - CRYPTO 2017*, 2017.

[12] Yevgeniy Dodis, Niels Ferguson, Eli Goldin, Peter Hall, and Krzysztof Pietrzak. Random oracle combiners: Breaking the concatenation barrier for collision-resistance. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023, Part II*, volume

14082 of *Lecture Notes in Computer Science*, pages 514–546. Springer, Heidelberg, August 2023.

[13] Yevgeniy Dodis, Thomas Ristenpart, John Steinberger, and Stefano Tessaro. To hash or not to hash again? (in)differentiability results for h2 and hmac. In *Advances in Cryptology - CRYPTO 2012*, 2012.

[14] Yevgeniy Dodis, Martijn Stam, John Steinberger, and Tianren Liu. Indifferentiability of confusion-diffusion networks. In *Advances in Cryptology - EUROCRYPT 2016*, 2016.

[15] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology – CRYPTO'86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, Heidelberg, August 1987.

[16] 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.

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

[18] 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`.

[19] 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.

[20] 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.

[21] Ueli M. 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.

[22] Ueli M. Maurer and Stefano Tessaro. Domain extension of public random functions: Beyond the birthday barrier. In Alfred Menezes, editor, *Advances in Cryptology – CRYPTO 2007*, volume 4622 of *Lecture Notes in Computer Science*, pages 187–204. Springer, Heidelberg, August 2007.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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*, pages 487–506. Springer, Heidelberg, May 2011.

A Hard Hybrid

Lemma A.1. *We will write $G7(h, H; r)$ and $G8(h, H; r)$ to denote $G7$ and $G8$ respectively when the random functions chosen by the game are h and H and the randomness chosen by the adversary is r. We will write $T^7(t, h, H)$ to be the state of T in $G7(h, H)$ just before the tth (direct or recursive) call to $\mathcal{O}_1$. We will write $Tpath^7(a, b; t, h, H)$ to denote the result of calling $Tpath^7$ on (a, b) with hash functions h, H immediately after the tth call to $\mathcal{O}_1$. We similarly define $T^8(t, h, H)$ and $Tpath^8(a, b; t, h, H)$.*

Then with probability $\geq 1 - \frac{2t_{max}}{2^n}$ over h, H and the internal randomness r of $\mathcal{D}$, the following holds for all time steps t^7, t^8. If $FAIL$ is not set at time $t^7 - 1$ in $G7(h, H)$ and at time $t^8 - 1$ in $G8(h, H)$, and if the t^7th query in $G7$ and the t^8th query in $G8$ are to the same string (a, b), then

$$Tpath^7(a, b; t^7, h, H) = Tpath^8(a, b; t^8, h, H)$$

as long as either path is $\leq \ell + k$ blocks long.

Proof. Let ϵ be the probability that the condition in Theorem A.1 fails. If the condition fails, then the paths in $G7$ and $G8$ corresponding to (a, b) must diverge. That is, either

1. There exists $(a^7, b^7) \in T^7(t^7, h, H)$ and $(a^8, b^8) \in T^8(t^8, h, H)$ such that $(a^7, b^7) \notin T^8(t^8, h, H)$ and $T^7[a^7, b^7] = a^8$.
2. There exists $(a^8, b^8) \in T^8[t^8, h, H]$ and $(a^7, b^7) \in T^7(t^7, h, H)$ such that $(a^8, b^8) \notin T^7(t^7, h, H)$ and $T^8[a^8, b^8] = a^7$.

Let

$$(a_1^7, b_1^7), \ldots, (a_{r_7}^7, b_{r_7}^7)$$

and

$$(a_1^8, b_1^8), \ldots, (a_{r_8}^8, b_{r_8}^8)$$

be the paths in $G7$ and $G8$ traversed by $TPath^7(a, b; t^7, h, H)$ and $TPath^8(a, b; t^8, h, H)$ respectively.

By assumption, if $r_7 > \ell + k$, then $r_8 \leq \ell + k$. We observe that if this occurs, we must be in the second case. If not, then this means that the entire path in $G8$ must be included in $T^7(t^7 - 1, h, H)$. And since there are no collisions in $T^7(t^7 - 1, h, H)$ (as $FAIL$ is not set), we must have

$$(a_1^7, b_1^7), \ldots, (a_{r_7}^7, b_{r_7}^7) = (a_1^7, b_1^7), \ldots, (a_{r_7 - r_8}^7, a_{r_7 - r_8}^7), (a_1^8, b_1^8), \ldots, (a_{r_8}^8, b_{r_8}^8)$$

But we know that $a_1^8 = 0$. But this means $T^7(t^7 - 1, h, H)[a_{r_7 - r_8}^7, b_{r_7 - r_8}^7] = 0$, which is impossible since we know $FAIL$ is not set in $G7$ at time $t^7 - 1$.

Analogously, if the path in $G8$ is longer than $\ell + k$ blocks, then we must be in the first case. This means that in the first case we can assume that the length of the path leading up to (a^7, b^7) is $< \ell + k$ blocks (since $(a^7, b^7) \neq (a, b)$), and so $a^8 = T^7[a^7, b^7] = h(a^7, b^7)$. Similarly, in the second case, we can assume $a^7 = h(a^8, b^8)$. More formally, we have that one of the following two conditions hold

1. There exists $(a^7, b^7) \in T^7(t^7, h, H)$ and $(a^8, b^8) \in T^8(t^8, h, H)$ such that $(a^7, b^7) \notin T^8(t^8, h, H)$ and $h(a^7, b^7) = a^8$.
2. There exists $(a^8, b^8) \in T^8[t^8, h, H]$ and $(a^7, b^7) \in T^7(t^7, h, H)$ such that $(a^8, b^8) \notin T^7(t^7, h, H)$ and $h(a^8, b^8) = a^7$.

