Adversarially Robust Property-Preserving Hash Functions

Elette Boyle[1]
IDC Herzliya, Kanfei Nesharim Herzliya, Israel
elette.boyle@idc.ac.il

Rio LaVigne[2]
MIT CSAIL, 32 Vassar Street, Cambridge MA, 02139 USA
rio@mit.edu

Vinod Vaikuntanathan[3]
MIT CSAIL, 32 Vassar Street, Cambridge MA, 02139 USA
vinodv@mit.edu

Abstract

Property-preserving hashing is a method of compressing a large input $\mathbf{x}$ into a short hash $h(\mathbf{x})$ in such a way that given $h(\mathbf{x})$ and $h(\mathbf{y})$, one can compute a property $P(\mathbf{x}, \mathbf{y})$ of the original inputs. The idea of property-preserving hash functions underlies sketching, compressed sensing and locality-sensitive hashing.

Property-preserving hash functions are usually probabilistic: they use the random choice of a hash function from a family to achieve compression, and as a consequence, err on some inputs. Traditionally, the notion of correctness for these hash functions requires that for every two inputs $\mathbf{x}$ and $\mathbf{y}$, the probability that $h(\mathbf{x})$ and $h(\mathbf{y})$ mislead us into a wrong prediction of $P(\mathbf{x}, \mathbf{y})$ is negligible. As observed in many recent works (incl. Mironov, Naor and Segev, STOC 2008; Hardt and Woodruff, STOC 2013; Naor and Yogev, CRYPTO 2015), such a correctness guarantee assumes that the adversary (who produces the offending inputs) has no information about the hash function, and is too weak in many scenarios.

We initiate the study of *adversarial robustness* for property-preserving hash functions, provide definitions, derive broad lower bounds due to a simple connection with communication complexity, and show the necessity of computational assumptions to construct such functions. Our main positive results are two candidate constructions of property-preserving hash functions (achieving different parameters) for the (promise) gap-Hamming property which checks if $\mathbf{x}$ and $\mathbf{y}$ are "too far" or "too close". Our first construction relies on generic collision-resistant hash functions, and our second on a variant of the syndrome decoding assumption on low-density parity check codes.

2012 ACM Subject Classification Theory of computation → Cryptographic primitives

Keywords and phrases Hash function, compression, property-preserving, one-way communication

Digital Object Identifier 10.4230/LIPIcs.ITCS.2019.16

[1] Supported in part by ISF grant 1861/16, AFOSR Award FA9550-17-1-0069, and ERC grant 742754 (project NTSC).

[2] This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. 1122374. Any opinion, findings, and conclusions or recommendations expressed in this material are those of the authors(s) and do not necessarily reflect the views of the National Science Foundation. This research was also supported in part by grants of the third author.

[3] Research supported in part by NSF Grants CNS-1350619 and CNS-1414119, an MIT-IBM grant and a DARPA Young Faculty Award.

10th Innovations in Theoretical Computer Science (ITCS 2019).
Editor: Avrim Blum; Article No. 16; pp. 16:1–16:20
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Related Version A full version of the paper is available at `https://eprint.iacr.org/2018/1158.pdf`.

Acknowledgements We thank Daniel Wichs for suggesting the construction in Section 4 and for discussions regarding the connection to secure sketches and fuzzy extractors, and Shafi Goldwasser for enlightening conversations on the connections to robustness in machine learning.

1 Introduction

The problem of *property-preserving hashing*, namely how to compress a large input in a way that preserves a class of its properties, is an important one in the modern age of massive data. In particular, the idea of property-preserving hashing underlies sketching [23, 22, 1, 8, 6], compressed sensing [7], locality-sensitive hashing [14], and in a broad sense, much of machine learning.

