Collision Resistant Hashing for Paranoids: Dealing with Multiple Collisions*

Ilan Komargodski Moni Naor Eylon Yogev

Abstract

A collision resistant hash (CRH) function is one that compresses its input, yet it is hard to find a collision, i.e. a $x_1 \neq x_2$ s.t. $h(x_1) = h(x_2)$. Collision resistant hash functions are one of the more useful cryptographic primitives both in theory and in practice and two prominent applications are in signature schemes and succinct zero-knowledge arguments.

In this work we consider a relaxation of the above requirement that we call Multi-CRH: a function where it is hard to find $x_1, x_2, \ldots, x_k$ which are all distinct, yet $h(x_1) = h(x_2) = \cdots = h(x_k)$. We show that for some of the major applications of CRH functions it is possible to replace them by the weaker notion of an Multi-CRH, albeit at the price of adding interaction: we show a statistically hiding commitment schemes with succinct interaction (committing to $\mathsf{poly}(n)$ bits requires exchanging $O(n)$ bits) that can be opened locally (without revealing the full string). This in turn can be used to provide succinct arguments for any statement. On the other hand we show black-box separation results from standard CRH and a hierarchy of such Multi-CRHs.

*Department of Computer Science and Applied Mathematics, Weizmann Institute of Science Israel, Rehovot 76100, Israel. Emails: {`ilan.komargodski,moni.naor,eylon.yogev`}`@weizmann.ac.il`. Supported in part by grants from the Israel Science Foundation. Ilan Komargodski is supported in part by a Levzion Fellowship. Moni Naor is the incumbent of the Judith Kleeman Professorial Chair.

Contents

1 Introduction

The task we must set for ourselves is not to feel secure, but to be able to tolerate insecurity.

Erich Fromm

In any function that compresses its input, say from $2n$ bits to n bits, there are many collisions, that is, pairs of distinct inputs whose image is the same. But what is the complexity of finding such a collision? Families of functions where this collision finding task is hard are known as collision resistant hash (CRH) functions.[1] A family of CRH functions have many appealing properties such as preservation under composition and concatenation. The presumed hardness of finding collisions in such functions is the basis for increased efficiency of many useful cryptographic schemes, in particular signature schemes and succinct (zero-knowledge) arguments, i.e., methods for demonstrating the correctness of a statement that are much shorter than the proof or even the statement itself (see Kilian [Kil92] and Barak and Goldreich [BG08]). The latter is achieved via a hash tree commitment[2] scheme whose opening is local i.e., opening a bit does not require revealing the full string (known as "Merkle tree"). These sort of results are essential for efficient delegation of computation where the goal is to offload significant computation to some server but also to verify the computation.

Such a task ("breaking a collision resistant hash function") is indeed hard based on a variety of assumptions such as the hardness of factoring integers, finding discrete logs in finite groups or learning with errors (LWE). There are popular functions (standards) with presumed hardness of collision finding such as SHA-2 and SHA-3 (adapted by NIST[3] in 2015). These functions can be evaluated very quickly, however, the their hardness is based on more ad hoc assumptions and indeed previous standards have been shown to be insecure (such as MD4, MD5, SHA-1). On the other hand there is no known construction of CRHs based solely on the existence of one-way functions or even permutations and, furthermore, they where shown to be separated in a black-box model (see Simon [Sim98]).

But a sufficiently compressing function also assures us that there are **multiple collisions**, i.e., k distinct values whose image under the function is equal. What about the problem of finding a k-collision? Assuming such hardness is a *weaker* computational assumption than hardness of finding a single pair of colliding inputs and the question is whether it yields a useful primitive.

In this paper we deal with *multiple collision resistant hash* (MCRH) functions and investigate their properties. We show that for a major application of CRH functions it is possible to replace them by an MCRH, albeit at the price of adding some rounds of interaction: **a commitment scheme with succinct communication that can be opened locally** (without revealing the full string) (see Theorem 2). This implies that it is possible to effectively verify the correctness of computation much more efficiently than repeating it. As an application we get constant-round[4]

[1]The function $h \in H$ can be sampled efficiently, it is easy to compute $h(x)$, however, it is hard to find $x_1 \neq x_2$ s.t. $h(x_1) = h(x_2)$.

[2]A commitment scheme is a protocol where a sender commits to a string x in the "commit" phase and that later can be revealed at the opening phase. The two properties are binding and hiding: in what sense is the sender bound to the string x (computationally or information theoretically) and in what sense is x hidden from the receiver before the opening - statistically or computationally.

[3]NIST is the National Institute of Standards and Technology.

[4]Whenever we say "constant-round" we mean that to commit to a string of length n^c for some constant $c \in N$

universal arguments [BG08] and constant-round zero-knowledge argument systems for NP [Bar01] based on MCRH functions. We also provide a constant-round statistically-hiding scheme and thus we can get constant-round statistical zero-knowledge arguments [BCC88].[5]

On the other hand we show various **black-box separation** results concerning MCRH. First, we separate them from one-way permutations. This follows from the lower bound of Haitner et al. [HHRS15] on the number of rounds needed to build a statistically-hiding commitment from one-way permutations. Furthermore, we show a black-box separation from standard CRH: there is no fully black-box construction of a k-CRH from a $(k+1)$-CRH for all k with polynomial security loss (see Theorem 4). These results yield an infinite hierarchy of natural cryptographic primitives, each two being separated by a fully black-box construction, between one-way function/permutations and collision-resistant hash function.[6]

One motivation for investigating MCRH functions is the work on finding collisions in the hash function SHA-1 (that has long been considered insecure [WYY05]) and recently an actual meaningful collision has been found [SBK+17]. In general finding a collision with arbitrary IV allows finding multiple collisions in an iteration of the function (say in a Merkle-Damgård lopsided tree), as shown by Joux [Jou04]. However, for the compression function of SHA-1 (or other such functions) there is no non-trivial algorithm for finding multi-collisions.[7] So the question is what can we do if all we assume about a given hash function is that multi-collision are hard to find, rather than plain collisions.

Our interest in MCRH functions originated in the work on the complexity of finding cliques or independent sets in graphs whose existence is assured by Ramsey Theory. It was shown that in bipartite graphs the problem is equivalent to MCRH: a hard distribution for finding these objects implies the existence of a k-CRH and vice-versa (with slightly different parameters) [KNY17].

Unlike CRHs that compose very nicely, and once we have a CRH that compresses by a single bit we can get any polynomial compression (i.e. from $\mathsf{poly}(n)$ to n bits), we do not know how to construct an MCRH on very large domains from a fixed MCRH.

Another relaxation of CRH is to consider Universal One-way Hash Functions (UOWHF) or second pre-image resistance, that first the target x is chosen (perhaps adversarially), then a function $h \in_R H$ is sampled and the goal is to find $x' \neq x$ s.t. $h(X) = h(x')$. Such families are good enough for signatures (at least existentially) and can be used for the hash-and-sign paradigm, if one chooses h per message (see Naor and Yung [NY89] and Mironov [Mir06]). It is known how to get such family of functions from one-way functions, but the construction is rather involved and inefficient (and there are some inherent reasons for that, see [GGKT05]). We show how to go from any interactive commitment where the communication is shorter than the string committed to (a succinct commitment) to a UOWHF (see Theorem 5). Together with our commitment schemes, this gives new constructions of UOWHFs on long inputs based on MCRHs with much shorter description than the ones known starting from a UOWHF on fixed input length.

A different way to relax the standard notion of collision resistance is what we call *multi-pair-collision-resistance*, where the challenge is to find *arbitrary* k distinct pairs of inputs that collide (possibly to different values). One may wonder what is the difference between these two notions and why we focus on the hardness of finding a k-wise collision rather than hardness of finding k

the number is a constant that depends on c.

[5]For constant values of k, we give a 4-round computationally-binding commitment scheme with succinct communication (see Theorem 3).

[6]This hierarchy translates to an infinite hierarchy of natural subclasses in TFNP. See Appendix B for details.

[7]Beyond the "birthday-like" algorithm that will take time $2^{n \cdot \frac{k-1}{k}}$ [Jou04], where 2^n is the size of the range.

distinct colliding pairs. The short answer is that the notion of k-pair-collision-resistance is actually *existentially equivalent* to the standard notion of collision resistance (see Appendix A).

Four worlds of hashing

Impagliazzo's five worlds [Imp95] are a way to characterize the strength of a cryptographic assumption. The worlds he defined are: *Algorithmica* (where $\mathsf{P} = \mathsf{NP}$), *Heuristica* (where NP is hard in the worst case but easy on average, i.e., one simply does not encounter hard problems in NP), *Pessiland* (where hard-on-the-average problems in NP exist, but one-way functions do not exist), *Minicrypt* (where one-way functions exist), and *Cryptomania* (where Oblivious Transfer exists). Nowadays, it is possible to add a sixth world, *Obfustopia*, where indistinguishability obfuscation for all programs exists.

In the spirit of Impagliazzo's five worlds of cryptographic assumptions, we define four worlds of hashing-related primitives:

1. *Nocrypt:* A world where there are no one-way functions.

2. *Unihash:* A world where one-way functions exist (and so do UOWHFs), but there are no MCRH functions.

3. *Minihash:* A world where MCRH exists but there is no CRH.

4. *Hashomania:* A world where CRH exists.

In the *Nocrypt* world there are no cryptographic commitments [IL89]. In the *Unihash* world there are universal one-way hash functions which suffice for some of the hashing applications such as hash-and-sign paradigm [NY89] and statistically-hiding commitments (albeit with a linear number of rounds) [NOVY98, HHK+09, HNO+09, HHRS15]. In the *Minihash* world, where k-MCRH functions exist, we give a protocol for short and statistically-hiding commitments with a *constant* number of rounds. Lastly, in the *Hashomania* world, there is a short commitment protocol that requires only *two* round (i.e., two messages).

Note that our separation results imply the these four worlds have black-box separations. Moreover, the separation in Section 7 actually implies that the world *Minihash* can be split further into sub-worlds parameterized by k, the number of collisions it is hard to find.

Concurrent work

In parallel to this work, MCRH functions were studied by two other groups that obtained various related results [BDRV17, BKP17].

Paper organization

In Section 2 we provide an overview of our main ideas and techniques. In Sections 3 and 4 we provide preliminary definitions and the definition of MCRH functions, respectively. In Section 5 we present the construction of our short commitment scheme with local opening. In Section 6 we construct our 4-round short commitment scheme. In Section 7 we prove the black-box separation between standard collision resistance and multi-collision resistance. Finally, in Section 8 we present the efficient construction of UOWHFs from multi-collision resistance.

In Appendices A and B we discuss the related notion of multi-pair collision resistance, and the implication of our infinite hierarchy of assumptions to TFNP, respectively.

2 Our Techniques

In this section we present some of our main ideas and techniques used in the construction of the commitment scheme and in the black-box separation result.

2.1 The main commitment scheme

A commitment scheme is a two stage interactive protocol between a sender and a receiver such that after the first stage the sender is bound to at most one value (this is called "binding"). In the second stage the sender can open his committed value to the sender. There are a few security properties one can ask from such a protocol: have statistical/computational binding, and have the committed value of the sender be statistically/computationally hidden *given* the commitment (this is called "hiding"). Our commitment scheme satisfies computational binding and statistical hiding. In this overview we will mostly focus on obtaining computational binding and briefly discuss how we obtain hiding towards the end.

There is a trivial commitment protocol that is (perfectly) binding: let the sender send its value to the receiver. In order to require some non-trivial properties, perhaps the most natural one is that the commitment is *shorter* than the committed string. There are additional useful properties one can require such as *local-opening* which allows the sender to open a small fraction of its input without sending the whole string (local opening is very important in applications of such commitment schemes, e.g., to proof systems and delegation protocols). Our protocol has a short commitment and it supports local-opening; we will focus here on the former and shortly discuss the latter towards the end.

Our goal now is to construct a commitment scheme which is computationally binding and the commitment is shorter than the value of the sender. If we had a standard collision resistant hash function mapping strings of length $2n$ to strings of length n, this task would be easy to achieve: the receiver will sample a hash function h and send it to the sender which will reply with $h(x^*)$, where $x^* \in \{0,1\}^{2n}$ is its value. The commitment thus consists of $(h, h(x^*))$ and its size is n bits.[8] It is easy to verify that for a sender to cheat during the opening phase it actually has to break the collision resistance of h (i.e., come up with a value $x \neq x^*$ such that $h(x) = h(x^*)$).

When h is only a k-MCRH for $k > 2$, the above protocol is clearly insecure: the sender can potentially find two inputs that collide to the same value and cheat when asked to open its commitment. The first observation we make is that even though the sender is not bound to a single value after sending $h(x^*)$, it is bound to a set of size $k-1$ of values. Otherwise, at least intuitively, he is able to find k inputs that map to the same output relative to h, which contradicts the security of the k-MCRH. Our first idea is to take advantage of this fact by adding an additional round of communication whose goal is to "eliminate" all but one possible value for the sender. Specifically, after the receiver got $h(x^*)$, it samples a random universal hash function[9] mapping strings of length

[8]For simplicity of presentation here, we ignore the cost of the description of h, so it will not significantly affect the size of the commitment.

[9]A universal hash function is a function of families $\mathcal{G} = \{g\colon \{0,1\}^n \to \{0,1\}^m\}$ such that for any $x, y \in \{0,1\}^n$ such that $x \neq y$ it holds that $\Pr_{g \leftarrow \mathcal{G}}[g(x) = g(y)] \leq 2^{-m}$.

$2n$ to strings of length m. and sends it to the sender. The sender then responds with $g(x)$. The commitment thus consists of $(h, h(x^*), g, g(x^*))$. To open the commitment the sender just sends x^* (just as before).

Indeed, one can show that this protocol is binding: Out of the $k-1$ possible values the sender knows that are consistent with $h(x^*)$, with probability roughly $k^2 \cdot 2^{-m}$ there will be no two that agree on $g(x^*)$. Conditioning on this happening, the sender cannot submit $x \neq x^*$ that is consistent with both $h(x^*)$ and $g(x^*)$. To formally show that the protocol is binding we need to show how to find a k-wise collision using a malicious sender. We simulate the protocol between the malicious sender and the receiver *multiple times* (roughly k times) with the same h but with freshly sampled g, by *partially rewinding* the malicious sender. We show that with good probability, every iteration will result with a new collision.

The protocol consists now of 4 rounds, but is the commitment short? Well, it depends on m and on the description size of g. The description size of a universal function is proportional to the input size ($2n$ in our case), which totally ruins the shortness of the protocol. One simple way to fix this is to sample g from an *almost* universal family and apply the above protocol. We obtain a 4-round protocol in which the commitment size is of the order roughly $n + m + \log(1/\delta)$, where δ is related to the error probability of the almost universal function. Choosing m and δ appropriately we obtain a protocol with short ($< 2n$) commitments.

Handling longer inputs. How would we commit on a longer string, say of $10n$ bits or even n^{10} bits? (Recall that all we have is a hash function mapping $2n$ bits into n bits.) The "text-book" solution is based on a standard collision resistant hash function, and is known as a Merkle tree. The input $x \in \{0,1\}^{2^d n}$ (for simplicity think of d as either a large constant or even $O(\log n)$) is partitioned into 2^d blocks each of size n. These blocks are partitioned into pairs and the hash function is applied to each pair resulting in 2^{d-1} blocks. Then, the remaining blocks are partitioned into pairs and the hash function is applied on each pair. This is repeated d times resulting in a binary tree of hash values of depth d. The value associated with the root of the tree is called the root-hash.

If h is a standard CRH sent by the receiver, then it is known that sending the root-hash by the sender is actually a commitment on the input x^* [Mer89a, Kil92]. Can we apply the same trick from before to make this a commitment protocol even when h is only a k-MCRH? That is, after sending the root-hash, let the sender sample a good-enough combinatorial hash function $g\colon \{0,1\}^{2^d n} \to \{0,1\}^m$ and send it to the sender that will reply with $g(x^*)$. Is this protocol binding? The answer is "no", even for large values of m. Indeed, observe that for every node in the Merkle tree, the sender can potentially provide $k-1$ valid inputs (that hash to the same value). Since the tree is of depth d, one can observe that by a mix-and-match method of different colliding values on different nodes of the tree the sender might be able to come up with up with $(k-1)^{2^d}$ valid inputs x whose corresponding root-hash is $h(x^*)$. Thus, to satisfy that no two have the same value under g, we have to choose $m \approx 2^d \cdot \log(k-1)$, in which case the almost uniform hash function has a pretty long description. Nevertheless, it is less than n so we might hope that some progress has been made. Is this protocol computationally-binding? Unclear. Using the proof technique from above (of partially rewinding and "collecting" collisions) would require running the malicious sender more than $(k-1)^{2^d}$ times until it has to come up with more than $k-1$ collisions for some value. This is, of course, way too expensive.

The bottom line of the above paragraph is that "mix-and-match" attacks are very powerful for a malicious sender in the context of tree hashing by allowing him to leverage the ability of finding few collisions into an ability to find exponentially many collisions. The reason why this happens is that we compose hash functions but apply the universal hash function on the whole input as a single string. Our next idea is to apply a "small" hash function $g\colon \{0,1\}^{2n} \to \{0,1\}^n$ per node in the Merkle tree. That is, after the sender sends the root-hash $h(x^*)$, the receiver samples g and sends it to the sender. The sender computes $g(\cdot,\cdot)$ for every pair of neighbors along the Merkle tree, concatenates them all and sends this long string back to the receiver. This protocol is more promising since, in some sense, we have a small consistency check per node in the tree which should rule out simple "mix-and-match" attacks. Indeed, this is our construction and the proof of security works by partially rewinding a malicious sender and "collecting" collisions until we get k collisions with respect to some internal node in the tree (we need to collect roughly $2^d k$ collisions overall so that such a node exists, by the pigeonhole principle). How efficient is the protocol? Details follow.

The protocol still consists of 4 rounds. A commitment consists of the hash function h, the root hash $h(x^*)$, a universal hash function $g\colon \{0,1\}^{2n} \to \{0,1\}^m$ and the value of g on *every* internal node of the tree. The overall size is thus of order $n + 2^d m$. Notice that $n + 2^d m \ll 2^d n$ whenever d is not too small, so we have made progress! We reduced the size of the commitment by a factor of m/n. The final step is to really get down to a commitment of size roughly n. To achieve this, we apply our protocol *recursively*: Instead of sending the hashes (w.r.t g) of all internal nodes, we run our *commit* protocol recursively on this string. Notice that this string is indeed shorter ($2^d m$ compared to $2^d n$) so the recursion does progress. The base of the recursion is when the string length is smaller than n, then the sender can simply send it to the receiver.

Choosing the parameters carefully, we get various trade-offs between the number of rounds, the commitment size, and the security of the resulting protocol. For example, setting $m = n^{0.99}$, results with a $O(1)$-round protocol in which the commitment size is $O(n)$ (here the big "O" hides constants that depend on $\log_n(|x^*|)$ which is constant for a polynomially long x^*), and whose security is worse than the security of the k-MCRH by an additive factor of $\exp(-n^{0.99})$.

Local opening. Due to the tree structure of our commitment protocol, it can be slightly modified to support local opening. Recall that the goal here is to allow the receiver to send an index i of a block to the sender, who can reply with the opening of the block, with communication proportional to n but not to the number of blocks 2^d. The idea here is, given an index i of a block, to open the hash values along the path corresponding to the i-block along with the tree neighbors of every node in the path. Then, i' is defined to be the index of the block in the shorter string (the string committed to in the next step of the recursion) which containing all the $g(\cdot,\cdot)$ values of the nodes on the path (we make sure that such a block exists). Then, we add the hash values of the path for block i' and continue in a recursive manner.

Statistical hiding. We show how to transform any short commitment scheme that is computationally binding (but perhaps not hiding) to a new scheme that is short, computationally binding and *statistically hiding*. Moreover, if the original scheme admits a local-opening, then the new scheme admits a local-opening as well. Our transformation is information theoretic, adds no additional assumptions and preserves (up to constant factors) the security, the number of rounds and communication complexity of the original scheme (up to a small constant factor). The transformation is partially based on ideas originating in the work of Naor and Yung [NY89, Section 5.2] and the follow-up works of Damgård, Pedersen, and Pfitzmann [DPP97, DPP98] giving constructions of statistical-hiding commitments from (standard) collision resistant hash function.