We will produce an attacker $\mathcal{D}$ for Theorem 2.4 which succeeds with probability $\geq \frac{\epsilon}{2t_{max}}$.

1. Sample h, H, r uniformly at random.
2. Guess $t^7, t^8 \xleftarrow{\$} [t_{max}]$
3. Run $G7(h, H; r)$ up until just before the t^7th query to $\mathcal{O}_1$, with input $(\widetilde{a}^7, \widetilde{b}^7)$.
4. Run $G8(h, H; r)$ up until just before the t^8th query to $\mathcal{O}_1$, with input $(\widetilde{a}^8, \widetilde{b}^8)$.
5. With probability 0.5, output $(x = (\widetilde{a}^7, \widetilde{b}^7), y = \widetilde{a}^8)$.
6. With probability 0.5, output $(x = (\widetilde{a}^8, \widetilde{b}^8), y = \widetilde{a}^7)$.

If the first case holds for h, H, then with probability $\geq \frac{1}{t_{max}}$, t^7 marks the first time (a^7, b^7) is queried to $\mathcal{O}_1$ in $G7(h, H)$. If so, then $\widetilde{a}^7 = a^7, \widetilde{b}^7 = b^7, \widetilde{a}^8 = a^8$, and $\mathcal{D}$ never queries $h(a^7, b^7)$. Since $h(a^7, b^7) = a^8$, $\mathcal{D}$ succeeds with probability $\geq \frac{1}{2t_{max}}$. Similarly, if the second case holds for h, H, $\mathcal{D}$ succeeds with probability $\geq \frac{1}{2t_{max}}$.

Since either the first or second case holds with probability ϵ, $\mathcal{D}$ succeeds with probability

$$\frac{\epsilon}{2t_{max}}$$

and so by Theorem 2.4,

$$\epsilon \leq \frac{2t_{max}}{2^n}$$

□ □

Corollary A.2 (Theorem 3.10 restated)**.**

$$|\Pr[G7 \to 1] - \Pr[G8 \to 1]| \leq \Pr[G7 \to \perp] + \frac{2t_{max}}{2^n}$$

and

$$\Pr[G8 \to \perp] \leq \Pr[G7 \to \perp] + \frac{2t_{max}}{2^n}$$

Proof. Let h, H, r be any collection of randomness such that $G7(h, H)$ does not return $\perp$ and the condition of Theorem A.1 holds.

Let t_i^7 be the time step of $G7(h, H)$ immediately after the ith non-recursive call to $\mathcal{O}_1$ or $\mathcal{O}_2$. Similarly define t_i^8. We will show by induction that for all i, $T^8[t_i^8, h, H] \subseteq T^7[t_i^7, h, H]$ and the response to query i is the same in both games.

In particular, this is trivial for $i = 0$. So assume that for all $i \leq k$, $T^8[t_i^8, h, H] \subseteq T^7[t_i^7, h, H]$ and the response to query i is the same in both $G7(h, H; r)$ and $G8(h, H; r)$. Then, since $\mathcal{D}$ in both games has the same randomness and has received the same responses to the same queries up until query k, the $k + 1$st query made by $\mathcal{D}$ in both games is also the same.

Furthermore, since $G7$ has not set $FAIL$ at this point in time, and since $T^8[t_k^8, h, H] \subseteq T^7[t_k^7, h, H]$, $G8$ has also not set $FAIL$ at this point in time.

If query $k + 1$ is a query to $\mathcal{O}_2$, then it returns $H(M)$ in both games and so gives the same response.

If query $k + 1$ is a query to $\mathcal{O}_1$, we will denote it by (a, b). By Theorem A.1, we have that $TPath^7(a, b) = TPath^8(a, b)$.

If $G8$ treats this as a stored query, then so will $G7$ by the inductive hypothesis. Thus, both games will not modify T and will simply return $T^7[a, b] = T^8[a, b]$.

If $G8$ treats this as a direct query, so will $G7$, and so both games return $h(a, b)$ and set $T[a, b] = h(a, b)$. Thus, $T^8[t_{k+1}^8, h, H] \subseteq T^7[t_{k+1}^7, h, H]$ since both are just the previous rounds value with $T[a, b] = h(a, b)$ added on.

If $G8$ treats this as a recursive query, so will $G7$. An inductive argument following the same lines as this case analysis shows that this recursive query will return the same value and any elements added in the recursive calls in $G8$ will either be added in $G7$ or were already there. Thus, we get that $T^8[t_{k+1}^8, h, H] \subseteq T^7[t_{k+1}^7, h, H]$ and both games return the same value to query $k + 1$.

Thus, since for all i, $T^8[t_i^8, h, H] \subseteq T^7[t_i^7, h, H]$, since $G7$ does not set flag $FAIL$, neither does $G8$. Furthermore, since they return the same query responses in both games, $\mathcal{D}$ returns the same value in both games. The corollary follows. □ □