As two concrete examples in theoretical computer science, consider universal hash functions [5] which can be used to test the equality of data points, and locality-sensitive hash functions [14, 13] which can be used to test the ℓ_p-distance between vectors. In both cases, we trade off accuracy in exchange for compression. For example, in the use of universal hash functions to test for equality of data points, one stores the hash $h(x)$ of a point x together with the description of the hash function h. Later, upon obtaining a point y, one computes $h(y)$ and checks if $h(y) = h(x)$. The pigeonhole principle tells us that mistakes are inevitable; all one can guarantee is that they happen with an acceptably small probability. More precisely, universal hash functions tell us that

$$\forall x \neq y \in D, \Pr[h \leftarrow \mathcal{H} : h(x) \neq h(y)] \geq 1 - \epsilon$$

for some small ϵ. A cryptographer's way of looking at such a statement is that it asks the adversary to pick x and y first; and evaluates her success w.r.t. a hash function chosen randomly from the family $\mathcal{H}$. In particular, the adversary has no information about the hash function when she comes up with the (potentially) offending inputs x and y. Locality-sensitive hash functions have a similar flavor of correctness guarantee.

The starting point of this work is that this definition of correctness is too weak in the face of adversaries with access to the hash function (either the description of the function itself or perhaps simply oracle access to its evaluation). Indeed, in the context of equality testing, we have by now developed several notions of robustness against such adversaries, in the form of pseudorandom functions (PRF) [11], universal one-way hash functions (UOWHF) [25] and collision-resistant hash functions (CRHF). Our goal in this work is to expand the reach of these notions beyond testing equality; that is, our aim is *to do unto property-preserving hashing what CRHFs did to universal hashing.*

Several works have observed the deficiency of the universal hash-type definition in adversarial settings, including a wide range of recent attacks within machine learning in adversarial environments (e.g., [20, 17, 28, 26, 16]). Such findings motivate a rigorous approach to combatting adversarial behavior in these settings, a direction in which significantly less progress has been made. Mironov, Naor and Segev [21] showed *interactive protocols* for sketching in such an adversarial environment; in contrast, we focus on non-interactive hash functions. Hardt and Woodruff [12] showed negative results which say that linear functions cannot be robust (even against computationally bounded adversaries) for certain natural ℓ_p distance properties; our work will use non-linearity and computational assumptions to overcome the [12] attack. Finally, Naor and Yogev [24] study adversarial Bloom filters which compress a set in a way that supports checking set membership; we were able to use their lower bound techniques (in the full version of this paper), proving the necessity for cryptographic assumptions for many predicates.

Motivating Robustness: Facial Recognition

In the context of facial recognition, authorities A and B store the captured images x of suspects. At various points in time, say authority A wishes to look up B's database for a suspect with face x. A can do so by comparing $h(x)$ with $h(y)$ for all y in B's database.

This application scenario motivated prior notions of fuzzy extractors and secure sketching. As with secure sketches and fuzzy extractors, a locality-sensitive property-preserving hash guarantees that close inputs (facial images) remain close when hashed [9]; this ensures that small changes in ones appearance do not affect whether or not that person is authenticated. However, neither fuzzy extractors nor secure sketching guarantees that *far* inputs remain far when hashed. Consider an adversarial setting, not where a person wishes to evade detection, but where she wishes to be mistaken for someone else. Her face x' will undoubtably be different (far) from her target x, but there is nothing preventing her from slightly altering her face and passing as a completely different person when using a system with such a one-sided guarantee. This is where our notion of robustness comes in (as well as the need for cryptography): not only will adversarially chosen close x and x' map to close $h(x)$ and $h(x')$, but if adversarially chosen x and x' are *far*, they will be mapped to far outputs, unless the adversary is able to break a cryptographic assumption.

Comparison to Secure Sketches and Fuzzy Extractors

It is worth explicitly comparing fuzzy extractors and secure sketching to this primitive [9], as they aim to achieve similar goals. Both of these seek to preserve the privacy of their inputs. Secure sketches generate random-looking sketches that hide information about the original input so that the original input can be reconstructed when given something close to it. Fuzzy extractors generate uniform-looking keys based off of fuzzy (biometric) data also using entropy: as long as the input has enough entropy, so will the output. As stated above, both guarantee that if inputs are close, they will 'sketch' or 'extract' to the same object. Now, the entropy of the sketch or key guarantees that randomly generated far inputs will not collide, but there are no guarantees about adversarially generated far inputs. To use the example above, it could be that once an adversary sees a sketch or representation, she can generate two far inputs that will reconstruct to the correct input.