The idea of our transformation is to leverage the fact that the basic commitment protocol is short. Specifically, when committing to a long string x^*, the communication is very short. Thus, a large portion of x^* is not revealed to the receiver by the protocol so this part of x^* is statistically hidden. The task that remains is to make sure that all of x^* is hidden. Instead of committing to x^* directly, we commit on a random string r that is independent of x^* and slightly longer. Then, we extract from r the remaining randomness r' *given the communication of the protocol* using a strong extractor. Finally, we commit on the string $x^* \oplus r'$. It is not hard to show that if the original scheme was computationally binding, then the new one is as well. The fact that the scheme is statistically-hiding follows from the use of the strong extractor and the fact that the commitment is short.

One problem with the recipe above, is that the protocol (as describe) no longer admits a local-opening. This is because to open an index i, we need the i-th output bit of the extractor, but computing this bit might require reading a large portion of the input of r. Our solution is to break the input to sufficiently small parts such that each part is small enough to fit in a local-opening but is long enough to have enough entropy (given the communication of the protocol) so that we can apply the extractor on it.

2.2 Separating Multi-CRH from standard CRH

We show barriers of constructing a collision-resistant hash function from a 3-multi-collision-resistant hash functions. We rule out fully black-box constructions (see Definition 9). Our proof technique is inspired by the works of Asharov and Segev [AS16] and Haitner et al. [HHRS15], that are based in turn on ideas originating in the works of Simon [Sim98] Gennaro et al. [GGKT05]) and Wee [Wee07]. However, following their proof one encounters many obstacles so we explain how to overcome them.

The high-level overview of the proof is to show that there exist an oracle Γ which relative to Γ there exists a 3-MCRH, however there exist no standard CRH. Our oracle will contain a truly random function f that maps $2n$ bits to n bits. Relative to this oracle, it is clear that 3-MCRH exists, however, also standard CRH exist. We add an oracle $\mathsf{ColFinder}$ that will be used to break any CRH construction. The main difficulty of the proof is to show that this oracle cannot be used to break the 3-MCRH.

The oracle $\mathsf{ColFinder}$ is essentially the same as in Simon [Sim98]. It gets as an input a circuit C, possibly with f gates and it outputs two random elements w, w' such that $C(w) = C(w')$. It is easy to see that no family of hash functions can be collision resistant in the presence of such an oracle. A single call to $\mathsf{ColFinder}$ with the query $C(x) = f(x)$ will find a collision with high probability. The main question is can this oracle be used to find *multiple* collisions?

Originally, Simon showed that this oracle cannot be used to invert a one-way function (or even a permutation). Let $\mathcal{A}$ be an adversary that uses $\mathsf{ColFinder}$ to invert f on a random challenge $y = f(x)$. Clearly, if $\mathcal{A}$ make no calls to $\mathsf{ColFinder}$ then his chances in invert y are negligible. Assume, for simplicity, that $\mathcal{A}$ performs only a single query to $\mathsf{ColFinder}$. In order for $\mathcal{A}$ to gain some advantage, it must make an "interesting" query to $\mathsf{ColFinder}$. That is, a query which results in w, w' and the computation of either $C(w)$ or $C(w')$ makes a direct query to some $x \in f^{-1}(y)$. This event is called a hit. An important point is that for any circuit C the marginal distribution of w and of w' is uniform. Therefore, the probability of the event "hit" in the $\mathsf{ColFinder}$ query is at most twice that probability when evaluating $C(z)$ for a random z. Thus, we can construct a simulator that replaces $\mathcal{A}$ query to $\mathsf{ColFinder}$ with the evaluation of $C(z)$ and hits an inverse of

y with roughly the same probability as $\mathcal{A}$ (while making no queries to $\mathsf{ColFinder}$). The task of inverting y without $\mathsf{ColFinder}$ can be shown to be hard, ruling out the existence of such a simulator and in turn of such an adversary $\mathcal{A}$.

Our goal is to extend this proof and show that $\mathsf{ColFinder}$ cannot be used to find 3-wise collisions. The difficulty here is that the approach above simply does not work. Specifically, in our case the event "hit" corresponds to query C to $\mathsf{ColFinder}$ that results in w, w' and the computation of $C(w)$ and $C(w')$ *together* make direct queries three elements x_1, x_2, x_3 that collide under f (i.e., $f(x_1) = f(x_2) = f(x_3)$). It might be the case that these three elements are hit by $C(w)$ and $C(w')$ combined, but never by one of them alone. Thus, when simulating the $C(z)$ for a random z we will hit only part of the trio x_1, x_2, x_3 and might never hit all three.

Our main observation is that since $\mathcal{A}$ finds a 3-wise collision, but $\mathsf{ColFinder}$ finds a only a 2-wise collision (namely w, w'), by the pigeonhole principle, either w or w' will hit two of the three elements of the 3-wise collision. Again, since the marginals of w and w' each are uniform, we can construct a simulator that runs $\mathcal{A}$ and on the query to $\mathsf{ColFinder}$ samples a uniform z and compute $C(z)$ and will get a colliding x_1, x_2 without performing any queries to $\mathsf{ColFinder}$. Then, one can show that such a simulator cannot exist.

Several problems arise with this approach. First, notice that this does not extend naturally to an adversary $\mathcal{A}$ that makes more than one query. In such a case, the resulting simulator finds a 2-wise collision x_1, x_2 without the "hit" event occurring (i.e., finding a 3-wise collision) but while performing several $\mathsf{ColFinder}$ queries. Such a simulator (that finds a collision), of course, *trivially* exists, and we do not get the desired contradiction. Nevertheless, we show that the collision found by our simulator is somewhat special, and using $\mathsf{ColFinder}$ one can only find "non-special" collisions, ruling out the existence of $\mathcal{A}$ in this case. Second, the event "hit" itself might be spread out within several $\mathsf{ColFinder}$ queries, where, for example, one queries finds x_1 and then another query finds x_2, x_3. We tailor a simulator for each case, and show that for each case the resulting simulating cannot exist, completely ruling out the possibility of $\mathcal{A}$ to exist.

3 Preliminaries

Unless stated otherwise, the logarithms in this paper are base 2. For a distribution $\mathcal{D}$ we denote by $x \leftarrow \mathcal{D}$ an element chosen from $\mathcal{D}$ uniformly at random. For an integer $n \in \mathbb{N}$ we denote by $[n]$ the set $\{1, \ldots, n\}$. We denote by U_n the uniform distribution over n-bit strings. We denote by $\circ$ the string concatenation operation. A function $\mathsf{negl}\colon \mathbb{N} \to \mathbb{R}^+$ is *negligible* if for every constant $c > 0$, there exists an integer N_c such that $\mathsf{negl}(n) < n^{-c}$ for all $n > N_c$.

Definition 1 (Statistical Distance)**.** *The statistical distance between two random variables X, Y is defined by*

$$\Delta(X, Y) \triangleq \frac{1}{2} \cdot \sum_x |\Pr[X = x] - \Pr[Y = x]|$$

We say that X and Y are δ-close (resp. -far) if $\Delta(X, Y) \leq \delta$ (resp. $\Delta(X, Y) \geq \delta$).

3.1 Limited independence

Definition 2 (k-wise independence)**.** *Fix $n, m, k \in \mathbb{N}$. A function family $\mathcal{G} = \{g\colon \{0,1\}^n \to \{0,1\}^m\}$ is k-wise independent if for every distinct $x_1, \ldots, x_k \in \{0,1\}^n$ and every $y_1, \ldots, y_k \in$*

$\{0,1\}^m$ *it holds that*

$$\Pr_{g \leftarrow \mathcal{G}}[\forall i \in [k] \colon g(x_i) = y_i] = \frac{1}{2^{km}}.$$

It is known that for every $m \leq n$, there exists a k-wise independent family of functions, where each function is described by $k \cdot n$ bits. One well-known construction which is optimal in terms of size is by letting each $g \in \{0,1\}^{k \cdot n}$ describe a degree $k-1$ polynomial over $\mathsf{GF}[2^n]$. The description of the polynomial requires k field elements so $k \cdot n$ bits are enough. Evaluation of such a function is merely an evaluation of the polynomial.

In some applications (including some of ours) the input size n is very large and we prefer that the description size of the hash function to be much shorter. To circumvent this, it is sometimes enough to use *almost* k-wise independent function.

Definition 3 (Almost k-wise independence)**.** *Fix $n, m, k \in \mathbb{N}$ and $\delta \in \mathbb{R}$. A function family $\mathcal{G} = \{g \colon \{0,1\}^n \to \{0,1\}^m\}$ is (k,δ)-wise independent if for every distinct $x_1, \ldots, x_k \in \{0,1\}^n$ the distribution of $(g(x_1), \ldots, g(x_k))$ is δ-close to the distribution $(u_1, \ldots, u_k)$, where $g \leftarrow \mathcal{G}$ and each $u_i \leftarrow \{0,1\}^m$ are chosen uniformly at random.*

It is known that for every $m \leq n$, there exists a (k,δ)-wise independent function with each function $g \in \mathcal{G}$ being described by $O(mk + \log(n/\delta))$ bits [NN93, AGHP92].

3.2 Randomness extractors

We consider random variables supported on n-bit strings. A random variable X is said to have min-entropy $\mathsf{H}_\infty(X) = k$ if for every $x \in \mathsf{Supp}(X)$ it holds that $\Pr[X = x] \leq 2^{-k}$.

We say that a function $\mathsf{Ext} \colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ is a (k,ϵ)-*seeded extractor* if for every distribution X over $\{0,1\}^n$ with min-entropy k, it holds that

$$\Delta(\mathsf{Ext}(X, U_d), U_m) \leq \epsilon.$$

The extractor Ext is said to be *strong* if $\mathsf{Ext}'(x,s) = \mathsf{Ext}(x,s) \circ s$ is a (k,ϵ)-seeded extractor. That is, if

$$\Delta((\mathsf{Ext}(X, U_d) \circ U_d), (U_m \circ U_d)) \leq \epsilon.$$

The famous leftover hash lemma [ILL89, HILL99] says that a pairwise independent function family is a strong extractor.

Proposition 1. *Let $\mathcal{G} = \{g \colon \{0,1\}^n \to \{0,1\}^m\}$ be a pairwise independent family of hash functions where $m = k - 2\log(1/\epsilon)$. Then, $\mathsf{Ext}(x,h) = h(x)$ is a strong (k,ϵ)-seeded extractor.*

Note that the seed length in this extractor equals the number of bits required to sample $g \leftarrow \mathcal{G}$ which is $2n$ bits.

We will also need the following standard proposition that says that conditioning does not reduce entropy by more than the information given by the conditioning.

Proposition 2. *Let X and Y be random variables. Then, if Y is supported on strings of length k, then $\mathsf{H}_\infty(X \mid Y) \geq \mathsf{H}_\infty(X) - k$.*

3.3 List-recoverable codes

The classical notion of error correcting codes ensures that for a code $C \subseteq \mathbb{F}^n$, where $\mathbb{F}$ is a finite field, given a somewhat corrupted version of $c \in C$, it is possible to recover c. The model of allowed corruptions is that some fraction of the symbols in the codeword might be adversarially changed. List recoverable codes were introduced to handle a different model of corruptions: they allow an adversary to submit, for every coordinate $i \in [n]$ a small list $S_i \subseteq \mathbb{F}$ of possible symbols. In this model, it is impossible to completely recover a codeword given the lists, but these codes guarantee that there is only a small list of codewords that are consistent with all the lists.

More precisely, a mapping $C : \mathbb{F}^k \to \mathbb{F}^n$ from length k messages to length n codewords, is called (α, ℓ, L)-*list-recoverable* if there is a procedure that is given a sequence of lists $S_1, \ldots, S_n \subseteq \mathbb{F}$ each of size ℓ, and is able to output all messages $x \in \mathbb{F}^k$ such that $C(x)_i \notin S_i$ for at most an α fraction of the coordinates $i \in [n]$. The code guarantees that there are at most L such messages.

Definition 4 (List-recoverable codes)**.** *Let $\alpha \in [0, 1]$. We say that a tuple $x \in (\{0, 1\}^k)^n$ is* α-consistent *with sets $S_1, \ldots, S_n \subseteq \{0, 1\}^k$, if $|\{i : x_i \in S_i|\} \geq \alpha n$.*

A function $C : \{0, 1\}^v \to (\{0, 1\}^k)^n$ is (α, ℓ, L)-list recoverable, if for every set $S_1, \ldots, S_n \subseteq \{0, 1\}^k$ each of size at most ℓ, there are at most L strings $x \in \{0, 1\}^v$ such that $C(x)$ is α-consistent with $S_1, \ldots, S_n$. For $\alpha = 1$, we omit α in the above notation and call C (ℓ, L)-list recoverable. The strings in the image of C are referred to as codewords.

These code were initially studied in the context of *list-decoding* (and indeed the latter is just a special case of the former with $\ell = 1$) by [GS99, GI02, GI03, GI04]. More recently, they were proven useful in other areas such as compressed sensing [NPR12], non-adaptive domain extension for hashing [HIOS15], and more (see [HW15] and references therein).

A natural relaxation of the above codes is to require that $S_1 = \ldots = S_n$. This variant is called *weakly* list-recoverable codes. A list-recoverable code is immediately weakly list-recoverable and the converse also holds albeit with a minor loss in parameters: An (ℓ, L)-weakly list-recoverable code is an (ℓ, nL)-list-recoverable code. Looking ahead, this loss will not make much of a difference for us since our L will be polynomial in n.

For our purposes, we will need a list-recoverable code with $\alpha = 1$. It is well-known (see e.g., [HIOS15]) that the notion of weakly list-recoverable codes is equivalent to unbalanced expanders with a certain expansion property. The left set of vertices in the graph is $\{0, 1\}^v$, the right set of vertices is $\{0, 1\}^k$ and the left degree is n. This graph naturally induces a mapping $C : \{0, 1\}^v \to (\{0, 1\}^k)^n$ which on input $x \in \{0, 1\}^v$ (left vertex) outputs n neighbors (right vertices). The mapping C is (ℓ, L)-list-recoverable iff for every set $S \subseteq \{0, 1\}^k$ of size larger than L of nodes on the right, the set of left neighbors of S is of size larger than ℓ.

The following instantiation of locally-recoverable codes based on the explicit construction of unbalanced expanders of [GUV09] is taken (with minor modifications) from [HIOS15].

Theorem 1 ([GUV09, HIOS15])**.** *For every $\alpha \geq 1/2$, and $k < v$, there exists a* $\mathsf{poly}(n)$*-time computable function $C : \{0, 1\}^v \to \left(\{0, 1\}^k\right)^n$ for $n = O(v \cdot k)^2$ which defines an (α, ℓ, L)-list recoverable code for every $L \leq 2^{k/2}$ and $\ell = \Omega(L)$. The list-recovery algorithm runs in time* $\mathsf{poly}(v, \ell)$*.*

3.4 Cryptographic primitives

A function f, with input length $m_1(n)$ and outputs length $m_2(n)$, specifies for every $n \in \mathbb{N}$ a function $f_n : \{0, 1\}^{m_1(n)} \to \{0, 1\}^{m_2(n)}$. We only consider functions with polynomial input lengths (in n) and occasionally abuse notation and write $f(x)$ rather than $f_n(x)$ for simplicity. The function f is computable in polynomial time (efficiently computable) if there exists an adversary that for any $x \in \{0,1\}^{m_1(n)}$ outputs $f_n(x)$ and runs in time polynomial in n.

A function family ensemble is an infinite set of function families, whose elements (families) are indexed by the set of integers. Let $\mathcal{F} = \{\mathcal{F}_n : \mathcal{D}_n \to \mathcal{R}_n\}_{n\in\mathbb{N}}$ stand for an ensemble of function families, where each $f \in \mathcal{F}_n$ has domain $\mathcal{D}_n$ and range $\mathcal{R}_n$. An efficient function family ensemble is one that has an efficient sampling and evaluation algorithms.

Definition 5 (Efficient function family ensemble)**.** *A function family ensemble $\mathcal{F} = \{\mathcal{F}_n : \mathcal{D}_n \to \mathcal{R}_n\}_{n\in\mathbb{N}}$ is efficient if:*

- *$\mathcal{F}$ is samplable in polynomial time: there exists a probabilistic polynomial-time machine that given 1^n, outputs (the description of) a uniform element in $\mathcal{F}_n$.*
- *There exists a deterministic algorithm that given $x \in \mathcal{D}_n$ and (a description of) $f \in \mathcal{F}_n$, runs in time $\mathsf{poly}(n, |x|)$ and outputs $f(x)$.*

Universal one-wayness. A one-way function is an efficiently computable function which is hard to invert on a random output for any probabilistic polynomial-time machine. A universal one-way hash function (UOWHF) is a family of compressing functions $\mathcal{H}$ for which any PPT adversary has a negligible chance of winning in the following game: the adversary submits an x and gets back a uniformly chosen $h \leftarrow \mathcal{H}$. The adversary wins if it finds an $x' \neq x$ such that $h(x) = h(x')$. UOWHF were introduced by Naor and Yung [NY89] and were shown to imply secure digital signature schemes. Rompel [Rom90] (see also [KK05]) showed how to construct UOWHF based on the minimal assumption that one-way functions exist.

Definition 6 (Universal one-way hash functions (UOWHF))**.** *An efficient function family ensemble $\mathcal{F} = \{\mathcal{F}_n : \{0,1\}^{m_1(n)} \to \{0,1\}^{m_2(n)}\}_{n\in\mathbb{N}}$ is a universal one-way hash function family if the probability of every probabilistic polynomial-time adversary $\mathcal{A}$ to win in the following game is negligible in n:*

1. *$\mathcal{A}$, given 1^n, submits $x \in \{0,1\}^{m_1(n)}$.*
2. *Challenger responds with a uniformly random $f \leftarrow \mathcal{F}_n$.*
3. *$\mathcal{A}$ (given f) outputs $x' \in \{0,1\}^{m_1(n)}$.*
4. *$\mathcal{A}$ wins iff $x \neq x'$ and $f(x) = f(x')$.*

3.5 Commitment schemes

A commitment scheme is a two-stage interactive protocol between a sender $\mathcal{S}$ and a receiver $\mathcal{R}$. The goal of such a scheme is that after the first stage of the protocol, called the commit protocol, the sender is bound to at most one value. In the second stage, called the opening protocol, the sender opens its committed value to the receiver. We also require that the opening protocol allows to open only a single bit of the committed string. More precisely, a commitment scheme for a domain of strings $\{0,1\}^\ell$ is defined via a pair of probabilistic polynomial-time algorithms $(\mathcal{S}, \mathcal{R}, \mathcal{V})$ such that:

- The commit protocol: $\mathcal{S}$ receives as input the security parameter 1^n and a string $s \in \{0,1\}^\ell$. $\mathcal{R}$ receives as input the security parameter 1^n. At the end of this stage, $\mathcal{S}$ outputs $\mathsf{decom}_1 \ldots, \mathsf{decom}_\ell$ (the local decommitments) and $\mathcal{R}$ outputs com (the commitment).

- The local-opening procedure: $\mathcal{V}$ receives as input the security parameter 1^n, a commitment com, an index $i \in [\ell]$, a local-decommitment decom_i, and outputs either a bit b or $\perp$.

A commitment scheme is *public coin* if all messages sent by the receiver are independent random coins.

Denote by $(\mathsf{decom}_1, \ldots, \mathsf{decom}_\ell, \mathsf{com}) \leftarrow \langle \mathcal{S}(1^n, s), \mathcal{R} \rangle$ the experiment in which $\mathcal{S}$ and $\mathcal{R}$ interact with the given inputs and uniformly random coins, and eventually $\mathcal{S}$ outputs a list of ℓ decommitment strings and $\mathcal{R}$ outputs a commitment. The completeness of the protocol says that for all $n \in \mathbb{N}$, every string $s \in \{0,1\}^\ell$, every tuple $(\mathsf{decom}_1, \ldots, \mathsf{decom}_\ell, \mathsf{com})$ in the support of $\langle \mathcal{S}(1^n, s), \mathcal{R} \rangle$, and every $i \in [\ell]$, it holds that $\mathcal{V}(i, \mathsf{decom}_i, \mathsf{com}) = s_i$.

Below we define two security properties one can require from a commitment scheme. The properties we list are *statistical-hiding* and *computational-binding.* These roughly say that after the commit stage, the sender is *bound* to a specific value but the receiver cannot know this value. We focus on the notions of statistical-hiding and computational-binding.

Definition 7 (ϵ-binding)**.** *A commitment scheme $(\mathcal{S}, \mathcal{R}, \mathcal{V})$ is $(t(n), \epsilon(n))$-binding if for every probabilistic adversary $\mathcal{S}^*$ that runs in time at most $t(n)$, it holds that*

$$\Pr\left[\begin{array}{c}(i, \mathsf{decom}_i, \mathsf{decom}'_i, \mathsf{com}) \leftarrow \langle \mathcal{S}^*(1^n), \mathcal{R} \rangle \text{ and} \\ \perp \neq \mathcal{V}(i, \mathsf{decom}_i, \mathsf{com}) \neq \mathcal{V}(i, \mathsf{decom}'_i, \mathsf{com}) \neq \perp\end{array}\right] \leq \epsilon(n)$$

for all sufficiently large n, where the probability is taken over the random coins of both $\mathcal{S}^*$ and $\mathcal{R}$.

Given a commitment scheme $(\mathcal{S}, \mathcal{R}, \mathcal{V})$ and an adversary $\mathcal{R}^*$, we denote by $\mathsf{view}_{\langle \mathcal{S}(s), \mathcal{R}^* \rangle}(n)$ the distribution on the view of $\mathcal{R}^*$ when interacting with $\mathcal{S}(1^n, s)$. The view consists of $\mathcal{R}^*$'s random coins and the sequence of messages it received from $\mathcal{S}$. The distribution is take over the random coins of both $\mathcal{S}$ and $\mathcal{R}$. Without loss of generality, whenever $\mathcal{R}^*$ has no computational restrictions, we can assume it is deterministic.

Definition 8 (ρ-hiding)**.** *A commitment scheme $(\mathcal{S}, \mathcal{R}, \mathcal{V})$ is $\rho(n)$-hiding if for every (deterministic) adversary $\mathcal{R}^*$ and every distinct $s_0, s_1 \in \{0,1\}^\ell$, it holds that*

$$\Delta\left(\{\mathsf{view}_{\langle \mathcal{S}(s_0), \mathcal{R}^* \rangle}(n)\}, \{\mathsf{view}_{\langle \mathcal{S}(s_1), \mathcal{R}^* \rangle}(n)\}\right) \leq \rho(n)$$

for all sufficiently large $n \in \mathbb{N}$.

Complexity measures. The parameters of a commitment scheme that are of interest are (1) the number of rounds the commit protocol requires, (2) the size of a commitment, and (3) the size of a local opening.

The *size* of a commitment is the size (in bits) of the output of $\mathcal{S}$ denoted above by com. A *short commitment* is such that the size of com is much smaller than ℓ. Preferably, the size of a short commitment depends solely on n, but poly-logarithmic dependence on ℓ is also okay. The size of a local opening is the maximum size of decom_i (in bits). A protocol is said to support local opening if this size depends only on n and is otherwise independent of ℓ.

3.6 Fully black-box constructions

We give a definition of a fully black-box reduction from an MCRH to standard CRH. For this, we generalize the definition of an MCRH to the setting of oracle-aided computation: The generation and evaluation algorithms of an MCRH are given access to an oracle Γ relative to which they can generate a description of a hash function and evaluate an index at a point. The adversary is also given oracle access to Γ in the security game and has to find multiple collisions relative to it.

We focus here on k-MCRH functions with $k = 3$. The following definition of a "black-box construction" is directly inspired by those of [AS16, HHRS15].

Definition 9. *A fully black-box construction of a collision-resistant function family $\mathcal{H}'$ from a 3-MCRH function family $\mathcal{H}$ mapping $2n$ bits to n bits consists of a pair of probabilistic polynomial-time algorithms $(\mathcal{H}.\mathsf{G}, \mathcal{H}.\mathsf{E})$ and an oracle-aided polynomial-time algorithm M such that:*

- **Completeness:** *For any $n \in \mathbb{N}$, for any 3-MCRH function family $\mathcal{H}$ and any function h produced by $h \leftarrow \mathcal{H}'^{\mathcal{H}}(1^n)$, it holds that $h^{\mathcal{H}} \colon \{0,1\}^{2n} \to \{0,1\}^n$.*
- **Black-box proof of security:** *For any collision resistant hash $\mathcal{H}'$, any probabilistic polynomial-time oracle-aided algorithm $\mathcal{A}$, every polynomial $p(\cdot)$, if*

$$\Pr\left[\begin{matrix} x_1 \neq x_2 \\ h^{\mathcal{H}}(x_1) = h^{\mathcal{H}}(x_2) \end{matrix} \middle| \begin{matrix} h \leftarrow h^{\mathcal{H}}(1^n) \\ (x_1, x_2) \leftarrow \mathcal{A}^{\mathcal{H}}(1^n, h) \end{matrix}\right] \geq \frac{1}{p(n)}$$

for infinitely many values of n, then there exists a polynomial $p'(\cdot)$ such that

$$\Pr\left[\begin{matrix} x_1, x_2, x_3 \text{ are distinct and} \\ h(x_1) = h(x_2) = h(x_3) \end{matrix} \middle| \begin{matrix} h \leftarrow \mathcal{H}(1^n) \\ (x_1, x_2, x_3) \leftarrow M^{\mathcal{A},\mathcal{H}}(1^n, h) \end{matrix}\right] \geq \frac{1}{p'(n)}$$

for infinitely many values of n.

4 Multi-Collision-Resistant Function Families

We define the notion of a multi-collision-resistant hash function. This definition is a relaxation of standard collision-resistant hash function in which it is hard to find *multiple* collisions on the same value.

Definition 10 (Multi-Collision-Resistant Hashing)**.** *Let $k = k(n)$ be a polynomial function. An efficient function family ensemble $\mathcal{H} = \{\mathcal{H}_n \colon \{0,1\}^{2n} \to \{0,1\}^n\}_{n \in \mathbb{N}}$ is a (t, ϵ)-secure k-*multi-collision-resistant hash *(MCRH) function family if for any probabilistic algorithm $\mathcal{A}$ that runs in time at most $t(n)$, for large enough $n \in \mathbb{N}$:*

$$\Pr\left[\begin{matrix} x_1, \ldots, x_k \text{ are distinct and} \\ h(x_1) = \cdots = h(x_k) \end{matrix} \middle| \begin{matrix} h \leftarrow \mathcal{H}_n \\ (x_1, \ldots, x_k) \leftarrow \mathcal{A}(h) \end{matrix}\right] \leq \epsilon(n).$$

*We call such $x_1, \ldots, x_k$ that map to the same value under h a k-*wise collision*. Lastly, we say that $\mathcal{H}$ is a secure k-MCRH if it is $(p, 1/p)$-secure for every polynomial $p(\cdot)$.*

The output length. In the definition above we assume that the hash function compresses its input from $2n$ bits into n. A k-MCRH exists unconditionally whenever it is allowed to output more than $2n - \log k$ bits (there simply will be no k-wise collision). The interesting range of parameters is when the output is at most $2n - \log k$ bits since then a k-wise collision must exist. Since k is at most a polynomial in n, we just set the output length to be $n \ll 2n - \log k$.

For standard collision resistant hash function (with $k = 2$), it is known that composition allows to translate hash functions that compress by one bit into hash functions that compress by any polynomial factor (from n bits into n^δ bits for any constant $\delta > 0$). Obtaining a similar result for MCRH functions is an interesting open problem.

5 Tree Commitments from Multi-CRH

We show how to build a commitment scheme which is computationally-binding, statistically-hiding, round-efficient, has short commitments, and supports local opening. We refer to Section 3.5 for the definition of a commitment scheme, for computational-binding, statistical-hiding, and the efficiency measures of commitments we consider below.

Theorem 2. *Assume that there exists a (t, ϵ)-secure k-MCRH $\mathcal{H}$ for a polynomial $k = k(n)$ in which every function can be described using $\ell = \ell(n)$ bits. For any parameters $d = d(n)$ and $1 < z \leq n/2d$, there is commitment protocol for strings of length $2^d \cdot n$ with the following properties:*

1. *(t', ϵ')-computationally-binding for $\epsilon' = O\left(2^{\frac{d}{\log(n/(zd))}} \cdot \left(\frac{k^2}{2^{z-d}} + \epsilon\right)\right)$ and $t' = O\left(\frac{\epsilon'^2 \cdot t}{nk2^d \cdot p(n)}\right)$, where $p(\cdot)$ is some fixed polynomial function.*
2. *2^{-n}-statistically-hiding.*
3. *takes $O\left(\frac{d}{\log(n/(zd))}\right)$ rounds.*
4. *the commitment has length $O(d\ell + dn)$.*
5. *supports local opening of size $O\left(d^2 n\right)$.*

There are various ways to instantiate z compared to n and d, offering various trade-offs between security and efficiency. We focus here on the case in which we wish to commit on a polynomially-long string, that is, $d = c \log n$ for a constant $c \in \mathbb{N}$. In the following the big "O" notation hides constants that depend on c. Setting $z = n^{1-\delta}$ for a small constant $\delta > 0$, the parameters of our commitment scheme are:

1. $\epsilon' = O\left(\frac{k^2}{2^{n^{1-\delta}}} + \epsilon\right)$ and $t' = O\left(\frac{\epsilon'^2 \cdot t}{n^c k p(n)}\right)$.
2. 2^{-n}-statistically-hiding.
3. takes $O(1)$ rounds.
4. the commitment has length $O(\ell + n \log n)$.
5. supports local opening of size $O(n \log^2 n)$.

This setting is very efficient in terms of rounds (it is constant) but suffers in security loss (the resulting scheme is at most $2^{-n^{1-\delta}}$ secure). Another setting for z is $z = n/(2c \log n)$ for a small constant $\delta > 0$. The resulting commitment scheme has the following properties:

1. $\epsilon' = O\left(n^{c-1} \cdot \left(\frac{k^2}{2^{n/\log n}} + \epsilon\right)\right)$ and $t' = O\left(\frac{\epsilon'^2 \cdot t}{n^c k p(n)}\right)$.
2. 2^{-n}-statistically-hiding.
3. takes $O(\log n)$ rounds.
4. the commitment has length $O(\ell \log n + n \log n)$.
5. supports local opening of size $O(n \log^2 n)$.

This setting has a logarithmic number of rounds, but the security loss is much smaller than before (only of order $2^{-n/\log n}$).

Roadmap. Our protocol is constructed in two main steps. In the first step (given in Section 5.1) we construct a protocol with the above properties (i.e., Theorem 2) except that it is *not* statistically hiding. That is, the protocol is computationally binding, takes few rounds, has short commitment, and supports local opening. In the second step (given in Section 5.2), we show how to *generically* bootstrap any commitment scheme with the above properties, into one that is also statistically-hiding. This reduction is both efficient and security preserving with respect to all parameters.

5.1 A computationally-binding scheme

The main ingredients in our first protocol are an MCRH (Definition 10) and a limited-independent family (Definition 2):

- A (t, ϵ)-secure k-MCRH for a polynomial $k = k(n)$:

$$\mathcal{H} = \{h \colon \{0,1\}^{2n} \to \{0,1\}^n\}.$$

  We assume that every $h \in \mathcal{H}$ can be described using $\ell = \ell(n)$ bits.

- A family of pairwise-independent functions mapping strings of length $2n$ to strings of length z:

$$\mathcal{G} = \{g \colon \{0,1\}^{2n} \to \{0,1\}^z\}.$$

  Recall that every $g \in \mathcal{G}$ can be described using $4n$ bits.

Description of the protocol. Our protocol relies on the notion of a Merkle hash tree. This is a method to hash a long string into a short one using a hash function with fixed input length. Let $x \in \{0,1\}^\ell$ be a string. A Merkle hash tree is a binary tree T, associated with a string $x \in \{0,1\}^\ell$ and a hash function $h \colon \{0,1\}^{2n} \to \{0,1\}^n$. Let $x = x_1, \ldots, x_{2^d}$ be the decomposition of x into 2^d blocks, each of length n. Every node v in the tree has a neighbor denoted by $N(v)$ (we assume that the neighbor of the root is $\perp$). Every node v in the tree is labeled with a string $\pi_v \in \{0,1\}^n$. The tree has 2^d leaves $v_1, \ldots, v_{2^d}$ and the label of v_i are set to $\pi_{v_i} = x_i$. The labels of the rest of the nodes are computed iteratively from the leaves to the root. Given a node v whose both children u_1, u_2 are labeled with π_{u_1}, π_{v_2}, we set the label of v to be $\pi_v = h(\pi_{u_1}, \pi_{v_2})$. The node root has label y and we call it the root-hash.

Given a Merkle hash tree for a string $x = x_1 \dots x_{2^d} \in \{0,1\}^{2^d \cdot n}$, let path_i be a set of nodes in the tree including the nodes on the path from x_i to the root of the tree and all their neighbors along this path. We further let $P_i = \{\pi_v \mid v \in \mathsf{path}_i\}$ be the set of all the labels of the nodes in the set path_i (the labels of the nodes on the path from x_i to the root of the tree and the labels of their neighbors). Each set path_i contains $2d$ nodes and thus the description size of each P_i is $2dn$ bits.

The commitment protocol $(\mathcal{S}, \mathcal{R}, \mathcal{V})$ specifies how to commit to a string of length $2^d \cdot n$. Our protocol uses a Merkle hash tree with a function h supplied by the receiver and a root hash replied by the sender. In the next round, the receiver chooses a limited-independence function g with z bits of output (z is a tunable parameter) and sends it to the sender. The sender then computes g on the *hash values* of every pair of neighboring nodes in the Merkle hash tree (i.e., $g(\pi_v \circ \pi_{N(v)})$). Then it concatenates all of these values into one long string s'. Then, the protocol continues in a recursive manner on this string s'. The length of s' is roughly $z2^d$ bits (there are 2^d internal nodes in the tree and each requires z bits) which is still too long to send as is, however is smaller than the original string s. This allows us to apply the same ideas recursively with the base case, committing on a string of length n, being the trivial protocol of sending the string as is. Choosing parameters carefully, we balance between the efficiency and security of our resulting commitment.

The commitment protocol for strings of length $2^d \cdot n$ for $d \geq 1$ is described in Figure 1.

Rounds and communication analysis. Denote by $\mathsf{Size}(2^d n)$, $\mathsf{Rounds}(2^d n)$, and $\mathsf{Decom}(2^d n)$, the *total size* of the commitment, the *number of rounds* of the commit stage, and the *size of a local opening* on a string of length $2^d \cdot n$, respectively. The commitment consists of a description of a hash function $h \in \mathcal{H}$ (whose size is denoted by $\ell(n)$), the root-hash value of a Merkle hash-tree (which is of size n), a function $g \in \mathcal{G}$ (which is of size $4n$), and the recursive commitment. The opening for an index $i \in [2^d]$ consists of a block (of size n), the full i-th path (which consists of $2dn$ bits), and the recursive opening.

Recall that the protocol for committing on strings of length $2^d n$ uses (recursively) a protocol for committing on strings of length $2^{d'} n$, where

$$d' = d - (\log n - \log z - \log d) = d - \log(n/(zd)).$$

Moreover, the commitment protocol on strings of length n has communication complexity n, consists of a single round and the opening is n bits. Thus, the total number of recursive call will be

$$\left\lceil \frac{d}{\log(n/(zd))} \right\rceil.$$

We get that the total number of communication rounds is bounded by

$$\mathsf{Rounds}(2^d n) \leq 3 + \mathsf{Rounds}(2^{d'} n) \leq \cdots \leq \left\lceil \frac{3d}{\log(n/(zd))} \right\rceil + O(1)$$

Actually, since each recursive call consists of three messages (except the base of the recursion), we can join the last round of every iteration with the first round in the next one. Therefore, we can get the improved bound

$$\mathsf{Rounds}(2^d n) \leq \frac{2d}{\log(n/(zd))} + O(1)$$

The commit protocol between $\mathcal{S}$ and $\mathcal{R}$

The sender $\mathcal{S}$ has string $s = s_1 \dots s_{2^d}$ where $s_i \in \{0,1\}^n$ for all $i \in [2^d]$.

1. $\mathcal{R} \Rightarrow \mathcal{S}$: Samples $h \leftarrow \mathcal{H}$ and sends h.
2. $\mathcal{S} \Rightarrow \mathcal{R}$: Compute a Merkle hash-tree T of s using h and send the root-hash y. Let π_v be the hash value in the tree for node $v \in T$, and let $P_i = \left\{\pi_v \circ \pi_{N(v)}\right\}_{v \in \mathsf{path}_i}$ for all $i \in [d]$.
3. $\mathcal{R} \Rightarrow \mathcal{S}$: Sample $g \leftarrow \mathcal{G}$ and send g.
4. $\mathcal{S} \Leftrightarrow \mathcal{R}$: Recursively interact to commit on the string $s' = u_1 \circ \dots \circ u_{2^d}$, where $u_i = \left\{g(\pi_v \circ \pi_{N(v)})\right\}_{v \in \mathsf{path}_i}$. Notice that $|s'| = 2^d \cdot dz = 2^{d'} n$, where $d' = d - (\log n - \log z - \log d)$. Denote the outputs of the sender and receiver by
$$((D_1, \dots, D_{2^{d'} n}), C) \leftarrow \langle \mathcal{S}(1^n, s'), \mathcal{R} \rangle.$$

The output of each party

- $\mathcal{R}$'s output: The receiver $\mathcal{R}$ outputs $\mathsf{com} = (h, y, g, C)$.
- $\mathcal{S}$'s output: The sender $\mathcal{S}$ outputs $(\mathsf{decom}_1, \dots, \mathsf{decom}_{2^d})$, where decom_i is defined as follows: Let i' be the index in s' of the block containing u_i. Set $\mathsf{decom}_i = (s_i, P_i, D_{i'})$.[a]

The local-opening procedure $\mathcal{V}$

The verifier $\mathcal{V}$ has an index $i \in [2^d]$, a decommitment decom_i, and a commitment com.

1. Verify that s_i appears in P_i in the right location.
2. Verify that the values in P_i are consistent with respect to h and y.
3. Compute $u_i = \left\{g(\pi_v \circ \pi_{N(v)})\right\}_{v \in \mathsf{path}_i}$, where $P_i = \left\{\pi_v \circ \pi_{N(v)}\right\}_{v \in \mathsf{path}_i}$. Recursively compute $u_i' \leftarrow \mathcal{V}(i', D_{i'}, C)$ and verify that $u_i' = u_i$.
4. If all tests pass, output s_i. Otherwise, output $\perp$.

[a]We assume without loss of generality that each u_i is contained in a single block (otherwise, we pad the string).

Figure 1: Our commitment protocol for strings of length $2^d \cdot n$.