Robust Property-Preserving Hash Functions

We put forth several notions of *robustness* for property-preserving hash (PPH) functions which capture adversaries with increasing power and access to the hash function. We then ask which properties admit robust property-preserving hash functions, and show positive and negative results.

- On the negative side, using a connection to communication complexity, we show that most properties and even simple ones such as set disjointness, inner product and greater-than do not admit non-trivial property-preserving hash functions.
- On the positive side, we provide two constructions of robust property-preserving hash functions (satisfying the strongest of our notions). The first is based on the standard cryptographic assumption of collision-resistant hash functions, and the second achieves

more aggressive parameters under a new assumption related to the hardness of syndrome decoding on low density parity-check (LDPC) codes. The first is expanded upon in this version (section 4), while the second is in the full version.

- Finally, in the full version, we show that for essentially any non-trivial predicate (which we call collision-sensitive), achieving even a mild form of robustness requires cryptographic assumptions.

We proceed to describe our contributions in more detail.

1.1 Our Results and Techniques

We explore two notions of properties. The first is that of property classes $\mathcal{P} = \{P : D \to \{0,1\}\}$, sets of single-input predicates. This notion is the most general, and is the one in which we prove lower bounds. The second is that of two-input properties $P : D \times D \to \{0,1\}$, which compares two inputs. This second notion is more similar to standard notions of universal hashing and collision-resistance, stronger than the first, and where we get our constructions. We note that a two-input predicate has an analogous predicate-class $\mathcal{P} = \{P_x\}_{x \in D}$, where $P_{x_1}(x_2) = P(x_1, x_2)$.

The notion of a property can be generalized in many ways, allowing for promise properties which output 0, 1 or $\star$ (a don't care symbol), and allowing for more than 2 inputs. The simplest notion of correctness for property-preserving hash functions requires that, analogously to universal hash functions,

$$\forall x, y \in D,\ \Pr[h \leftarrow \mathcal{H} : \mathcal{H}.\mathsf{Eval}(h, h(x), h(y)) \neq P(x,y)] = \mathrm{negl}(\lambda)$$

or for single-input predicate-classes

$$\forall x \in D \text{ and } P \in \mathcal{P},\ \Pr[h \leftarrow \mathcal{H} : \mathcal{H}.\mathsf{Eval}(h, h(x), P) \neq P(x)] = \mathrm{negl}(\lambda)$$

where λ is a security parameter. Notice in the single-input case, the "second" input can be seen as the predicate chosen from the class.

For the sake of simplicity in our overview, we will focus on two-input predicates.

Defining Robust Property-Preserving Hashing

We define several notions of *robustness* for PPH, each one stronger than the last. Here, we describe the strongest of all, called *direct-access PPH*.

In a direct-access PPH, the (polynomial-time) adversary is given the hash function and is asked to find a pair of bad inputs, namely $x, y \in D$ such that $\mathcal{H}.\mathsf{Eval}(h, h(x), h(y)) \neq P(x,y)$. That is, we require that

$$\forall \text{ p.p.t. } \mathcal{A}, \Pr[h \leftarrow \mathcal{H}; (x,y) \leftarrow \mathcal{A}(h) : \mathcal{H}.\mathsf{Eval}(h, h(x), h(y)) \neq P(x,y)] = \mathrm{negl}(\lambda),$$

where we use the notation $\Pr[A_1; \ldots; A_m : E]$ to denote the probability that event E occurs following an experiment defined by executing the sequence $A_1, \ldots, A_m$ in order. The direct-access definition is the analog of collision-resistant hashing for general properties.

Our other definitions vary by how much access the adversary is given to the hash function, and are motivated by different application scenarios. From the strong to weak, these include double-oracle PPH where the adversary is given access to a hash oracle and a hash evaluation oracle, and evaluation-oracle PPH where the adversary is given only a combined oracle. Definitions similar to double-oracle PPH have been proposed in the context of adversarial bloom filters [24], and ones similar to evaluation-oracle PPH have been proposed in the context of showing attacks against property-preserving hash functions [12]. For more details, we refer the reader to Section 2.