The size of a commitment is bounded by

$$\begin{aligned}\mathsf{Size}(2^d n) &\leq \ell(n) + 5n + \mathsf{Size}(2^{d'} n) \\ &\leq \left\lceil \frac{d \cdot (\ell(n) + 5n)}{\log(n/(zd))} \right\rceil \leq d \cdot (\ell(n) + 5n).\end{aligned}$$

The size of a local opening is bounded by

$$\begin{aligned}\mathsf{Decom}(2^d n) &\le n + 2dn + \mathsf{Decom}(2^{d'} n)\\ &\le \left\lceil \frac{3d^2 n}{\log(n/(zd))} \right\rceil \le 3d^2 n.\end{aligned}$$

Computational-binding. We show that our protocol (t_d, ϵ_d)-computationally-binding for strings of length $2^d \cdot n$. We assume that the k-MCRH is (t, ϵ)-secure and that the internal protocol for strings of length $2^{d'} n$ is $(t_{d'}, \epsilon_{d'})$-computationally binding. We set ϵ_d to satisfy the following recursive relation:

$$\epsilon_d = \frac{4k^2}{2^{z-d}} + 4\epsilon + 4\epsilon_{d'}.$$

Plugging in the number of rounds our recursion takes, we get that indeed

$$\epsilon_d \le 2^{\frac{6d}{\log n - \log z - \log d}} \cdot \left(\frac{4k^2}{2^{z-d}} + 4\epsilon \right).$$

Let $\mathcal{S}^*$ be a (cheating) adversary that runs in time t_d and breaks the binding of the protocol, namely, with probability ϵ_d, $\mathcal{S}^*$ is able to (locally) open the commitment in two ways without getting caught. That is, after the commitment stage, there is an index $i \in [2^d]$ for which the adversary $\mathcal{S}^*$ is able to open the corresponding block in two *different* ways with probability ϵ_d:

$$\Pr_{\mathcal{S}^*, \mathcal{R}} \left[\begin{array}{c} (i, \mathsf{decom}_i^0, \mathsf{decom}_i^1, \mathsf{com}) \leftarrow \langle \mathcal{S}^*(1^n), \mathcal{R} \rangle \text{ and} \\ \perp \ne \mathcal{V}(i, \mathsf{decom}_i, \mathsf{com}) \ne \mathcal{V}(i, \mathsf{decom}'_i, \mathsf{com}) \ne \perp \end{array} \right] \ge \epsilon_d, \tag{1}$$

where the probability is taken over the random coins of both $\mathcal{S}^*$ and $\mathcal{R}$. The randomness of $\mathcal{R}$ consists of uniformly chosen functions $h \leftarrow \mathcal{H}$ and a function $g \leftarrow \mathcal{G}$, so we can rewrite Equation (1) as

$$\Pr_{\substack{\mathcal{S}^*, h \leftarrow \mathcal{H}, \\ g \leftarrow \mathcal{G}}} \left[\begin{array}{c} (i, \mathsf{decom}_i^0, \mathsf{decom}_i^1, \mathsf{com}) \leftarrow \langle \mathcal{S}^*(1^n), (h, g) \rangle \\ \text{and } \perp \ne \mathcal{V}(i, \mathsf{decom}_i, \mathsf{com}) \ne \mathcal{V}(i, \mathsf{decom}'_i, \mathsf{com}) \ne \perp \end{array} \right] \ge \epsilon_d. \tag{2}$$

We will show how to construct an adversary $\mathcal{A}$ that runs in time at most t and finds a k-wise collision relative to a randomly chosen hash function $h \leftarrow \mathcal{H}$. The procedure $\mathcal{A}$ will run the protocol $\langle \mathcal{S}^*(1^n), (h, g) \rangle$ with the given h and a function g chosen uniformly at random and get (with good probability) two valid and different openings for some index i. Then, $\mathcal{A}$ will *partially* rewind the adversary $\mathcal{S}^*$ to the stage after he received the hash function h and replied with the root hash y (of s), and execute it again but with a fresh function $g \leftarrow \mathcal{G}$. With noticeable probability, this will again result with two valid openings for some (possibly different) index i. Repeating this process enough times, $\mathcal{A}$ will translate a large number of different (yet valid) decommitments into a large number of collisions relative to h. The fact that these collisions are distinct (with good probability) will follow from the fact that the g's are sampled independently in every repetition.

We introduce some useful notation. Recall that $\mathcal{S}^*$ is the sender that is able to locally open its commitment in two different ways. We slightly abuse notation and also think of $\mathcal{S}^*$ as a distribution over senders (this is without loss of generality since we can assume it chooses all of its randomness ahead of time). For an index $i^* \in [R]$, string $y \in \{0,1\}^n$, we denote by $\mathcal{S}^*|_{h,y}$ a distribution over all senders $\mathcal{S}^*$ in which the first message received is h and the reply is the root-hash y. For a string y that is sampled by choosing $h \leftarrow \mathcal{H}$ and running $\mathcal{S}^*$ for one round to obtain y, the adversary $\mathcal{S}^*|_{h,y}$ is uniformly chosen from all possible continuations of the adversary $\mathcal{S}^*$ given these first two messages. Given this notation, we can write the adversary $\mathcal{S}^*$ as a pair of two distributions $(Y_h, \mathcal{S}^*|_{h,Y_h})$, where Y_h is the distribution of the first message $\mathcal{S}^*$ sends in response to h, and $\mathcal{S}^*|_{h,Y_h}$ is the distribution over the rest of the protocol.

In a high-level, our algorithm $\mathcal{A}$ maintains a binary trees of depth d and 2^d leaves, in which each node v is associated with a set of labels S_v. At the beginning, each such set S_v is initialized to be empty. The algorithm $\mathcal{A}$ uses $\mathcal{S}^*$ to get many valid pairs of openings decom_i^0 and decom_i^1 for an index i:

$$\mathsf{decom}_i^0 = (s_i^0, P_i^0, D_{i'}^0) \text{ and } \mathsf{decom}_i^1 = (s_i^1, P_i^1, D_{i'}^1).$$

that are consistent with a commitment:

$$\mathsf{com} = (h, y, g, C).$$

Since both openings are valid and $s_i \neq s_i'$, it must be that 1 s_i^0 (resp., s_i^1) appears in P_i^0 (resp., P_i^1) and (2) P_i^0 and P_i^1 are consistent with the root-hash y. Thus, it must be that $P_i^0 \neq P_i^1$ and there is a node v on the path path_i that is the first (going from the root to the leaves) non-trivial collision between P_i^0 and P_i^1. Now, by the induction hypothesis, the probability that $\mathcal{S}^*$ cheats in the internal decommitment is small, and since g is sampled uniformly at every iteration, we show that it must be a *new* collision that did not appear before. We will identify the location of the collision at a node v and add this collision to the set S_v. The algorithm $\mathcal{A}$ is formally defined in Figure 2.

We analyze the success probability of $\mathcal{A}$ in finding a k-wise collision. We say that $h \in \mathcal{H}$ and $y \in \mathsf{supp}(Y_h)$ are "good" if it holds that

$$\Pr_{\mathcal{S}^*|_{h,y}, g \leftarrow \mathcal{G}}\left[\begin{array}{l}(i, \mathsf{decom}_i^0, \mathsf{decom}_i^1, \mathsf{com}) \leftarrow \langle \mathcal{S}^*|_{h,y}(1^n), g\rangle \text{ and} \\ \perp \neq \mathcal{V}(i, \mathsf{decom}_i, \mathsf{com}) \neq \mathcal{V}(i, \mathsf{decom}_i', \mathsf{com}) \neq \perp\end{array}\right] \geq \frac{\epsilon_d}{2}. \tag{3}$$

By an averaging argument, Equation (2) implies that there are many such "good" (h, y) pairs, namely:

$$\Pr_{h \leftarrow \mathcal{H}, y \leftarrow Y_h}\left[\Pr_{\mathcal{S}^*|_{h,y}, g \leftarrow \mathcal{G}}\left[\begin{array}{l}(i, \mathsf{decom}_i^0, \mathsf{decom}_i^1, \mathsf{com}) \leftarrow \langle \mathcal{S}^*|_{h,y}(1^n), g\rangle \text{ and} \\ \perp \neq \mathcal{V}(i, \mathsf{decom}_i, \mathsf{com}) \neq \mathcal{V}(i, \mathsf{decom}_i', \mathsf{com}) \neq \perp\end{array}\right] \geq \frac{\epsilon_d}{2}\right] \geq \frac{\epsilon_d}{2}. \tag{4}$$

For the rest of the proof we condition on the event that $\mathcal{A}$ sampled an h and got back a y such that (h, y) are good.

We say that iteration j of $\mathcal{A}$ is *successful* $\mathcal{A}$ passes step 3d and reaches step 3e: if $\mathcal{S}_{h,y}^*$ outputs (at Item 3b) a valid pair of openings $(i, \mathsf{decom}_i^0, \mathsf{decom}_i^1, \mathsf{com})$ that pass the tests. We denote this event by SUC_i. Recall that in every successful iteration, $\mathcal{A}$ adds X and Y to the set S_v for some node v. Notice that S_v is a set so it does not contain duplicate elements. Since we condition on (h, y) being good and cheating in the internal commitment is possible with probability at most $\epsilon_{d'}$, we have that

$$\Pr_{\mathcal{A}}[\mathsf{SUC}_j] \geq \frac{\epsilon_d}{2} - \epsilon_{d'}. \tag{5}$$

```
The adversary $\mathcal{A}(1^n, h)$:

1. For all nodes $v$, set $S_v = \emptyset$ to be the empty set.
2. Send to $\mathcal{S}^*$ the function $h$ and receive a root-hash $y \in \{0,1\}^n$.
3. Do the following $T = 50nk2^d/\epsilon_d^2$ times:
   (a) Sample $g \leftarrow \mathcal{G}$.
   (b) Obtain $(i, \mathsf{decom}_i^0, \mathsf{decom}_i^1, \mathsf{com}) \leftarrow \langle \mathcal{S}^*|_{h,y}(1^n), g \rangle$.
   (c) Parse $\mathsf{decom}_i^0 = \left(s_i^0, P_i^0, D_{i'}^0\right)$ and $\mathsf{decom}_i^1 = \left(s_i^1, P_i^1, D_{i'}^1\right)$. Parse $\mathsf{com} = (h, y, g, C)$.
   (d) If any of the following occurs, continue to the next iteration:
       i. $\perp \neq \mathcal{V}(i, \mathsf{decom}_i^0, \mathsf{com}) \neq \mathcal{V}(i, \mathsf{decom}_i^1, \mathsf{com}) \neq \perp$.
       ii. $\perp \neq \mathcal{V}(i', D_{i'}^0, C) \neq \mathcal{V}(i', D_{i'}^1, C) \neq \perp$.
   (e) Let $v$ the node of the first (from the root to the leaves) non-trivial collision between
       $P_i^0$ and $P_i^1$. Let $X = \pi_v^0, \pi_{N(v)}^0$ and $Y = \pi_v^1, \pi_{N(v)}^1$ be the values of the collision for $P_i^0$
       and $P_i^1$, respectively. Add $X$ and $Y$ to $S_v$.
   (f) If there exists a node $v$ for which $|S_v| \geq k$, then output $S_v$ and halt.
4. Output $\perp$.
```

Figure 2: The algorithm $\mathcal{A}$ to find a k-wise collision in $\mathcal{H}$.

We say that iteration j is *effective* if for the chosen $g \in \mathcal{G}$ (at Item 3a) it holds that $\forall v, \forall X \neq Y \in S_v \colon g(X) \neq g(Y)$. We denote this event by EFF_i. Since g is pairwise independent, there are 2^d internal nodes in the tree, and at most k^2 pairs in each node (since $|S_v| \leq k$), it holds that

$$\Pr_{\mathcal{A}}[\mathsf{EFF}_i] = 1 - \Pr_{\mathcal{A}}[\exists v, \exists X \neq Y \in S_v \colon g(X) = g(Y)] \geq 1 - \frac{2^d \cdot k^2}{2^z}. \tag{6}$$

Combining Equations (5) and (6) together with the inequality that $\epsilon_d \geq 4k^2/2^{z-d} + 4\epsilon_{d'}$, we get that an iteration i is both successful and effective with probability at least

$$\Pr_{\mathcal{A}}[\mathsf{SUC}_i \text{ and } \mathsf{EFF}_i] \geq 1 - \Pr_{\mathcal{A}}[\neg\mathsf{SUC}_i] - \Pr_{\mathcal{A}}[\neg\mathsf{EFF}_i] \geq \frac{\epsilon_d}{2} - \epsilon_{d'} - \frac{k^2}{2^{z-d}} \geq \frac{\epsilon_d}{4}. \tag{7}$$

We observe that every iteration which is both successful and effective, increases the size of the set S_v for some node v. Indeed, let X^* and Y^* be the pair added to some set S_v in this iteration. Since the iteration is successful it holds that $X^* \neq Y^*$ and furthermore these values are consistent with the commitment, namely, $g(X^*) = g(Y^*)$ for the g that was chosen in this iteration. However, since the iteration is effective, there are no other two values $X, Y \in S_v$ that map to the same value relative to g. This means that either X^* or Y^* were not in the set S_v before.

To complete the proof we have to show that (with high probability) there are at least $k \cdot 2^d$ iterations that are both successful and effective. Indeed, if this happens, there is a node v such that $|S_v| \geq k$ and this means that we found a k-wise collision. Denote by X_i the event that the

i-th iteration is both successful and effective. Denote by $Y_n = \sum_{i=1}^{n}(X_i - \mathbf{E}[X_i])$ and $Y_0 = 0$. We observe that the sequence $Y_0, Y_1, \ldots$ is a martingale. Indeed:

$$\mathbf{E}[Y_n \mid Y_0, \ldots, Y_{n-1}] = \mathbf{E}[Y_{n-1} + X_n - \mathbf{E}[X_n] \mid Y_0, \ldots, Y_{n-1}] = Y_{n-1}$$

By linearity of expectation and Equation (11):

$$\mathbf{E}\left[\sum_{i=1}^{T} X_i\right] = \sum_{i=1}^{T} \mathbf{E}[X_i] \geq T \cdot \frac{\epsilon_d}{4}.$$

Since $T = 50nk2^d/\epsilon_d^2$ then $T\epsilon_d^2/50 \geq n$ and $T\epsilon_d/n \geq k2^d$. Thus, by the Azuma bound (see e.g., [AS08, Theorem 7.2.1]):

$$\begin{aligned}
\Pr_{\mathcal{A}}\left[\sum_{i=1}^{T} X_i < k2^d\right] &= \Pr_{\mathcal{A}}\left[\sum_{i=1}^{T} X_i - \sum_{i=1}^{T} \mathbf{E}[X_i] < k2^d - \sum_{i=1}^{T} \mathbf{E}[X_i]\right] \\
&\leq \Pr_{\mathcal{A}}[Y_T < -(T\epsilon_d/4 - T\epsilon_d/n)] \\
&\leq \Pr_{\mathcal{A}}[Y_T < -T\epsilon_d/5] \\
&\leq \Pr_{\mathcal{A}}\left[Y_T < -(\sqrt{T}\epsilon_d/5) \cdot \sqrt{T}\right] \\
&\leq 2^{-T\epsilon_d^2/50} \leq 2^{-n}.
\end{aligned}$$

Summarizing the proof, since $\epsilon_d > 4\epsilon$, we get that $\mathcal{A}$ finds a collision in $h \leftarrow \mathcal{H}$ with probability

$$\begin{aligned}
&\Pr_{h,\mathcal{A}}[\mathcal{A}(1^n, h) \text{ outputs a } k\text{-wise collision}] = \Pr_{h,\mathcal{A}}[\exists v \colon |S_v| \geq k] \\
&\geq \Pr_{h,\mathcal{A}}\left[(h, y) \text{ are good } \wedge\ k \cdot 2^d \textit{ successful} \text{ and } \textit{effective} \text{ iterations}\right] \\
&\geq \frac{\epsilon_d}{2} \cdot \left(1 - \frac{1}{2^n}\right) \geq \frac{\epsilon_d}{4} > \epsilon,
\end{aligned}$$

The running time of $\mathcal{A}$ is bounded by T executions of $\mathcal{S}^*$ and a fixed polynomial factor $p(n)$ (that bounds the time it takes to sample $g \in \mathcal{G}$ and the additional simple operations $\mathcal{A}$ makes including the evaluation of h inside $\mathcal{V}$). In total, since $\mathcal{S}^*$ runs in time t', then $\mathcal{A}$'s running time is at most

$$T \cdot t' \cdot p(n) = \frac{50nk2^d \cdot p(n)}{\epsilon_d^2} \cdot t' \leq t.$$

Remark 1 (Optimization I: recycling h)**.** *In the recursive step of our protocol, we can use the same hash function h as in the first step of the recursion. This saves sending its description (which is of size $\ell(n)$ at every iteration and the resulting size of a commitment is $(O(\ell(n) + d^2n)$.*

Remark 2 (Optimization II: almost-universal hashing)**.** *In our protocol we used a pairwise independent hash function whose description size is proportional to their input size. This costs us in communication. We could save communication by using almost-universal hash functions whose description size is proportional to their output size.*

5.2 Getting statistical-hiding generically

We show how to transform any short commitment scheme Π that is computationally binding (but perhaps not hiding) to a new scheme Π' that is short, computationally binding and *statistically hiding*. Moreover, if Π admits a local-opening, then Π' admits a local-opening as well. Our transformation is information theoretic, adds no additional assumptions and preserves (up to constant factors) the security, the number of rounds and communication complexity of the original scheme (up to a small constant factor).

High-level idea. The underlying idea of our transformation is to leverage the fact that the commitment protocol Π is short. Specifically, when committing to a long string $s \in \{0,1\}^{n^c}$ for some large $c \in \mathbb{N}$, the communication is very short: $\Lambda(n)$ bits for a fixed polynomial function.[10] Thus, when we commit to s, a large portion of s is not revealed to the receiver by the protocol so this part of s is statistically hidden. The question is how to make sure that all of s is hidden.

Our solution takes advantage of the fact that some fraction of s remains hidden. We commit on a random string r that is independent of s and slightly longer. Then, we extract from r the remaining randomness r' *given the communication of the protocol* using a strong extractor (see Section 3.2). Finally, we commit on the string $s \oplus r'$. We need to show that this scheme is computationally-binding and statistically-hiding. The former follows from the computational-binding of the original scheme. The latter follows from the fact that r' is completely hidden to the receiver and it masks the value of s.

The commitment protocol $(\mathcal{S}, \mathcal{R}, \mathcal{V})$ that we describe uses the underlying commitment protocol $(\widehat{\mathcal{S}}, \widehat{\mathcal{R}}, \widehat{\mathcal{V}})$ (which is *not* statistically-hiding) and a family of pairwise-independent functions

$$\mathcal{G} = \{g \colon \{0,1\}^{8\Lambda(n)} \to \{0,1\}\}.$$

Recall that each function $g \in \mathcal{G}$ can be represented with $8\Lambda(n)$ bits.

In Section 5.1, we presented a commitment scheme that has also linear dependence on c. Specifically, the length of a commitment there is at most

$$(c-1)\log n \cdot (\ell(n) + 5n),$$

where $\ell(n)$ is the size of the descriptions of a hash function. For simplicity of presentation in this section we suppress this linear term in c and assume that $\Lambda(n)$ is independent of c. The precise description of the protocol is given in Figure 3.

Computational-binding. Assume that after the commitment stage, there is an index $i \in [N]$ for which a malicious sender $\mathcal{S}^*$ is able to open the corresponding bit in two *different* ways with probability $\epsilon = \epsilon(n)$:

$$\Pr_{\mathcal{S}^*, \mathcal{R}}\left[\begin{array}{c} (i, \mathsf{decom}_i, \mathsf{decom}'_i, \mathsf{com}) \leftarrow \langle \mathcal{S}^*(1^n), \mathcal{R}\rangle \text{ and} \\ \perp \neq \mathcal{V}(i, \mathsf{decom}_i, \mathsf{com}) \neq \mathcal{V}(i, \mathsf{decom}'_i, \mathsf{com}) \neq \perp \end{array}\right] \geq \epsilon,$$

where the probability is taken over the random coins of both $\mathcal{S}^*$ and $\mathcal{R}$. We sketch how to use this malicious sender to design a malicious sender $\widehat{\mathcal{S}}^*$ that breaks the computational binding of the basic protocol with $\widehat{\mathcal{R}}$.

[10] Our protocol from Section 5.1 has an additional linear dependence on c, but we will ignore this in this section to simplify notation.

The statistically-hiding commit protocol between $\mathcal{S}$ and $\mathcal{R}$

The sender $\mathcal{S}$ has string $s \in \{0,1\}^N$.

1. $\mathcal{S} \Leftrightarrow \mathcal{R}$: Interact according to $(\widehat{\mathcal{S}}, \widehat{\mathcal{R}})$ to obtain a commitment on a random string $r^1 = r_1 \circ \ldots \circ r_N \leftarrow \{0,1\}^{8\Lambda(n)N}$. Denote the outputs of the sender and receiver by
$$((\mathsf{decom}^1_1, \ldots, \mathsf{decom}^1_N), \mathsf{com}^1) \leftarrow \langle \mathcal{S}(1^n, r^1), \mathcal{R} \rangle,$$
where decom^1_i corresponds to the opening of the whole string r_i.
2. $\mathcal{S} \Leftrightarrow \mathcal{R}$: Interact according to $(\widehat{\mathcal{S}}, \widehat{\mathcal{R}})$ to obtain a commitment on a string $r^2 = g_1 \circ \ldots \circ g_N \in \{0,1\}^{8\Lambda(n)N}$, where $g_i \leftarrow \mathcal{G}$ is sampled at random. Denote the outputs of the sender and receiver by
$$((\mathsf{decom}^2_1, \ldots, \mathsf{decom}^2_N), \mathsf{com}^2) \leftarrow \langle \mathcal{S}(1^n, r^2), \mathcal{R} \rangle,$$
where decom^2_i corresponds to the opening of the whole string g_i.
3. $\mathcal{S} \Leftrightarrow \mathcal{R}$: Interact according to $(\widehat{\mathcal{S}}, \widehat{\mathcal{R}})$ to obtain a commitment on the string $r^3 = s \oplus (g_1(r_1) \circ \ldots \circ g_N(r_N))$. Denote the outputs of the sender and receiver by
$$((\mathsf{decom}^3_1, \ldots, \mathsf{decom}^3_N), \mathsf{com}^3) \leftarrow \langle \mathcal{S}(1^n, r^3), \mathcal{R} \rangle,$$
where decom^3_i corresponds to the opening of the bit $s_i \oplus g_i(r_i)$.

The output of each party

- $\mathcal{S}$: The sender $\mathcal{S}$ outputs $(\mathsf{decom}_1, \ldots, \mathsf{decom}_N)$, where
$$\mathsf{decom}_i = \left(\mathsf{decom}^1_i, \mathsf{decom}^2_i, \mathsf{decom}^3_i\right)$$
- $\mathcal{R}$: The receiver $\mathcal{R}$ outputs $\mathsf{com} = \left(\mathsf{com}^1, \mathsf{com}^2, \mathsf{com}^3\right)$.

The local-opening procedure $\mathcal{V}$

The verifier $\mathcal{V}$ has an index $i \in [N]$, a decommitment decom_i, and a commitment com.

1. Execute $\widehat{r^*_i} \leftarrow \widehat{\mathcal{V}}(i, \mathsf{decom}^1_i, \mathsf{com}^1)$.
2. Execute $\widehat{g^*_i} \leftarrow \widehat{\mathcal{V}}(i, \mathsf{decom}^2_i, \mathsf{com}^2)$.
3. Execute $\widehat{u^*_i} \leftarrow \widehat{\mathcal{V}}(i, \mathsf{decom}^3_i, \mathsf{com}^3)$.
4. If all executions succeed (output non-$\perp$), output $\widehat{g^*_i}(\widehat{r^*_i}) \oplus \widehat{u^*_i}$, where $\widehat{g^*_i}$ is treated as a description of a hash function in $\mathcal{G}$. Otherwise, output $\perp$.

Figure 3: A statistically-hiding commitment protocol.

Recall that $\mathsf{decom}_i = (\widehat{r_i^*}, \widehat{g_i^*}, \widehat{u_i^*})$ and $\mathsf{decom}'_i = (\widehat{r_i^*}', \widehat{g_i^*}', \widehat{u_i^*}')$ and the fact that $\mathcal{V}$ succeeds and outputs non-$\perp$ and different values, implies that

$$\widehat{g_i^*}(\widehat{r_i^*}) \oplus \widehat{u_i^*} \neq \widehat{g_i^{*\prime}}(\widehat{r_i^{*\prime}}) \oplus \widehat{u_i^{*\prime}}.$$

Thus, either $\widehat{g_i^*}(\widehat{r_i^*}) \neq \widehat{g_i^{*\prime}}(\widehat{r_i^{*\prime}})$ or $\widehat{u_i^*} \neq \widehat{u_i^{*\prime}}$. This means that one of the following three must occur: (1) $\widehat{r_i^*} \neq \widehat{r_i^{*\prime}}$, (2) $\widehat{g_i^*} \neq \widehat{g_i^{*\prime}}$, or (3) $\widehat{u_i^*} \neq \widehat{u_i^{*\prime}}$. Let us assume that the first event occurs and the rest of the cases are handled analogously. In this case, our malicious sender $\widehat{\mathcal{S}}^*$ will simply simulate the first step of $\mathcal{S}^*$ (namely, the part in which it commits on a random string r^1). The rest of the protocol will be executed with a simulation of $\mathcal{R}$, without real interaction. As a result, with probability ϵ, the output of $\langle \widehat{\mathcal{S}}^*(1^{n+N}), \widehat{\mathcal{R}} \rangle$ results with a single commitment and two openings which $\widehat{\mathcal{V}}$ will accept as valid. This is a contradiction to the computational binding of the basic protocol which implies that ϵ cannot be noticeable.

In conclusion, if the original protocol is (t, ϵ)-secure, our new protocol is $(t, \epsilon/3)$-secure.

Statistical-hiding. We will show that for every (deterministic) adversary $\mathcal{R}^*$ and every distinct $s_0, s_1 \in \{0,1\}^N$, the ensembles $\{\mathsf{view}_{\langle \mathcal{S}(s_0), \mathcal{R}^* \rangle}(n)\}$ and $\{\mathsf{view}_{\langle \mathcal{S}(s_1), \mathcal{R}^* \rangle}(n)\}$ have statistical distance at most $\rho(n) = 2^{-n}$ for all sufficiently large $n \in \mathbb{N}$.

Recall that the view of $\mathcal{R}^*$ consists of three parts where the first two parts are independent of the message and the last part depends on it. For a message s, denote the (distribution of) the view of $\mathcal{R}^*$ by a triple of random variables $(R^{\mathsf{view}}, G^{\mathsf{view}}, E_s^{\mathsf{view}})$. We further denote by (R, G, E_s) the distribution over (r^1, r^2, r^3). Using this notation, we have that

1. $R = R_1, \ldots, R_N$, where each R_i is the uniform distribution over $8\Lambda(n)$ strings.
2. $G = G_1, \ldots, G_N$, where each G_i is the uniform distribution over pairwise independent functions samples from $\mathcal{G}$. Each such G_i is supported on strings of length $8\Lambda(n)$.
3. $E_s = G_1(R_1) \oplus s, \ldots, G_N(R_N) \oplus s$.

Since each R_i is essentially the uniform distribution over strings of length $8\Lambda(n)$, it holds that $\mathsf{H}_\infty(R) = 8\Lambda(n)$. Moreover, since the whole communication of the protocol is bounded by $3\Lambda(n)$, by Proposition 2, we have that for every $i \in [N]$:

$$\begin{aligned}\mathsf{H}_\infty(R_i \mid (R^{\mathsf{view}}, G^{\mathsf{view}}, E_s^{\mathsf{view}})) &= 8\Lambda(n) - |(R^{\mathsf{view}}, G^{\mathsf{view}}, E_s^{\mathsf{view}})| \\ &\geq 8\Lambda(n) - 3\Lambda(n) = 5\Lambda(n).\end{aligned}$$

Thus, by the leftover hash lemma (see Proposition 1), for every $i \in [N]$:

$$\Delta((R^{\mathsf{view}}, G_i, G_i(R_i)), (R^{\mathsf{view}}, G_i, U)) \leq 2^{-2n},$$

where U is the uniform distribution on a single bit. Moreover, for every $s \in \{0,1\}^N$:

$$\Delta((R^{\mathsf{view}}, G_i, U), (R^{\mathsf{view}}, G_i, U \oplus s)) = 0.$$

Therefore, by a triangle inequality, for every two different messages $s_0, s_1 \in \{0,1\}^N$,

$$\Delta((R^{\mathsf{view}}, G_i, G_i(R_i) \oplus s_0), (R^{\mathsf{view}}, G_i, G_i(R_i) \oplus s_1)) \leq 2 \cdot 2^{-2n}.$$

Taking a union bound over the different i's, we get that

$$\begin{aligned}
&\Delta(\{\mathsf{view}_{\langle \mathcal{S}(s_0),\mathcal{R}^*\rangle}(n)\},\{\mathsf{view}_{\langle \mathcal{S}(s_1),\mathcal{R}^*\rangle}(n)\})\\
&\quad= \Delta((R^{\mathsf{view}},G^{\mathsf{view}},E_s^{\mathsf{view}}),(R^{\mathsf{view}},G^{\mathsf{view}},E_s^{\mathsf{view}}))\\
&\quad\le N\cdot\Delta((R^{\mathsf{view}},G_i,G_i(R_i)\oplus s_0),(R^{\mathsf{view}},G_i,G_i(R_i)\oplus s_1))\\
&\quad\le 2N\cdot 2^{-2n}\le 2^{-n}.
\end{aligned}$$

Efficiency. The (1) size of a commitment, (2) of a decommitment, or (3) or the number of rounds in $(\mathcal{S},\mathcal{R},\mathcal{V})$ are all the same as that of $(\widehat{\mathcal{S}},\widehat{\mathcal{R}},\widehat{\mathcal{V}})$ up to a multiplicative factor of 3, respectively.

6 Four-Round Short Commitments from Multi-CRH

We show how to construct a *4-round* short commitment protocol based on a family of k-MCRH functions. Compared to the protocol from Section 5, this protocol has *no* local opening and is secure only for *constant* values of k. However, it consists only of 4 rounds. Furthermore, using techniques similar to Section 5.2 the protocol can be made also statistically-hiding which suffices for some applications such as statistical zero-knowledge arguments [BCC88].

We discuss methods that allow us to prove security even for polynomial values of k in Section 6 below.

Theorem 3. *Assume that there exists a (t,ϵ)-secure k-MCRH $\mathcal{H}$ for a constant k in which every function can be described using $\ell=\ell(n)$ bits. For any $c\in\mathbb{N}$, there is a commitment protocol for strings of length n^c with the following properties:*

1. *(t',ϵ')-computationally-binding for $\epsilon'=4\epsilon+O(k^{c\log n})/2^n$ and $t'=t\cdot\epsilon'^2/(O(k^{c\log n})\cdot p(n))$, where $p(\cdot)$ is some fixed polynomial function.*
2. *takes 4 rounds (i.e., 4 messages).*
3. *the commitment has length $\ell+O(n)$.*

Proof. We describe a commitment protocol for a string $s=s_1\dots s_{2^d}$ of length $2^d\cdot n$ for $d=(c-1)\cdot\log n$. We show that our protocol is computationally-binding and getting statistical-hiding can be done using the generic transformation from Section 5.2. Our protocol uses a Merkle hash-tree as described in Section 5. The main observation made in this construction is that before using the Merkle hash-tree we can use a special type of encodings, called list-recoverable codes (as defined in Definition 4) to get meaningful security. Let $C\colon\{0,1\}^{2^dn}\to(\{0,1\}^{2n})^{2^{d'}}$ be an (ℓ,L)-list-recoverable code for $\ell=k^d$, $L=O(k^d)$, and $d'=O(\log n)$ with some large enough hidden constants. Such a code exists by Theorem 1 (with efficient encoding and list-recovery). Let G be a family of (2^{-n})-almost pairwise-independent functions mapping strings of length 2^dn to strings of length n (see Definition 3):

$$\mathcal{G}=\{g\colon\{0,1\}^{2^dn}\to\{0,1\}^n\}.$$

Recall that every $g\in\mathcal{G}$ can be described using at most $2n+\log(2^dn)+\log(2^n)\le 4n$ bits.

The commitment protocol for a string $s=s_1\dots s_{2^d}$ of length $2^d\cdot n$ for $d=(c-1)\cdot\log n$ works as follows. The receiver first sends a description of an $h\leftarrow\mathcal{H}$ which is a k-MCRH to the sender. The sender then computes the encoding $C(s)$ of s and computes the Merkle hash tree of $s' = C(s)$ (and not of s). The sender sends the root of the hash tree y to the receiver Figure 4. The receiver replies with a hash function $g \leftarrow \mathcal{G}$ and finally the sender replies with $u = g(s)$. The opening is done is the natural way by letting the sender reveal s to the receiver who then simulates the computation and makes sure that the messages y and u are consistent. See Figure 4 for the precise description.

The commit protocol between $\mathcal{S}$ and $\mathcal{R}$

The sender $\mathcal{S}$ has string $s = s_1 \dots s_{2^d}$ where $s_i \in \{0,1\}^n$ for all $i \in [2^d]$.

1. $\mathcal{R} \Rightarrow \mathcal{S}$: Samples $h \leftarrow \mathcal{H}$ and sends h.
2. $\mathcal{S} \Rightarrow \mathcal{R}$: Compute $s' = C(s)$ and a Merkle hash-tree T of s' using h and send the root-hash y.
3. $\mathcal{R} \Rightarrow \mathcal{S}$: Sample $g \leftarrow \mathcal{G}$ and send g.
4. $\mathcal{S} \Leftrightarrow \mathcal{R}$: Send $u = g(s)$ to the receiver

The output of the receive is $\mathsf{com} = (h, y, g, u)$.

The verifier $\mathcal{V}$ gets the input s simulates the sender to verify y and u.

Figure 4: Our four-round commitment protocol for strings of length $2^d \cdot n$.

By the description of the protocol, it is easy to see that the protocol consists of 4-rounds and has communication complexity of $\ell + n + 4n + n = \ell + 6n$ bits. Also, it is easy to see that for the honest sender the verification succeeds with probability 1. We proceed with the proof of security.

Assume that there is a malicious sender $\mathcal{S}^*$ that runs in time $t' = t \cdot \epsilon'^2/(L \cdot p(n))$ (for some large enough polynomial $p(\cdot)$) and is able open a commitment in two *different* ways with probability $\epsilon' = \epsilon'(n) \geq 4\epsilon + 10L/2^n$:

$$\Pr_{\mathcal{S}^*,\mathcal{R}}\left[\begin{array}{c}(x, x', \mathsf{com}) \leftarrow \langle \mathcal{S}^*(1^n), \mathcal{R}\rangle \text{ and} \\ \bot \neq \mathcal{V}(x, \mathsf{com}) \neq \mathcal{V}(x', \mathsf{com}) \neq \bot\end{array}\right] \geq \epsilon',$$

where the probability is taken over the random coins of both $\mathcal{S}^*$ and $\mathcal{R}$. We sketch how to use this malicious sender to come up with more than k values that collide relative to the MCRH h. The randomness of $\mathcal{R}$ consists of uniformly chosen functions $h \leftarrow \mathcal{H}$ and a function $g \leftarrow \mathcal{G}$, so we can rewrite the above equation as

$$\Pr_{\substack{\mathcal{S}^*, h \leftarrow \mathcal{H}, \\ g \leftarrow \mathcal{G}}}\left[\begin{array}{c}(x, x', \mathsf{com}) \leftarrow \langle \mathcal{S}^*(1^n), (h, g)\rangle \\ \text{and } \bot \neq \mathcal{V}(x, \mathsf{com}) \neq \mathcal{V}(x', \mathsf{com}) \neq \bot\end{array}\right] \geq \epsilon'. \tag{8}$$

The high-level idea is that an adversary that succeeds in the task above must succeed in the task for a random h and a bunch of *different* g's. Simulating $\mathcal{S}^*$ on sufficiently many g's (but the same h) we obtain a list of different x's that map to the same value under $h(\cdot)$. Then, one of two must happen: (1) either the adversary is able to create many different consistent codewords using few options per coordinate, or (2) there is one entry with many different values. Using the properties of the list-recoverable code we rule-out option (1). We are then left with option (2) which means that there must be a location where the adversary uses many openings which translates to a k-wise collision relative to h for some node on the path between the location and the root.

We formalize this intuition next by defining an algorithm $\mathcal{A}$ that gets as input 1^n and $h \leftarrow \mathcal{H}$ and finds a k-wise collision. We say that $h \in \mathcal{H}$ is Good if

$$\Pr_{\mathcal{S}^*, g \leftarrow \mathcal{G}}\left[\begin{array}{c}(x, x', \mathsf{com}) \leftarrow \langle \mathcal{S}^*(1^n), (h, g)\rangle \\ \text{and } \perp \neq \mathcal{V}(x, \mathsf{com}) \neq \mathcal{V}(x', \mathsf{com}) \neq \perp\end{array}\right] \geq \frac{\epsilon'}{2}.$$

Thus, by averaging,

$$\Pr_{h \leftarrow \mathcal{H}}[h \text{ is } \mathsf{Good}] \geq \frac{\epsilon'}{2}.$$

Our procedure $\mathcal{A}$ simulates the receiver in the protocol and then iteratively partially rewinds the sender with different choices of g.

The algorithm $\mathcal{A}(1^n, h)$:

1. Run $\mathcal{S}^*$'s first part with the hash function h to obtain a root hash y.
2. Initialize a set $\mathcal{X} = \emptyset$.
3. Do the following $T = 100n \cdot L/\epsilon'^2$ times:
   (a) Sample a fresh $g \leftarrow \mathcal{G}$.
   (b) Run $\mathcal{S}^*$'s second part (by rewinding) given the new hash function g to obtain a value u.
   (c) Open the commitment to get a values x and x' and add them to the set X.
   (d) If the set $\mathcal{X}$ contains more than L values, quit the loop.
4. Find a coordinate for which there are ℓ different values in the codewords of messages in $\mathcal{X}$. In this coordinate, traverse the path to the root of the hash tree and find a location that has more than k values that has to the same value. Output these values.

The running time of $\mathcal{A}$ is polynomial in n, $L = O(k^d)$, and $1/\epsilon'$. Altogether, since k is constant, $d = O(\log n)$, and $1/\epsilon'$ is a fixed polynomials in n, the running time of $\mathcal{A}$ is a fixed polynomial in n. We show that the algorithm $\mathcal{A}$ succeeds in finding a k-wise collision with noticeable probability. In the following claim we show that with noticeable probability the execution of the loop in the description of the adversary $\mathcal{A}$ terminates with a large set $\mathcal{X}$ (which will then translate into a large enough collision). The proof of this claim is similar to the proof in Theorem 2.

Claim 1. $\Pr_{\mathcal{A}}[|\mathcal{X}| > L \mid h \textit{ is } \mathsf{Good}] \geq 1 - 2^{-n}$.

Proof. Since h is Good, with probability $\epsilon'/2$, in any iteration we obtain x and x' for which verification succeeds yet outputs two different values (x and x' in our case). We say that iteration i is *successful*, denoted by SUC_i if this iteration satisfies the above (it outputs two different x and x' that pass verification). Thus, it holds that

$$\Pr_{\mathcal{A}}[\mathsf{SUC}_i] \geq \frac{\epsilon'}{2}. \tag{9}$$

We further say that iteration j is *effective*, denoted by EFF_i, if for the chosen $g \in \mathcal{G}$, the outputted x and x', and every $x'' \in \mathcal{X}$, it holds that $g(x) \neq g(x'')$. Since g is (2^{-n})-almost pairwise independent and the set X is of size at most L, it holds that

$$\Pr_{\mathcal{A}}[\mathsf{EFF}_i] \geq 1 - \Pr_{\mathcal{A}}\left[\exists x'' \in \mathcal{X} \colon g(x) = g(x'')\right] \geq 1 - \frac{L}{2^n} - \frac{1}{2^n} \geq 1 - \frac{L+1}{2^n}. \tag{10}$$

Combining the inequalities in Eq. (9) and in Eq. (10) with the assumption that $\epsilon' \geq 10L/2^n$, we get that

$$\Pr_{\mathcal{A}}[\mathsf{SUC}_i \text{ and } \mathsf{EFF}_i] \geq 1 - \Pr_{\mathcal{A}}[\neg\mathsf{SUC}_i] - \Pr_{\mathcal{A}}[\neg\mathsf{EFF}_i] \geq \frac{\epsilon'}{2} - \frac{L+1}{2^n} \geq \frac{\epsilon'}{4}. \tag{11}$$

Observe that every iteration which is both successful and effective increases the size of the set $\mathcal{X}$ by at least one. To see this, fix a pair x and x' that the malicious sender $\mathcal{S}^*$ outputs in a fixed iteration (which is both successful and effective). Since the iteration is successful it holds that $x \neq x'$ and furthermore these values are consistent with the g chosen in that iteration, namely, $g(x) = g(x')$. Since the iteration is effective, there is no other value $x'' \in \mathcal{X}$ that maps to the same value relative to g. This means that either x or x' were not in the set $\mathcal{X}$ before the current iteration.