Connections to Communication Complexity and Negative Results

Property-preserving hash functions for a property P, even without robustness, imply communication-efficient protocols for P in several models. For example, any PPH for P implies a protocol for P in the simultaneous messages model of Babai, Gal, Kimmel and Lokam [3] wherein Alice and Bob share a common random string h, and hold inputs x and y respectively. Their goal is to send a single message to Charlie who should be able to compute $P(x, y)$ except with small error. Similarly, another formalization of PPH that we present, called PPH for single-input predicate classes (see Section 2) implies efficient protocols in the one-way communication model [31].

We use known lower bounds in these communication models to rule out PPHs for several interesting predicates (even without robustness). There are two major differences between the PPH setting and the communication setting, however: (a) in the PPH setting, we demand an error that is negligible (in a security parameter); and (b) we are happy with protocols that communicate $n - 1$ bits (or the equivalent bound in the case of promise properties) whereas the communication lower bounds typically come in the form of $\Omega(n)$ bits. In other words, the communication lower bounds *as-is* do not rule out PPH.

At first thought, one might be tempted to think that the negligible-error setting is the same as the deterministic setting where there are typically lower bounds of n (and not just $\Omega(n)$); however, this is not the case. For example, the equality function which has a negligible error public-coin simultaneous messages protocol (simply using universal hashing) with communication complexity $CC = O(\lambda)$ and deterministic protocols require $CC \geq n$. Thus, deterministic lower bounds do not (indeed, cannot) do the job, and we must better analyze the randomized lower bounds. Our refined analysis shows the following lower bounds:

- PPH for the Gap-Hamming (promise) predicate with a gap of $\sqrt{n}/2$ is impossible by refining the analysis of a proof by Jayram, Kumar and Sivakumar [15]. The Gap-Hamming predicate takes two vectors in $\{0,1\}^n$ as input, outputs 1 if the vectors are very far, 0 if they are very close, and we do not care what it outputs for inputs in the middle.
- We provide a framework for proving PPHs are impossible for some total predicates, characterizing these classes as *reconstructing*. A predicate-class is *reconstructing* if, when only given oracle access to the predicates of a certain value x, we can efficiently determine x with all but negligible probability.[4] With this framework, we show that PPH for the Greater-Than (GT) function is impossible. It was known that GT required $\Omega(n)$ bits (for constant error) [27], but we show a lower bound of exactly n if we want negligible error. Index and Exact-Hamming are also reconstructing predicates.
- We also obtain a lower bound for a variant of GT: the (promise) Gap-k GT predicate which on inputs $x, y \in [N = 2^n]$, outputs 1 if $x - y > k$, 0 if $y - x > k$, and we do not care what it outputs for inputs in between. Here, exactly $n - \log(k) - 1$ bits are required for a perfect PPH. This is tight: we show that with fewer bits, one cannot even have a non-robust PPH, whereas there is a perfect robust PPH that compresses to $n - \log(k) - 1$ bits.

New Constructions

Our positive results are two constructions of a direct-access PPH for gap-Hamming for n-length vectors for large gaps of the form $\sim O(n/\log n)$ (as opposed to an $O(\sqrt{n})$-gap for which we have a lower bound). Let us recall the setting: the gap Hamming predicate P_{ham},

[4] In the single-predicate language of above, the predicate class corresponds to $\mathcal{P} = \{P(x, \cdot)\}$.

parameterized by n, d and ϵ, takes as input two n-bit vectors x and y, and outputs 1 if the Hamming distance between x and y is greater than $d(1+\epsilon)$, 0 if it is smaller than $d(1-\epsilon)$ and a don't care symbol $\star$ otherwise. To construct a direct-access PPH for this (promise) predicate, one has to construct a compressing family of functions $\mathcal{H}$ such that

$$\forall \text{ p.p.t. } \mathcal{A}, \Pr[h \leftarrow \mathcal{H}; (x,y) \leftarrow \mathcal{A}(h) : P_{\mathsf{ham}}(x,y) \neq \star \\ \wedge \mathcal{H}.\mathsf{Eval}(h, h(x), h(y)) \neq P_{\mathsf{ham}}(x,y)] = \operatorname{negl}(\lambda). \quad (1)$$