To complete the proof we need to show that there are many successful and effective iterations. Denote by X_i the event that the i-th iteration is both successful and effective. Define $Y_n = \sum_{i=1}^{n}(X_i - \mathbf{E}[X_i])$ and $Y_0 = 0$. The sequence $Y_0, Y_1, \ldots$ is a martingale:

$$\mathbf{E}[Y_n \mid Y_0, \ldots, Y_{n-1}] = \mathbf{E}[Y_{n-1} + X_n - \mathbf{E}[X_n] \mid Y_0, \ldots, Y_{n-1}] = Y_{n-1}$$

and furthermore, by linearity of expectation

$$\sum_{i=1}^{T} \mathbf{E}[X_i] = \mathbf{E}\left[\sum_{i=1}^{T} X_i\right] \geq T \cdot \frac{\epsilon'}{4}.$$

Thus, by the Azuma bound (see e.g., [AS08, Theorem 7.2.1]):

$$\begin{aligned}
\Pr_{\mathcal{A}}\left[|\mathcal{X}| < nk^d\right] &= \Pr_{\mathcal{A}}\left[\sum_{i=1}^{T} X_i < 4k^d\right] \\
&= \Pr_{\mathcal{A}}\left[\sum_{i=1}^{T} X_i - \sum_{i=1}^{T} \mathbf{E}[X_i] < 4k^d - \sum_{i=1}^{T} \mathbf{E}[X_i]\right] \\
&\leq \Pr_{\mathcal{A}}\left[Y_T < -(T\epsilon'/4 - T\epsilon'/n)\right] \\
&\leq \Pr_{\mathcal{A}}\left[Y_T < -T\epsilon'/5\right] \\
&\leq \Pr_{\mathcal{A}}\left[Y_T < -(\sqrt{T}\epsilon'/5) \cdot \sqrt{T}\right] \\
&\leq 2^{-T\epsilon'^2/50} \leq 2^{-n}.
\end{aligned}$$

This completes the proof of the claim. □

We now condition on an execution of $\mathcal{A}$ terminating with $|\mathcal{X}| > L$. This event implies that the malicious sender $\mathcal{S}^*$ came up with at least $|\mathcal{X}|$ different messages x all of which correspond to codewords of C that are consistent with the root hash. Each such x implies a different codeword $C(x)$ for our (ℓ, L)-list-recoverable code. By the definition of list-recoverability and since we have $|\mathcal{X}| > L$ different codewords, there must be a coordinate $i \in [2^{d'}]$ in the code that has more than ℓ different values. Since $\ell = k^d$ and the depth of the Merkle hash tree is d, by the pigeonhole principle, there is a coordinate (in which symbols are composed of $2n$ bits) in which there must be a location on the path from this coordinate to the root hash that has at least k different values that hash to the same value and thus give us a k-wise collision relative to h.

We now ready to complete the proof. The algorithm $\mathcal{A}$ runs in time $\mathsf{poly}(n) \cdot t' \cdot L/\epsilon'^2$ and finds a k-wise collision in $h \leftarrow \mathcal{H}$ with probability

$$\begin{aligned}\Pr_{h,\mathcal{A}}\left[\mathcal{A}(1^n, h) \text{ outputs a } k\text{-wise collision}\right] &= \Pr_{h,\mathcal{A}}[|\mathcal{X}| > L] \\ &\geq \Pr_{h,\mathcal{A}}[|\mathcal{X}| > L \mid h \text{ is } \mathsf{Good}] \cdot \Pr_{h \leftarrow \mathcal{H}}[h \text{ is } \mathsf{Good}] \\ &\geq \left(1 - \frac{1}{2^n}\right) \cdot \frac{\epsilon'}{2} \geq \epsilon,\end{aligned}$$

which is a contradiction to the security of the MCRH $\mathcal{H}$.

□

Supporting arbitrary larger k. The reason why we could prove security only for constant values of k stems from the fact that our adversary $\mathcal{A}$ for the MCRH runs in time proportional to k^d. The source of this term in the running time is that our Merkle hash tree in the construction is of depth $d = O(\log n)$ and when counting the number of possible openings per coordinate in a leaf, we get k^d possibilities. Our adversary $\mathcal{A}$ basically "collects" this number of different openings for some coordinate and thereby finds a k-wise collision. If k is super-constant the running time of $\mathcal{A}$ becomes super-polynomial.

There are two paths we can take to bypass this. One is to assume super-polynomial security of the MCRH and allow our adversary to run in super-polynomial time. The second assumption is to assume that we start with a stronger MCRH that compresses by a polynomial factor (i.e., from $n^{1+\Omega(1)}$ to n) rather than by a factor of 2. This will cause the Merkle hash tree to be of constant depth. Under either of the assumptions, we can support polynomial values of k (in n). However, both assumptions are rather strong on the underlying MCRH and we thus prefer to avoid them. Moreover, a hash function from $2n$ bits to n bits is the basic building block in most standards of cryptographic hash functions such as the ones published by NIST (e.g., the SHA-x family).

7 Separating Multi-CRH from CRH

In this section we rule out fully black-box constructions (see Definition 9) of CRH functions from k-MCRH functions for $k > 2$. Our proof technique is inspired by the works of Asharov and Segev [AS16] and Haitner et al. [HHRS15], that are based in turn on ideas originating in the works of Simon [Sim98] Gennaro et al. [GGKT05]) and Wee [Wee07].

We present an oracle Γ relative to which there exists a 3-multi-collision-resistant hash function, but any collision-resistant hash function can be easily broken. The theorem below is stated and proved only for the case of standard CRH and 3-MCRH. The ideas in this proof naturally extend to a separation of k-MCRH from $(k+1)$-MCRH for other values of k.

Theorem 4. *There is no fully black-box construction of a collision-resistant hash function family from a 3-multi-collision-resistant hash function family mapping $2n$ bits to n bits.*

We describe the oracle Γ next. It is in fact a distribution over pairs of oracles $(f, \mathsf{ColFinder}^f)$ in which f is a uniformly chosen function and $\mathsf{ColFinder}$ is an oracle that finds pair-wise collisions.

The oracle Γ. The oracle Γ consists of a pair of oracles $(f, \mathsf{ColFinder}^f)$:

- **The function $f = \{f_n\}_{n\in\mathbb{N}}$:** For every n, the function f_n is a uniformly chosen function from $2n$ bits to n bits.
- **The function $\mathsf{ColFinder}^f$:** This function consists of an infinite collection of permutations, where for every $n \in \mathbb{N}$ and every circuit $C^f \colon \{0,1\}^{2n} \to \{0,1\}^n$, there are two uniformly and independently chosen permutations π_C^1, π_C^2 over $\{0,1\}^{2n}$. When $\mathsf{ColFinder}^f$ is given C^f as input, it sets $x_1 = \pi_C^1(0^{2n})$ and then computes $x_2 = \pi_C^2(t)$ for the lexicographically smallest $t \in \{0,1\}^{2n}$ such that $C(x_1) = C(x_2)$ for every. Finally, $\mathsf{ColFinder}^f$ outputs x_1, x_2.

Using $\mathsf{ColFinder}^f$ it is easy to break any collision-resistant hash function.

Lemma 1. *There exists a probabilistic polynomial-time algorithm $\mathcal{A}$ such that for any function f, any $n \in \mathbb{N}$ and any oracle-aided circuit C^f mapping strings of length $2n$ to strings of length n, it holds that*

$$\Pr_{\mathsf{ColFinder}}\left[\begin{matrix} x_1, x_2 \text{ are distinct and} \\ C^f(\sigma, x_1) = C^f(\sigma, x_2) \end{matrix} \middle| (x_1, x_2) \leftarrow \mathcal{A}^{f,\mathsf{ColFinder}}(1^n, C)\right] \geq \frac{3}{4}.$$

Proof. Fix n and f. The algorithm $\mathcal{A}$, on input C, sends it to the oracle $\mathsf{ColFinder}^f$, and receives back a pair (x_1, x_2). By the definition of $\mathsf{ColFinder}^f$, it holds that $C(x_1) = C(x_2)$.

Since C maps strings of length $2n$ to strings of length n, there are at most $n \cdot 2^n$ values of $x \in \{0,1\}^{2n}$ for which $|f^{-1}(f(x))| \leq n$. Therefore, for at least $2^{2n} - n \cdot 2^n$ values of x it holds that $|f^{-1}(f(x))| > n$. Namely,

$$\Pr_{x \leftarrow \{0,1\}^n}\left[|f^{-1}(f(x))| > 1\right] \geq 1 - \frac{n}{2^n}.$$

So, with probability $1 - \frac{n}{2^n}$ the first x_1 chosen by $\mathsf{ColFinder}^f$ is one such that $|f^{-1}(f(x))| > n$. Then, x_2 is chosen uniformly at random from the set of x's in $f^{-1}(f(x))$. The probability of choosing $x_2 \neq x_1$ is at least $(n-1)/n$. Altogether, the probability of choosing two distinct (colliding) values x_1, x_2 is at least $(1 - n/2^n) \cdot (n-1)/n \geq 3/4$. $\square$

Next, we show that f is a 3-MCRH hash function family even in the presence of $\mathsf{ColFinder}$.

Lemma 2. *For every probabilistic oracle-aided algorithm $\mathcal{A}$ that makes polynomially-many queries, there exists a negligible function $\mathsf{negl}(\cdot)$ such that*

$$\Pr_{f,\mathsf{ColFinder}}\left[\begin{matrix} x_1, \ldots, x_3 \text{ are distinct and} \\ f(x_1) = \cdots = f(x_3) \end{matrix} \middle| (x_1, \ldots, x_3) \leftarrow \mathcal{A}^{\Gamma}(1^n)\right] \leq \mathsf{negl}(n).$$

We show how Theorem 4 follows from Lemmas 1 and 2 and then in Section 7.1 prove Lemma 2.

Proof of Theorem 4. Assume there exists a black-box construction (see Definition 9) of a 3-MCRH function from a CRH function $\mathcal{H}'$ specified by $(\mathcal{H}.\mathsf{G}, \mathcal{H}.\mathsf{E}, M)$.

By Lemma 1, there exists an oracle-aided algorithm $\mathcal{A}$ and a polynomial $p(\cdot)$, such that for every f, it holds that

$$\Pr\left[\begin{array}{c} x_1, x_2 \textit{ are distinct and} \\ \mathcal{H}'.\mathsf{E}^f(\sigma, x_1) = \mathcal{H}'.\mathsf{E}^f(\sigma, x_2) \end{array} \middle| \begin{array}{c} \sigma \leftarrow \mathcal{H}'.\mathsf{G}^f(1^n) \\ (x_1, x_2) \leftarrow \mathcal{A}^{f,\mathsf{ColFinder}}(1^n, \sigma) \end{array}\right] \geq \frac{1}{p(n)}.$$

Definition 9 says that we can use $\mathcal{A}$ to get an algorithms that finds a 3-wise collision relative to f. Specifically, there exists a probabilistic polynomial-time algorithm M that has oracle access to $\mathcal{A}$, and a polynomial $p'(\cdot)$ such that

$$\Pr_{\mathsf{ColFinder}}\left[\begin{array}{c} x_1, \dots x_3 \textit{ are distinct and} \\ f(x_1) = \dots = f(x_3) \end{array} \middle| (x_1, \dots, x_3) \leftarrow M^{\mathcal{A}, f}(1^n)\right] \geq \frac{1}{p'(n)}.$$

Viewing $M^{\mathcal{A}}$ as a single procedure, its total running time is bounded by some fixed polynomial. This is because every operation of M can be an oracle query to $\mathcal{A}$ with a security parameter polynomial in n and the running time of $\mathcal{A}$ is polynomial. This is a contradiction to Lemma 2. □

7.1 Proof of Lemma 2

We prove a slightly stronger lemma, namely, we show that for every algorithm $\mathcal{A}$ that performs polynomially many queries, and every fixing of the oracle f for all inputs except n, it holds that

$$\Pr_{f_n, \mathsf{ColFinder}}\left[\begin{array}{c} x_1, \dots, x_3 \textit{ are distinct and} \\ f(x_1) = \dots = f(x_3) \end{array} \middle| (x_1, \dots, x_3) \leftarrow \mathcal{A}^{f,\mathsf{ColFinder}}(1^n)\right] \leq \mathsf{negl}(n).$$

Denote by y the value of $f(x_1)$ (which is equal to the value of $f(x_2)$ and $f(x_3)$). Consider an algorithm $\mathcal{A}$ that has oracle access to f and $\mathsf{ColFinder}$. It can gain information about x's that map to y under f_n either through direct queries to f_n or via indirect queries made by $\mathsf{ColFinder}$ to f_n.

We formalize the latter by defining an event $\mathsf{3\text{-}multi\text{-}hit}$ that occurs when one of the f_n gates during the computation of $\mathsf{ColFinder}$ hits a value y that closes a 3-wise collision under f_n.

Consider a query C^f to $\mathsf{ColFinder}$ with output (w_1, w_2). Without loss of generality, we assume that A directly queries f_n on all elements in the computation of $C^f(w_1)$ and $C^f(w_2)$. Moreover, we assume that $\mathcal{A}$ never performs the same query twice.

Definition 11 (3-multi-hit)**.** *Let C^f be a query to* $\mathsf{ColFinder}$*, and let $x_1, \dots, x_q$ be all the previous f_n queries performed by $\mathcal{A}$ with responses $y_1, \dots, y_q$. The query of C^f to* $\mathsf{ColFinder}$ *outputs (w_1, w_2). Let $x_{q+1}, \dots, x_{q'}$ be the queries performed by the computation of $C^f(w_1)$ and $C^f(w_2)$ and let $y_{q+1}, \dots, y_{q'}$ be the responses.*

We say that the query C produced a 3-multi-hit *if there exist $1 \leq i_1 < \dots < i_k \leq q'$ and $q < i_k$ such that $y_{i_1} = \dots = y_{i_k}$. If $q < i_1 < \dots \leq i_k \leq q'$ then we say that a* 3-strong-multi-hit *occurred.*

Moreover, we define the event $\mathcal{A}$-3-wins that occurs when the adversary $\mathcal{A}$ is successful in finding a k-wise collision relative to f_n.

Definition 12 ($\mathcal{A}$-3-wins)**.** *We say that the event $\mathcal{A}$*-3-wins *occurred if $x_1, \dots, x_3$ are distinct and $f_n(x_1) = \dots = f_n(x_3)$ where $(x_1, \dots, x_3) \leftarrow \mathcal{A}^{f,\mathsf{ColFinder}}(1^n)$.*

Using the above definitions, we can write

$$\Pr_{f_n,\mathsf{ColFinder}}[\mathcal{A}\text{-}3\text{-}\mathsf{wins}] = \Pr_{f_n,\mathsf{ColFinder}}[\mathcal{A}\text{-}3\text{-}\mathsf{wins} \wedge 3\text{-}\mathsf{multi\text{-}hit}] + \Pr_{f_n,\mathsf{ColFinder}}\left[\mathcal{A}\text{-}3\text{-}\mathsf{wins} \wedge \overline{3\text{-}\mathsf{multi\text{-}hit}}\right]$$

The proof proceeds by bounding both of these terms. Bounding the second term is done by a compression argument. We show that if 3-multi-hit does not occur and nevertheless $\mathcal{A}$ successfully finds a 3-wise collision (with noticeable probability), then f_n can be succinctly represented given $\mathcal{A}$. This is a contradiction to the fact that f_n is a random function that does not admit such short representation. To bound the first term, we reduce it to the second one by showing that any occurrence of a 3-multi-hit can be simulated. That is, we show that if $\mathcal{A}$ succeeds in finding a 3-wise collision with probability $\epsilon(n)$, then there exists a machine M that succeeds in finding a 3-wise collision (with about the same probability as $\mathcal{A}$ and with proportional query complexity) but without producing any 3-multi-hit. This is summarized in the following two claims:

Claim 2. *For every q-query algorithm $\mathcal{A}^{f,\mathsf{ColFinder}}$, where $q(n)$ is a polynomial, it holds that*

$$\Pr_{f_n,\mathsf{ColFinder}}\left[\mathcal{A}\text{-}3\text{-}\mathsf{wins} \wedge \overline{3\text{-}\mathsf{multi\text{-}hit}}\right] \le \mathsf{negl}(n).$$

Claim 3. *Fix the oracle f and a poly-query algorithm $\mathcal{A}^{f,\mathsf{ColFinder}}$, There exists a poly-query oracle-aided machine M procedure such that if for a polynomial $q(\cdot)$ such that*

$$\Pr_{f_n,\mathsf{ColFinder}}[\mathcal{A}\text{-}3\text{-}\mathsf{wins} \wedge 3\text{-}\mathsf{multi\text{-}hit}] \ge \frac{1}{q(n)}$$

for infinitely many n's, then for some polynomial $q'(\cdot)$

$$\Pr_{f_n,\mathsf{ColFinder}}\left[M\text{-}3\text{-}\mathsf{wins} \wedge \overline{3\text{-}\mathsf{multi\text{-}hit}}\right] \ge \frac{1}{q'(n)}$$

for infinitely many n's.

From Claims 2 and 3 we can complete the proof of Theorem 4. Indeed, assume that there exists a poly-query algorithm $\mathcal{A}$, and a fixing of the oracle f for all inputs except n, such that for a polynomial $q(\cdot)$ it holds that

$$\Pr_{f_n,\mathsf{ColFinder}}[\mathcal{A}\text{-}3\text{-}\mathsf{wins}] > 1/q(n)$$

for infinitely many n's. This means that

$$\begin{aligned}1/q(n) < \Pr_{f_n,\mathsf{ColFinder}}&[\mathcal{A}\text{-}3\text{-}\mathsf{wins} \wedge 3\text{-}\mathsf{multi\text{-}hit}] + \\ \Pr_{f_n,\mathsf{ColFinder}}&[\mathcal{A}\text{-}3\text{-}\mathsf{wins} \wedge \overline{3\text{-}\mathsf{multi\text{-}hit}}].\end{aligned}$$

By Claim 2, the second term is bounded by some negligible term $\mathsf{negl}(n)$. This implies that

$$\frac{1}{q(n)} - \mathsf{negl}(n) < \Pr_{f_n,\mathsf{ColFinder}}[\mathcal{A}\text{-}3\text{-}\mathsf{wins} \wedge 3\text{-}\mathsf{multi\text{-}hit}].$$

By averaging,

$$\Pr_{f_n}\left[\Pr_{\mathsf{ColFinder}}[\mathcal{A}\text{-}\mathsf{3\text{-}wins} \wedge \mathsf{3\text{-}multi\text{-}hit}] \geq \frac{1}{8q(n)}\right] \geq \frac{1}{8q(n)}.$$

Let us call f_n *good* if the inner event happens. By Claim 3, we get a poly-query oracle-aided machine M such that for every good f_n it holds that M finds a 3-wise collision without making $\mathsf{3\text{-}multi\text{-}hit}$ occurring. Thus, we get that for some polynomial $q'(\cdot)$,

$$\begin{aligned}
&\Pr_{f_n,\mathsf{ColFinder}}[\mathcal{A}\text{-}\mathsf{3\text{-}wins} \wedge \overline{\mathsf{3\text{-}multi\text{-}hit}}] \\
&\geq \Pr_{f_n}[f_n \text{ is good}] \cdot \Pr_{\mathsf{ColFinder}}[M\text{-}\mathsf{3\text{-}wins} \wedge \overline{\mathsf{3\text{-}multi\text{-}hit}} \mid f_n \text{ is good}] \\
&\geq \frac{1}{8q(n)} \cdot \frac{1}{q'(n)}
\end{aligned}$$

which is a contradiction to Claim 2 since this is not a negligible function.

The proofs of Claims 2 and 3 appear in Sections 7.2 and 7.3, respectively.

7.2 Proof of Claim 2

Fix $\mathsf{ColFinder}$, an inverse-polynomial ϵ and recall that f is fixed at all input lengths except n. For any prefix s of size $n/2$ let $f_n|_s(x) = f(s \circ x)$. We show that for every f_n if

$$\Pr_{s \leftarrow \{0,1\}^{n/2}}[\mathcal{A}^{f_n|_s}\text{-}\mathsf{3\text{-}wins} \wedge \overline{\mathsf{3\text{-}multi\text{-}hit}}] \geq \epsilon,$$

then f_n has a succinct description.

Succinct description of f_n. The description of f_n consists of a truth-table Z a set of prefixes S, a set of images Y. We will be running $\mathcal{A}$ on $f_n|_s$ for many different prefixes s. Notice that if $f_n|_s(x) = f_n|_s(x')$ then $f_n(s \circ x) = f_n(s \circ x')$. Denote by S the set of prefixes s for which $\mathcal{A}$ successfully finds a collision for $f_n|_s$.

For every $s \in S$, we follow the computation of $\mathcal{A}^{f_n|_s,\mathsf{ColFinder}}$ and define $F_s = \{x_1, \dots, x_\ell\}$ to be the set of all inputs of $f_n|_s$-gates during this computation. Finally, $\mathcal{A}^{f_n|_s,\mathsf{ColFinder}}$ outputs a 3-wise collision $\{x_1, x_2, x_3\}$. Assume without loss of generality, that $\mathcal{A}$ directly queried x_1, x_2 and x_3, and never queries the same input twice. Let $I_s = \{i_1, i_2, i_3\}$ be the index of the queries performed by $\mathcal{A}$ to x_1, x_2, x_3 respectively. Assume that $i_1 < i_2 < i_3$. Let $X_s = \{s \circ x_3\}$. Continue to the next iteration which is defined analogously.

Let $X = \cup_{s \in S} X_s$. Notice that $|X| = |S|$, and that since ℓ is polynomial in n we can write each of the indexes using $c \log n$ bits for some constant $c \in \mathbb{N}$. Moreover, by the assumption we have that $|S| \geq \epsilon 2^{n/2}$.

We write f_n as follows: first we write the set S, then for each $s \in S$ we write I_s, and finally we write the truth-table of $Z = \{0,1\}^{2n} \setminus X$. The total size (in bits) is at most

$$\begin{aligned}
|f_n| &\leq \log \binom{2^{n/2}}{|S|} + |S| 3c \log n + (2^{2n} - |X|)n \\
&\leq (n/2)|S| + 2|S| - |S| \log |S| + |S| 3c \log n + n2^{2n} - n|S| \\
&= n2^{2n} - |S|(\log |S| - 3c \log n + n/2 - 2) \\
&\leq n2^{2n} - n.
\end{aligned}$$

We next show that this representation is indeed sufficient to answer f_n and $\mathsf{ColFinder}$ queries. Given x as input, either we know it explicitly (via the table Z), or we reconstructed it before, or we "hit" it by one of the indices i_1, i_2, i_3. For each $s \in S$, taken in lexicographical increasing order, the simulator reads I_s and runs $\mathcal{A}^{f_n|_s, \mathsf{ColFinder}}$. The latter requires answering $f_n|_s$-queries and $\mathsf{ColFinder}$ queries. Finally, $\mathcal{A}$ outputs x_1, x_2, x_3, where x_1 has its value in the truth-table Z, and thus we can reconstruct the values for x_2 and x_3.

Answering $f_n|_s$ queries. We show how to simulate a direct $f_n|_s$ query made by $\mathcal{A}$. Assume that $\mathcal{A}$ makes such a query on input x. If it is already known (either because $s \circ x$ is in Z or because we already reconstructed it), then the simulator can return it. If it is i_3-th query, then the simulator takes the answer of the i_1-th query and answers with the same value.

Answering $\mathsf{ColFinder}$-queries. On input oracle-aided circuit $C = C_i^{f_n|_s}$, the simulator computes $w_i = \pi_1^C(0)$ and begins with evaluating $C(w_i)$. Then, it enumerates over all $t \in \{0,1\}^{2n}$ in lexicographically increasing order, and stops with the first $w'_i = \pi_2^C(t)$ for which $C(w_i)$ can be computed (*i.e.*, it is able to answer all $f_n|_s$-queries as discussed above) and that $C(w_i) = C(w'_i)$. It then answers to $\mathcal{A}$ with the pair (w_i, w'_i).

We now show that the simulator can evaluate $C(w_i), C(w'_i)$ and answers the queries to $f_n|_s$. On a query x to $f_n|_s$-gate, we have:

1. If $s \circ x \notin X$ then the value $f_n|_s(x)$ is given by Z.
2. If $s \circ x \in X \setminus X_s$, then this cannot happen as $\mathcal{A}$ works on a different oracle $f_n|_s$ for different s's.
3. If $s \circ x \in X_s$, then this is impossible, as we condition on the event that $\mathsf{3\text{-}multi\text{-}hit}$ does not occur.

Concluding the proof. Assume that

$$\Pr_{f_n, \mathsf{ColFinder}}\left[\mathcal{A}\text{-}\mathsf{3\text{-}wins} \wedge \overline{\mathsf{3\text{-}multi\text{-}hit}}\right] \geq 2\epsilon,$$

for some inverse-polynomial ϵ. Then, by averaging we get that

$$\Pr_{f_n}[\Pr_{s \leftarrow \{0,1\}^{n/2}}[\mathcal{A}^{f_n|_s}\text{-}\mathsf{3\text{-}wins} \wedge \overline{\mathsf{3\text{-}multi\text{-}hit}}] \geq \epsilon] \geq \epsilon.$$

We have shown that using $\mathcal{A}$ we can represent many of the possible $2^{n \cdot 2^{2n}}$ functions. Thus, the fraction of functions f_n for which

$$\Pr_{s \leftarrow \{0,1\}^{n/2}}[\mathcal{A}^{f_n|_s}\text{-}\mathsf{3\text{-}wins} \wedge \overline{\mathsf{3\text{-}multi\text{-}hit}}] \geq \epsilon,$$

holds is at most

$$\frac{2^{n \cdot 2^{2n} - n}}{2^{n \cdot 2^{2n}}} = 2^{-n}.$$

This implies that any algorithm $\mathcal{A}$ that makes polynomially many queries to f and $\mathsf{ColFinder}$, where its oracle queries to $\mathsf{ColFinder}$ are bounded to circuits of polynomial size, it holds that

$$\Pr_{f_n, \mathsf{ColFinder}}[\mathcal{A}\text{-}\mathsf{3\text{-}wins} \wedge \overline{\mathsf{3\text{-}multi\text{-}hit}}] \leq \mathsf{negl}(n) + 2 \cdot 2^{-n} \leq \mathsf{negl}(n).$$

7.3 Proof of Claim 3

The algorithm M simulates the execution of $\mathcal{A}$ except for one change: whenever $\mathcal{A}$ makes a $\mathsf{ColFinder}$-query with some circuit C^f, the algorithm M first tries to simulate the query and make a $\mathsf{3\text{-}multi\text{-}hit}$ by itself as follows: It evaluates $C^f(z)$ and $C^f(z')$ on random z and z' and without submitting the queries to $\mathsf{ColFinder}$. If during the computation there three inputs x_1, x_2, x_3 (from the computation of $C^f(z)$ and $C^f(z')$ together with queries already performed in the past) that have the same output y, then M halts and outputs this collision. Otherwise, if a $\mathsf{3\text{-}multi\text{-}hit}$ does not occur, M passes the circuit C^f to $\mathsf{ColFinder}$ and outputs whatever it returns.

The algorithm M makes at most as many calls to $\mathsf{ColFinder}$ as $\mathcal{A}$. Moreover, M may perform at most $\mathsf{poly}(n)$ queries to f as direct queries of $\mathcal{A}$ to f, and additional $\mathsf{poly}(n)$ many queries as a result of the simulation of $\mathsf{ColFinder}$.

We split the proof into several cases related to how $\mathcal{A}$-$\mathsf{3\text{-}wins}$ happen. Recall that $\mathcal{A}$ makes queries to $\mathsf{ColFinder}$ and direct queries to f and finally outputs (with good probability) a triple of items x_1, x_2, x_3 that map under f to the same value. This collision happens due to some query to $\mathsf{ColFinder}$ which completes a 3-wise collision.

1. The first case is that the three values x_1, x_2, x_3 all happened during a *single* $\mathsf{ColFinder}$ query. Namely, $\mathcal{A}$ submitted a circuit C to $\mathsf{ColFinder}$ and during the computation of $\mathsf{ColFinder}$ these three queries were made.

   We show that this event has very low probability of happening so we can ignore it.

2. The second case is that the two values x_2, x_3 were queried during a *single* $\mathsf{ColFinder}$ query and joined (in the sense that they all have the same $f(\cdot)$ value) with a single x_1 that was queried sometime before.

   We show that if this event happens, then we can simulate the execution and make sure that we find the same triple but without making the last $\mathsf{ColFinder}$ query.

3. The second case is that one value x_3 was queried during a $\mathsf{ColFinder}$ query and joined with a pair x_1, x_2 that was queried sometime before.

   As in the second case, we show that if this event happens, then we can simulate the execution and make sure that we find the same triple but without making the last $\mathsf{ColFinder}$ query.

We write

$$\Pr_{f_n, \mathsf{ColFinder}}[\mathcal{A}\text{-}\mathsf{3\text{-}wins} \wedge \mathsf{3\text{-}multi\text{-}hit}] = \sum_{i=1}^{3} \Pr_{f_n, \mathsf{ColFinder}}[\mathcal{A}\text{-}\mathsf{3\text{-}wins} \wedge \mathsf{3\text{-}multi\text{-}hit} \wedge \text{case } i \text{ happens}]$$

and prove the claim for each i separately. For $i = 1$, we obtain an algorithm M that is able to output 2 colliding values such that non of them were ever in the output of any f or $\mathsf{ColFinder}$ query. Following similar lines to our compression lemma (see Section 7.2), one can show that such an adversary exists with only negligible probability. So, we ignore this event for the rest of the proof and assume that one of two happens: case 2 or case 3. This means that in the triple that $\mathcal{A}$ outputs (that collide), at least one input was queried on before the last query. The last query completes this triple by querying on either a colliding pair (case 2) or a single element (case 3).

Fix f, and denote by $C_1, \ldots, C_q$, where $q = q(n)$ is some polynomial, the random variables corresponding to the circuits with which $\mathcal{A}$ queries $\mathsf{ColFinder}$. Denote by $(x_1^1, x_2^1), \ldots, (x_1^q, x_2^q)$ the corresponding replies of $\mathsf{ColFinder}$. For every $i \in [q]$, denote by α_i the probability that x_1^i produces a $\mathsf{3\text{-}multi\text{-}hit}$. That is

$$\alpha_i = \Pr[x_1^i \text{ produces a } \mathsf{3\text{-}multi\text{-}hit}].$$

Recall that for every $i \in [q]$ the marginal distributions of x_1^i and x_2^i are uniform. Thus,

$$\forall i \in [q]\colon\ \Pr[x_1^i \text{ produces a } \mathsf{3\text{-}multi\text{-}hit}] = \Pr[x_2^i \text{ produces a } \mathsf{3\text{-}multi\text{-}hit}].$$

For every $i \in [q]$, denote by JUMP_i the event that $\alpha_i > 1/(20q^2)$ and let $\mathsf{JUMP} = \bigcup_{i=1}^{q} \mathsf{JUMP}_i$. Using the above notation, we have that

$$\begin{aligned}\Pr_{\mathsf{ColFinder}}[\mathsf{JUMP}] &\geq \Pr_{\mathsf{ColFinder}}[\mathsf{3\text{-}multi\text{-}hit}] - \Pr_{\mathsf{ColFinder}}[\mathsf{3\text{-}multi\text{-}hit} \mid \overline{\mathsf{JUMP}}] \\ &\geq \frac{1}{q} - \frac{1}{10q} \geq \frac{1}{2q},\end{aligned}$$

where the second inequality follows from the assumption and from the following claim.

Claim 4. $\Pr_{\mathsf{ColFinder}}[\mathsf{3\text{-}multi\text{-}hit} \mid \overline{\mathsf{JUMP}}] \leq \frac{1}{10q}$.

Proof. We divide into several cases depending on how many collisions we obtained in the last query.

$$\begin{aligned}\Pr_{\mathsf{ColFinder}}[\mathsf{3\text{-}multi\text{-}hit} \mid \overline{\mathsf{JUMP}}] = &\Pr_{\mathsf{ColFinder}}[\mathsf{3\text{-}multi\text{-}hit} \wedge \text{ case } 2 \mid \overline{\mathsf{JUMP}}] + \\ &\Pr_{\mathsf{ColFinder}}[\mathsf{3\text{-}multi\text{-}hit} \wedge \text{ case } 3 \mid \overline{\mathsf{JUMP}}].\end{aligned}$$

We analyze each term separately conditioned on $\overline{\mathsf{JUMP}}$. Observe that when $\mathsf{3\text{-}multi\text{-}hit} \wedge$ case 3 occurs, then it must be that our simulator will be able to simulate this query and obtain the collision by itself:

$$\begin{aligned}\Pr_{\mathsf{ColFinder}}[\mathsf{3\text{-}multi\text{-}hit} \wedge \text{ case } 3] &\leq \sum_{i=1}^{q} \Pr[\text{One of } x_1^i, x_2^i \text{ produces a } \mathsf{3\text{-}multi\text{-}hit}] \\ &\leq 2 \cdot q \cdot \frac{1}{30q^2} = \frac{1}{20q}.\end{aligned}$$

The case $\mathsf{3\text{-}multi\text{-}hit} \wedge$ case 3 is similar and we have that

$$\begin{aligned}\Pr_{\mathsf{ColFinder}}[\mathsf{3\text{-}multi\text{-}hit} \wedge \text{ case } 2] &\leq \sum_{i=1}^{q} \Pr[\text{One of } x_1^i, x_2^i \text{ produces a } \mathsf{3\text{-}multi\text{-}hit}] \\ &\leq 2 \cdot q \cdot \frac{1}{30q^2} = \frac{1}{20q}.\end{aligned}$$

Summing up, we get that

$$\Pr_{\mathsf{ColFinder}}[\mathsf{3\text{-}multi\text{-}hit} \mid \overline{\mathsf{JUMP}}] \leq \frac{1}{10q}.$$

□

Assume for now that the event JUMP occurs and denote by i^* the minimal $i \in [q]$ for which JUMP_i occurs. Consider the following two events:

1. None of the $C_1, \ldots, C_{i^*-1}$ produced a $\mathsf{3\text{-}multi\text{-}hit}$. Recall that the events $\mathsf{JUMP}_1, \ldots, \mathsf{JUMP}_{i^*-1}$ did not happen since i^* is minimal. The probability of this event happening is at least $1 - 1/(10q)$ (similarly to the computation in Claim 4).

2. Given C_{i^*}, the algorithm M produces a $\mathsf{3\text{-}multi\text{-}hit}$ (without submitting the query to $\mathsf{ColFinder}$). This event happens with probability at least $(\alpha_{i^*})^2/2 > (1/(20q^2))^2/2$. Indeed, in case 2, this happens with probability $\alpha_{i^*} > 1/(20q^2)$, but in case 3, we need to succeed in both simulation steps of M and to output different values.

Notice that these two events are independent since $\mathsf{ColFinder}$ uses an independent permutation per circuit. Thus we get that

$$\begin{aligned}
&\Pr_{\mathsf{ColFinder}}[M^{f,\mathsf{ColFinder}}(1^n) \text{ finds a 3-wise collision} \wedge \overline{\mathsf{3\text{-}multi\text{-}hit}}] \\
&\quad\geq \Pr_{\mathsf{ColFinder}}[M^{f,\mathsf{ColFinder}}(1^n) \text{ finds a 3-wise collision} \wedge \overline{\mathsf{3\text{-}multi\text{-}hit}} \mid \mathsf{JUMP}] \cdot \Pr_{\mathsf{ColFinder}}[\mathsf{JUMP}] \\
&\quad\geq \left(1 - \frac{1}{10q}\right) \cdot \frac{1}{2(20q^2)^2} \cdot \frac{1}{2q} \geq \frac{1}{q^7}.
\end{aligned}$$

8 UOWHFs, MCRHs and Short Commitments

We explore the relationship between short commitments, MCRHs and universal one-way hash functions (see Definition 6). Our main message is that short commitment protocols (with some requirements listed below) directly imply UOWHFs. The transformation is efficient in the sense that a description of a hash function corresponds to the messages sent by the receiver and evaluation of the function is done by executing the protocol. In some cases this gives a way to construct a UOWHF which is more efficient than the direct construction based on one-way functions or permutations [NY89, Rom90] (see comparison below). In Remark 4 we sketch how to get an efficient construction of a UOWHF that operates on inputs of fixed length directly from an MCRH.

Theorem 5. *Any short commitment protocol in which the receiver is public-coin yields a universal one-way hash function with the following properties:*

1. *The key size is the total number of (public) coins sent by the receiver.*

2. *The length of inputs that the UOWHF supports is the length of messages the commitment protocol commits to and the length of the output is the amount of bits sent from the sender to the receiver.*

3. *The evaluation of a hash function amounts to a single execution of the commitment protocol.*

Plugging in our construction of short commitments from MCRH functions from Theorem 2, we obtain a new construction of a UOWHF for messages of length $n2^d$ starting with a k-MCRH for a polynomial $k = k(n)$. The key size in the resulting family is proportional to the number of bits sent from the receiver to the sender: $\ell + O(d \cdot n/\log n)$ bits, where ℓ is the size of an MCRH key.[11]

[11] The overhead in the key size can be improved if the pairwise hash function is replaced by an almost uniform hash function as described in Remark 2

Using our construction of short commitments from MCRH functions from Theorem 3, we get a new construction of a UOWHF for messages of length $n2^d$ starting from a k-MCRH for any constant k. The key size in the resulting family is $\ell + O(n)$ bits. Notice that this term is independent of d and the hidden constant in the big "O" is pretty small.[12]

Comparison with previous constructions. Starting with a standard collision resistant hash function on short inputs, it is known how a collision resistant hash function on long inputs (based on the so called Merkle-Damgård iterated construction) [Mer89b, Dam89]. This directly implies a UOWHF. The key size in the resulting construction is optimal: it is just a key for a single collision resistant hash. However, when starting with weaker building blocks the key size grows.

Naor and Yung [NY89] suggested a solution based on a tree hash (similar to a construction of Wegman and Carter for universal hash functions [WC81]). In their scheme, hashing a message of length $n2^d$ is done by a balanced tree such that in the i-th level $n2^{d-i}$ bits are hashed into $n2^{d-i-1}$ bits by applying the same basic compression function 2^{d-i-1} times. Each level in the tree requires its own basic compression function which results with a total of d keys for the basic UOWHF. Thus, the total size of a key in the resulting function is $d\ell$ bits, where ℓ is the bit size of a key in the basic UOWHF.

In case that the keys of the basic scheme (ℓ above) are rather long, Shoup [Sho00], following on the XOR tree hash of Bellare and Rogaway [BR97], offered the following elegant scheme to transform a fixed-input UOWHF into a UOWHF that supports arbitrary long inputs. Given an input of 2^d blocks each of size n, for $i = 1, \ldots, 2^d$ we compute $c_i = h((c_{i-1} \oplus s_{\kappa(i)}) \circ m_i)$, where $s_0, \ldots, s_d$ are uniformly random "chaining variables", and $\kappa(i)$ chooses one of the elements $s_0, \ldots, s_d$.[13] Thus, the total size of a key in the resulting function is $\ell + (d+1)n$ bits.

Remark 3 (Two-round short commitments)**.** *If the short commitment protocol consists of two rounds (a message from the receiver to the sender and then a message back), then we actually get a collision resistant hash. The description of the hash is the message sent by the receiver and evaluation is done by simulating the (single) message sent by the sender.*

Proof of Theorem 5. Assume the existence of a short commitment protocol $(\mathcal{S}, \mathcal{R}, \mathcal{V})$ in which the receiver is public-coin and the sender is deterministic. The latter is without loss of generality due to the perfect completeness of the protocol. Assume that the protocol allows to commit on strings of length $\ell_{\mathsf{in}} = \ell_{\mathsf{in}}(n)$. Denote the length of the randomness used by the receiver by $\ell_{\mathsf{r}} = \ell_{\mathsf{r}}(n)$. Denote the length of the transcript by ℓ_{trns} (this is an upper bound on the maximum length of the transcript over all possible inputs and messages from the receiver). Assume that the opening of the sender opening is just sending the committed input x. The verifier then decides whether the opening is valid or not based on the input of the sender and on the (public) randomness of the verifier.