Our two constructions offer different benefits. The first provides a clean general approach, and relies on the standard cryptographic assumption of collision-resistant hash functions. The second builds atop an existing one-way communication protocol, supports a smaller gap and better efficiency, and ultimately relies on a (new) variant of the syndrome decoding assumption on low-density parity check codes.

Construction 1. The core idea of the first construction is to reduce the goal of robust Hamming PPH to the simpler one of robust equality testing; or, in a word, "subsampling." The intuition is to notice that if $\mathbf{x}_1 \in \{0,1\}^n$ and $\mathbf{x}_2 \in \{0,1\}^n$ are *close*, then *most small enough subsets* of indices of $\mathbf{x}_1$ and $\mathbf{x}_2$ will match identically. On the other hand, if $\mathbf{x}_1$ and $\mathbf{x}_2$ are *far*, then most *large enough subsets* of indices will differ.

The hash function construction will thus fix a collection of sets $\mathcal{S} = \{S_1, \ldots, S_k\}$, where each $S_i \subseteq [n]$ is a subset of appropriately chosen size s. The desired structure can be achieved by defining the subsets S_i as the neighbor sets of a bipartite expander. On input $\mathbf{x} \in \{0,1\}^n$, the hash function will consider the vector $\mathbf{y} = (\mathbf{x}|_{S_1}, \ldots, \mathbf{x}|_{S_k})$ where $\mathbf{x}|_S$ denotes the substring of $\mathbf{x}$ indexed by the set S. The observation above tells us that if $\mathbf{x}_1$ and $\mathbf{x}_2$ are close (resp. far), then so are $\mathbf{y}_1$ and $\mathbf{y}_2$.

Up to now, it is not clear that progress has been made: indeed, the vector $\mathbf{y}$ is not compressing (in which case, why not stick with $\mathbf{x}_1, \mathbf{x}_2$ themselves?). However, $\mathbf{y}_1, \mathbf{y}_2$ satisfy the desired Hamming distance properties with fewer symbols over a *large alphabet*, $\{0,1\}^s$. As a final step, we can then leverage (standard) collision-resistant hash functions (CRHF) to compress these symbols. Namely, the final output of our hash function $h(\mathbf{x})$ will be the vector $(g(\mathbf{x}|_{S_1}), \ldots, g(\mathbf{x}|_{S_k}))$, where each substring of $\mathbf{x}$ is individually compressed by a CRHF g.

The analysis of the combined hash construction then follows cleanly via two steps. The (computational) collision-resistence property of g guarantees that any efficiently found pair of inputs $\mathbf{x}_1, \mathbf{x}_2$ will satisfy that their hash outputs

$$h(\mathbf{x}_1) = (g(\mathbf{x}_1|_{S_1}), \ldots, g(\mathbf{x}_1|_{S_k})) \quad \text{and} \quad h(\mathbf{x}_2) = (g(\mathbf{x}_2|_{S_1}), \ldots, g(\mathbf{x}_2|_{S_k}))$$

are close if and only if it holds that

$$(\mathbf{x}_1|_{S_1}, \ldots, \mathbf{x}_1|_{S_k}) \quad \text{and} \quad (\mathbf{x}_2|_{S_1}, \ldots, \mathbf{x}_2|_{S_k})$$

are close as well; that is, $\mathbf{x}_1|_{S_i} = \mathbf{x}_2|_{S_i}$ for most S_i. (Anything to the contrary would imply finding a collision in g.) Then, the combinatorial properties of the chosen index subsets S_i ensures (unconditionally) that any such inputs $\mathbf{x}_1, \mathbf{x}_2$ must themselves be close. The remainder of the work is to specify appropriate parameter regimes for which the CRHF can be used and the necessary bipartite expander graphs exist. Informally, we get the following theorem statement:

▶ **Theorem 1** (Collision-resistance-PPH informal). *Let λ be a security parameter. Assuming that CRHFs exist, for any polynomial $n = n(\lambda)$, and any constants $\epsilon, \eta, c > 0$, there is an η-compressing robust property preserving hash family for* GapHamming(n, d, ϵ) *where $d = o(n/\lambda^c)$.*

Construction 2. The starting point for our second construction is a simple non-robust hash function derived from a one-way communication protocol for gap-Hamming due to Kushilevitz, Ostrovsky, and Rabani [19]. In a nutshell, the hash function is parameterized by a random sparse $m \times n$ matrix A with 1's in $1/d$ of its entries and 0's elsewhere; multiplying this matrix by a vector $\mathbf{z}$ "captures" information about the Hamming weight of $\mathbf{z}$. However, this can be seen to be trivially *not robust* when the hash function is given to the adversary. The adversary simply performs Gaussian elimination, discovering a "random collision" (x, y) in the function, where, with high probability $x \oplus y$ will have large Hamming weight. This already breaks equation (1).

The situation is somewhat worse. Even in a very weak, oracle sense, corresponding to our evaluation-oracle-robustness definition, a result of Hardt and Woodruff [12] shows that there are no *linear functions* h that are robust for the gap-ℓ_2 predicate. While their result does not carry over *as-is* to the setting of ℓ_0 (Hamming), we conjecture it does, leaving us with two options: (a) make the domain sparse: both the Gaussian elimination attack and the Hardt-Woodruff attack use the fact that Gaussian elimination is easy on the domain of the hash function; however making the domain sparse (say, the set of all strings of weight at most βn for some constant $\beta < 1$) already rules it out; and (b) make the hash function non-linear: again, both attacks crucially exploit linearity. We will pursue both options, and as we will see, they are related.

But before we get there, let us ask whether we even need computational assumptions to get such a PPH. Can there be information-theoretic constructions? The first observation is that by a packing argument, if the output domain of the hash function has size less than $2^{n-n\cdot H(\frac{d(1+\epsilon)}{n})} \approx 2^{n-d\log n(1+\epsilon)}$ (for small d), there are bound to be "collisions", namely, two far points (at distance more than $d(1+\epsilon)$) that hash to the same point. So, you really cannot compress much information-theoretically, especially as d becomes smaller. A similar bound holds when restricting the domain to strings of Hamming weight at most βn for constant $\beta < 1$.

With that bit of information, let us proceed to describe in a very high level our construction and the computational assumption. Our construction follows the line of thinking of Applebaum, Haramaty, Ishai, Kushilevitz and Vaikuntanathan [2] where they used the hardness of syndrome decoding problems to construct collision-resistant hash functions. Indeed, in a single sentence, our observation is that their collision-resistant hash functions are *locality-sensitive* by virtue of being *input-local*, and thus give us a robust gap-Hamming PPH (albeit under a different assumption).

In slightly more detail, our first step is to simply take the construction of Kushilevitz, Ostrovsky, and Rabani [19], and restrict the domain of the function. We show that finding two close points that get mapped to far points under the hash function is simply impossible (for our setting of parameters). On the other hand, there exist two far points that get mapped to close points under the hash functions (in fact, they even collide). Thus, showing that it is hard to find such points requires a computational assumption.

In a nutshell, our assumption says that given a random matrix $\mathbf{A}$ where each entry is chosen from the Bernoulli distribution with $\mathsf{Ber}(1/d)$ with parameter $1/d$, it is hard to find a large Hamming weight vector $\mathbf{x}$ where $\mathbf{Ax}$ (mod 2) has small Hamming weight. Of course, "large" and "small" here have to be parameterized correctly (see the full version for more details), however we observe that this is a generalization of the syndrome decoding assumption for low-density parity check (LDPC) codes, made by [2].