We construct a universal one-way hash function family $\mathcal{H} = \{\mathcal{H}_n\}_{n\in\mathbb{N}}$, where each function h in the family $\mathcal{H}_n$ gets as input strings of length ℓ_{in} and outputs a string of length ℓ_{trns}. This results with a compressing function since the length of a commitment is indeed shorter than the length of the input (i.e., $\ell_{\mathsf{in}} < \ell_{\mathsf{trns}}$). The description of a hash function $h \in \mathcal{H}_n$ is a string of length ℓ_{r} that contains the randomness used by the receiver. Evaluation of h on input $x \in \{0,1\}^{\ell_{\mathsf{in}}}$ is the

[12]The concrete constant in our scheme is roughly 6 but we did not try to optimize it further.

[13]The function $\kappa(i)$ counts the number of times 2 divides i, that is, for $i \geq 1$, $\kappa(i)$ is the largest integer κ such that 2^κ divides i.

transcript of an execution of the commitment protocol between the sender and the receiver (where the receiver uses randomness coming from the description of the hash function):

$$h(x) = \langle \mathcal{S}(1^n, x), \mathcal{R} \rangle.$$

We prove that this function family is a universal one-way hash function. In the latter, an adversary first submits a challenge x^*, then gets to see a description of a random hash function h from the family and finally needs to output (with noticeable probability) an $x \neq x^*$ such $h(x) = h(x^*)$. Let us assume that such an adversary $\mathcal{A}$ exists and it succeeds in finding such an x with noticeable probability $\epsilon > 0$. We design a sender $\mathcal{S}^*$ that breaks the computational-binding of the short commitment protocol.

The sender chooses x^* by simulating the choice of the latter by $\mathcal{A}$. Then, $\mathcal{S}^*$ simulates the execution of $\mathcal{S}$ when interacting with $\mathcal{R}$ and committing on the string x^*. Now, given the randomness of the receiver (here we use the fact that the receiver is public coin), we use it as a description of a hash function in $\mathcal{H}$ and simulate the execution of $\mathcal{A}$ with this function and output the pair of openings (x, x^*). With probability ϵ, the adversary $\mathcal{A}$ will output an $x \neq x^*$ for which $h(x) = h(x^*)$. In other words, $\mathcal{A}$ will output (with probability ϵ) an x with which the transcript of the commitment protocol would have been *the same*. Due to perfect completeness, the verifier of the commitment protocol must accept this value as a valid opening, as well. This is a contradiction to the computational-binding of the protocol.

$\square$

Remark 4 (UOWHF on short inputs)**.** *We can get a very efficient construction of a UOWHF family that operates on inputs of fixed length, say $2n$ and outputs $< 2n$ bits. For this we can use the construction of short commitments described in Section 2 which allows to commit to a string of length $2n$ using $< 2n$ bits. The construction is fairly efficient as it requires merely an evaluation of an almost-universal hash function (see Definition 3) in addition to the MCRH.*

Let $\mathcal{H} = \{\mathcal{H}_n \colon \{0,1\}^{2n} \to \{0,1\}^n\}_{n\in\mathbb{N}}$ be a k-MCRH for a polynomial $k = k(n)$. Assume that every family member of $\mathcal{H}$ is of length ℓ bits. Let $\mathcal{G} = \{g \colon \{0,1\}^{2n} \to \{0,1\}^{0.3n}\}$ be a δ-almost universal function family, where $\delta = 2^{-0.3n}$ and each member is described by $O(\log n)$ bits. We define a function family ensemble $\mathcal{F} = \{\mathcal{F}_n \colon \{0,1\}^{2n} \to \{0,1\}^{1.6n}\}_{n\in\mathbb{N}}$ as follows:

1. *Each member $f \in \mathcal{F}_n$ consists of a pair: an $h \in \mathcal{H}_n$ and a $g \in \mathcal{G}$.*
2. *For a member $f = (h, g) \in \mathcal{F}_n$, and an input $x \in \{0,1\}^{2n}$, we define $f(x) = h(x) \circ g(x)$.*

Using our technique of partial rewinding of the adversary to collect multiple collisions, we can show that if $\mathcal{H}$ is a k-MCRH for a polynomial $k = k(n)$, then $\mathcal{F}$ is a UOWHF, where each member of $\mathcal{F}_n$ can be described by a member h of $\mathcal{H}_n$ and additional $O(n)$ bits, and evaluation of a member of $\mathcal{F}_n$ consists of a single evaluation of a member in $\mathcal{H}$.

Acknowledgments

We are grateful to Noga Ron-Zewi for telling us about list-recoverable codes, for multiple useful discussions, and for sharing with us a preliminary version of [HRZW17]. We greatly acknowledge Gilad Asharov and Gil Segev for educating us about black-box separations. We thank Iftach Haitner and Eran Omri for answering questions related to [HIOS15].

References

[AGHP92] Noga Alon, Oded Goldreich, Johan Håstad, and René Peralta. Simple construction of almost k-wise independent random variables. *Random Struct. Algorithms*, 3(3):289–304, 1992.

[AS08] Noga Alon and Joel Spencer. *The Probabilistic Method*. John Wiley, third edition, 2008.

[AS16] Gilad Asharov and Gil Segev. Limits on the power of indistinguishability obfuscation and functional encryption. *SIAM J. Comput.*, 45(6):2117–2176, 2016.

[Bar01] Boaz Barak. How to go beyond the black-box simulation barrier. In *42nd Annual Symposium on Foundations of Computer Science, FOCS*, pages 106–115. IEEE Computer Society, 2001.

[BCC88] Gilles Brassard, David Chaum, and Claude Crépeau. Minimum disclosure proofs of knowledge. *J. Comput. Syst. Sci.*, 37(2):156–189, 1988.

[BDRV17] Itay Berman, Akshay Degwekar, Ron D. Rothblum, and Prashant Nalini Vasudevan. Multi collision resistant hash functions and their applications, 2017. Unpublished.

[BG08] Boaz Barak and Oded Goldreich. Universal arguments and their applications. *SIAM J. Comput.*, 38(5):1661–1694, 2008.

[BKP17] Nir Bitansky, Yael Tauman Kalai, and Omer Paneth. Multi-collision resistance: A paradigm for keyless hash functions, 2017. Unpublished.

[BR97] Mihir Bellare and Phillip Rogaway. Collision-resistant hashing: Towards making uowhfs practical. In *Advances in Cryptology - CRYPTO*, volume 1294, pages 470–484, 1997.

[Dam89] Ivan Damgård. A design principle for hash functions. In *Advances in Cryptology - CRYPTO*, volume 435, pages 416–427, 1989.

[DPP97] Ivan Damgård, Torben P. Pedersen, and Birgit Pfitzmann. On the existence of statistically hiding bit commitment schemes and fail-stop signatures. *J. Cryptology*, 10(3):163–194, 1997.

[DPP98] Ivan Damgård, Torben P. Pedersen, and Birgit Pfitzmann. Statistical secrecy and multibit commitments. *IEEE Trans. Information Theory*, 44(3):1143–1151, 1998.

[GGKT05] Rosario Gennaro, Yael Gertner, Jonathan Katz, and Luca Trevisan. Bounds on the efficiency of generic cryptographic constructions. *SIAM J. Comput.*, 35(1):217–246, 2005.

[GI02] Venkatesan Guruswami and Piotr Indyk. Near-optimal linear-time codes for unique decoding and new list-decodable codes over smaller alphabets. In *Proceedings on 34th Annual ACM Symposium on Theory of Computing*, pages 812–821. ACM, 2002.

[GI03] Venkatesan Guruswami and Piotr Indyk. Linear time encodable and list decodable codes. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing*, pages 126–135. ACM, 2003.

[GI04] Venkatesan Guruswami and Piotr Indyk. Linear-time list decoding in error-free settings: (extended abstract). In *Automata, Languages and Programming: 31st International Colloquium, ICALP*, volume 3142 of *Lecture Notes in Computer Science*, pages 695–707, 2004.

[GS99] Venkatesan Guruswami and Madhu Sudan. Improved decoding of reed-solomon and algebraic-geometry codes. *IEEE Trans. Information Theory*, 45(6):1757–1767, 1999.

[GUV09] Venkatesan Guruswami, Christopher Umans, and Salil P. Vadhan. Unbalanced expanders and randomness extractors from parvaresh-vardy codes. *J. ACM*, 56(4):20:1–20:34, 2009.

[HHK$^+$09] Iftach Haitner, Omer Horvitz, Jonathan Katz, Chiu-Yuen Koo, Ruggero Morselli, and Ronen Shaltiel. Reducing complexity assumptions for statistically-hiding commitment. *J. Cryptology*, 22(3):283–310, 2009.

[HHRS15] Iftach Haitner, Jonathan J. Hoch, Omer Reingold, and Gil Segev. Finding collisions in interactive protocols - tight lower bounds on the round and communication complexities of statistically hiding commitments. *SIAM J. Comput.*, 44(1):193–242, 2015.

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM J. Comput.*, 1999.

[HIOS15] Iftach Haitner, Yuval Ishai, Eran Omri, and Ronen Shaltiel. Parallel hashing via list recoverability. In *Advances in Cryptology - CRYPTO*, volume 9216 of *Lecture Notes in Computer Science*, pages 173–190. Springer, 2015.

[HNO$^+$09] Iftach Haitner, Minh-Huyen Nguyen, Shien Jin Ong, Omer Reingold, and Salil P. Vadhan. Statistically hiding commitments and statistical zero-knowledge arguments from any one-way function. *SIAM J. Comput.*, 39(3):1153–1218, 2009.

[HRZW17] Brett Hemenway, Noga Ron-Zewi, and Mary Wootters. Local list recovery of high-rate tensor codes & applications, 2017. Unpublished.

[HW15] Brett Hemenway and Mary Wootters. Linear-time list recovery of high-rate expander codes. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP*, volume 9134 of *Lecture Notes in Computer Science*, pages 701–712. Springer, 2015.

[IL89] Russell Impagliazzo and Michael Luby. One-way functions are essential for complexity based cryptography (extended abstract). In *30th Annual Symposium on Foundations of Computer Science, FOCS*, pages 230–235. IEEE Computer Society, 1989.

[ILL89] Russell Impagliazzo, Leonid A. Levin, and Michael Luby. Pseudo-random generation from one-way functions (extended abstracts). In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*, pages 12–24. ACM, 1989.

[Imp95] Russell Impagliazzo. A personal view of average-case complexity. In *Proceedings of the Tenth Annual Structure in Complexity Theory Conference*, pages 134–147. IEEE Computer Society, 1995.

[Jou04] Antoine Joux. Multicollisions in iterated hash functions. application to cascaded constructions. In *CRYPTO*, volume 3152, pages 306–316, 2004.

[JPY88] David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. How easy is local search? *J. Comput. Syst. Sci.*, 37(1):79–100, 1988.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *STOC*, pages 723–732. ACM, 1992.

[KK05] Jonathan Katz and Chiu-Yuen Koo. On constructing universal one-way hash functions from arbitrary one-way functions. *IACR Cryptology ePrint Archive*, 2005:328, 2005.

[KNY17] Ilan Komargodski, Moni Naor, and Eylon Yogev. White-box vs. black-box complexity of search problems: Ramsey and graph property testing. *Electronic Colloquium on Computational Complexity (ECCC)*, 24:15, 2017.

[Mer89a] Ralph C. Merkle. A certified digital signature. In *CRYPTO*, volume 435, pages 218–238. Springer, 1989.

[Mer89b] Ralph C. Merkle. One way hash functions and DES. In Gilles Brassard, editor, *Advances in Cryptology - CRYPTO*, volume 435, pages 428–446, 1989.

[Mir06] Ilya Mironov. Collision-resistant no more: Hash-and-sign paradigm revisited. In *Public Key Cryptography - PKC*, pages 140–156, 2006.

[MP91] Nimrod Megiddo and Christos H. Papadimitriou. On total functions, existence theorems and computational complexity. *Theor. Comput. Sci.*, 81(2):317–324, 1991.

[NN93] Joseph Naor and Moni Naor. Small-bias probability spaces: Efficient constructions and applications. *SIAM J. Comput.*, 22(4):838–856, 1993.

[NOVY98] Moni Naor, Rafail Ostrovsky, Ramarathnam Venkatesan, and Moti Yung. Perfect zero-knowledge arguments for *NP* using any one-way permutation. *J. Cryptology*, 11(2):87–108, 1998.

[NPR12] Hung Q. Ngo, Ely Porat, and Atri Rudra. Efficiently decodable compressed sensing by list-recoverable codes and recursion. In *29th International Symposium on Theoretical Aspects of Computer Science, STACS*, volume 14 of *LIPIcs*, pages 230–241. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.

[NY89] Moni Naor and Moti Yung. Universal one-way hash functions and their cryptographic applications. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*, pages 33–43. ACM, 1989.

[Pap94] Christos H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *J. Comput. Syst. Sci.*, 48(3):498–532, 1994.

[Rom90] John Rompel. One-way functions are necessary and sufficient for secure signatures. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*, pages 387–394. ACM, 1990.

[SBK+17] Marc Stevens, Elie Bursztein, Pierre Karpman, Ange Albertini, and Yarik Markov. The first collision for full SHA-1. *IACR Cryptology ePrint Archive*, 2017:190, 2017.

[Sho00] Victor Shoup. A composition theorem for universal one-way hash functions. In *Advances in Cryptology - EUROCRYPT*, volume 1807, pages 445–452, 2000.

[Sim98] Daniel R. Simon. Finding collisions on a one-way street: Can secure hash functions be based on general assumptions? In *EUROCRYPT*, volume 1403, pages 334–345. Springer, 1998.

[WC81] Mark N. Wegman and Larry Carter. New hash functions and their use in authentication and set equality. *J. Comput. Syst. Sci.*, 22(3):265–279, 1981.

[Wee07] Hoeteck Wee. One-way permutations, interactive hashing and statistically hiding commitments. In *Theory of Cryptography, 4th Theory of Cryptography Conference, TCC*, pages 419–433, 2007.

[WYY05] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding collisions in the full SHA-1. In *Advances in Cryptology - CRYPTO*, pages 17–36, 2005.

A Multi-Pair Collision Resistance

We have considered so far a weakening of the security definition of a standard CRH called *multi*-collision-resistance, where the challenge is to find a k-wise collision. In this section, we consider a *different* relaxation of collision resistance we call *multi-pair-collision-resistance*, where the challenge is to find *arbitrary* k distinct pairs of inputs that collide. More precisely, an efficient function family ensembles $\mathcal{H} = \{\mathcal{H}_n \colon \{0,1\}^{3n} \to \{0,1\}^n\}_{n\in\mathbb{N}}$ is a secure *k-multi-pair-collision-resistant hash* (MPCRH) if it is computationally hard to find $k = k(n)$ distinct pairs $(x_1, y_1), \ldots, (x_k, y_k)$ such that $h(x_1) = h(y_1), \ldots, h(x_k) = h(y_k)$.

It is immediate that a standard CRH is also an MPCRH, and we prove the other direction next. Let $\mathcal{H}$ be a secure k-MPCRH for a polynomial $k = k(n)$. Define the family

$$\mathcal{H}' = \{h' = (h, s)\}_{h\in\mathcal{H}, s\in\{0,1\}^{n/2}}$$

to be a family of function mapping $2n$ bits to n bits, where for $h' = (h, s)$ we define $h'(x) = h(s \circ x)$.

Lemma 3. *$\mathcal{H}'$ is a secure CRH.*

Proof. Assume that there exist an adversary $\mathcal{A}$ that can break the security of $\mathcal{H}'$. There is, There exits a polynomial $p(\cdot)$ such that

$$\Pr\left[\begin{array}{c} (x_1, y_1), \ldots, (x_k, y_k) \textit{ are distinct and} \\ h(x_1) = h(y_1), \ldots, h(x_k) = h(y_k) \end{array} \middle| \begin{array}{c} h \leftarrow \mathcal{H}_n \\ ((x_1, y_1), \ldots, (x_k, y_k)) \leftarrow \mathcal{A}(h) \end{array}\right] \geq 1/p(n).$$

We construct $\mathcal{A}'$ that breaks the security of the MPCRH. The algorithm $\mathcal{A}'$ acts as follows:
$\underline{\mathcal{A}'(h)}$:

1. Let $L = \emptyset$ be an empty list.

2. Repeat $T = (nkp(n))^2$ times:

(a) Sample $s \leftarrow \{0,1\}^n$ at random.

(b) Run $\mathcal{A}(h')$, where $h' = (s, h)$ to get a pair (x, y).

(c) If $h(s \circ x) = h(s \circ y)$, then add (x, y) to L.

3. Output L.

Since $\mathcal{A}$ succeeds with probability $\epsilon = 1/p(n)$, a standard Chernoff bound (see e.g., [AS08, §A.1]) gives that $\mathcal{A}'$ will collect k colliding pairs from $\mathcal{A}$ with all but exponentially small probability. Notice that if $s \neq s'$ then the collisions that $\mathcal{A}$ outputs on (h, s) must be distinct from the collisions that it outputs on (h, s'). Thus, $\mathcal{A}'$ will find k distinct pairs of collisions with high probability. □

B On TFNP and Multi-CRH

The class TFNP, defined by Megiddo and Papadimitriou [MP91], is the class of all search problems for which a solution is guaranteed to exist for every instance and verifying a solution can be done efficiently. TFNP has attracted extensive attention due to its important syntactic subclasses. The common way of defining such subclasses is via various non-constructive arguments used for proving totality of search problems. Some of the most known subclasses are PPAD (for Polynomial Parity Argument on Directed graphs [Pap94]), PPP (for Polynomial Pigeonhole Principle [Pap94]) and PLS (for Polynomial Local Search [JPY88]).

For many of the subclasses of TFNP we either can show containment of one class in the other, or an oracle separation ruling out the possibility of a black-box reduction from one to the other. Using our separation results from Section 7 we get an infinite hierarchy of subclass of TFNP, each one is contained in the next, but separated (relative to an oracle) from the previous one[14]. We define this hierarchy below.

The formal definition of TFNP is given by:

Definition 13 (TFNP)**.** *A* total NP search problem *is a relation $\mathcal{S}(x, y)$ such that it is (i) computable in polynomial (in $|x|$ and $|y|$) time (ii)* total*, i.e. there is a polynomial q such that for every x there exists a y such that $\mathcal{S}(x, y)$ and $|y| \leq q(|x|)$. The set of all total NP search problems is denoted by TFNP.*

The weak pigeon class (PWPP) is a subclass in TFNP defined by the collision finding problem for circuits that compress their input.

Definition 14 (PWPP^n_{n-1} complete problem)**.** *Given a circuit $C\colon \{0,1\}^n \to \{0,1\}^{n-1}$, find $x \neq y$ such that $C(x) = C(y)$. Moreover, we define $\mathsf{PWPP} = \mathsf{PWPP}^n_{n-1}$.*

Any hard distribution for PWPP^n_k naturally gives rise for a collision resistant hash function, and vice-versa. Using our notion of MCRH, we define a new class of total problems that we call Multi Pigeonhole Principle with a parameter k, or k-PMPP for short.

Definition 15 (k-PMPP complete problem)**.** *Given a circuit $C\colon \{0,1\}^n \to \{0,1\}^{n/2}$, find k distinct elements $x_1, \ldots, x_k$ such that $C(x_1) = \cdots = C(x_k)$.*

[14]We thank Bobby Kleinberg for asking us whether such an infinite hierarchy exists inside TFNP.

The first thing to notice is that k-PMPP is indeed a subclass of TFNP for any constant $k \in N$. Indeed, suppose that $C\colon \{0,1\}^n \to \{0,1\}^{n/2}$. If there is no k-wise collision then each output has at most $k-1$ inputs. Thus, the total number of element in the domain is at most $2^{n/2} \cdot (k-1) < 2^n$. This is of course a contradiction since the number of elements in the domain is exactly 2^n.

Notice that 2-PMPP $\subset$ 3-PMPP $\subset$ 4-PMPP .... Moreover, using the separation results from Section 7 we get that (under black-box reductions) all of these classes are separated. That is, 3-PMPP $\not\subseteq$ 2-PMPP, 4-PMPP $\not\subseteq$ 3-PMPP, 5-PMPP $\not\subseteq$ 4-PMPP and so on. This concludes the hierarchy.