In our second step, we remove the sparsity requirement on the input domain of the predicate. We show a sparsification transformation which takes arbitrary n-bit vectors and outputs $n' > n$-bit *sparse* vectors such that (a) the transformation is injective, and (b) the expansion introduced here does not cancel out the effect of compression achieved by the linear transformation $\mathbf{x} \to \mathbf{A}\mathbf{x}$. This requires careful tuning of parameters for which we refer the reader to the full version of this paper. Informally, our theorem statement is

▶ **Theorem 2** (Shortest-Vector PPH informal). *Let λ be a security parameter. Assuming the shortest-vector assumption (discussed above) with reasonable parameter settings, for any polynomial $n = n(\lambda)$, and any constants $\epsilon, \eta > 0$, there is an η-compressing robust property preserving hash family for* $\textsc{GapHamming}(n, d, \epsilon)$ *where* $d \leq \frac{n}{2\log n}((1-\epsilon) + (1+\epsilon))$.

Notice that we get a better parameter d than from our first construction. This is, of course, because we make a stronger assumption. For more details, we refer the reader to the full version of this paper.

The Necessity of Cryptographic Assumptions

The goal of robust PPH is to compress beyond the information theoretic limits, to a regime where incorrect hash outputs exist but are hard to find. If robustness is required even when the hash function is given, this inherently necessitates cryptographic hardness assumptions. A natural question is whether weaker forms of robustness (where the adversary sees only oracle access to the hash function) similarly require cryptographic assumptions, and what types of assumptions are required to build non-trivial PPHs of various kinds.

As a final contribution, we identify necessary assumptions for PPH for a kind of predicate we call *collision sensitive*. In particular, PPH for any such predicate in the double-oracle model implies the existence of one-way functions, and in the direct-access model implies existence of collision-resistant hash functions. In a nutshell, collision-sensitive means that finding a collision in the predicate breaks the property-preserving nature of any hash. The proof uses and expands on techniques from the work of Naor and Yogev on adversarially robust Bloom Filters [24]. The basic idea is the same: without OWFs, we can invert arbitrary polynomially-computable functions with high probability in polynomial time, and using this we get a representation of the hash function/set, which can be used to find offending inputs.

2 Defining Property-Preserving Hash Functions

Our definition of property-preserving hash functions (PPHs) comes in several flavors, depending on whether we support total or partial predicates; whether the predicates take a single input or multiple inputs; and depending on the information available to the adversary. We discuss each of these choices in turn.

Total vs. Partial Predicates

We consider *total predicates* that assign a 0 or 1 output to each element in the domain, and *promise (or partial) predicates* that assign a 0 or 1 to a subset of the domain and a wildcard (don't-care) symbol $\star$ to the rest. More formally, a *total predicate* P on a domain X is a function $P : X \to \{0, 1\}$, well-defined as 0 or 1 for every input $x \in X$. A *promise predicate* P on a domain X is a function $P : X \to \{0, 1, \star\}$. Promise predicates can be used to describe scenarios (such as gap problems) where we only care about providing an exact answer on a subset of the domain.

Our definitions below will deal with the more general case of promise predicates, but we will discuss the distinction between the two notions when warranted.

Single-Input vs Multi-Input Predicates

In the case of single-input predicates, we consider a class of properties $\mathcal{P}$ and hash a single input x into $h(x)$ in a way that given $h(x)$, one can compute $P(x)$ for any $P \in \mathcal{P}$. Here, h is a compressing function. In the multi-input setting, we think of a single fixed property P that acts on a tuple of inputs, and require that given $h(x_1), h(x_2), \ldots, h(x_k)$, one can compute $P(x_1, x_2, \ldots, x_k)$. The second syntax is more expressive than the first, and so we use the multi-input syntax for constructions and the single-input syntax for lower bounds[5].

Before we proceed to discuss robustness, we provide a working definition for a property-preserving hash function for the single-input syntax. For the multi-input predicate definition and further discussion, see the full version.

▶ **Definition 3.** A (non-robust) η-compressing Property Preserving Hash (η-PPH) family $\mathcal{H} = \{h : X \to Y\}$ for a function η and a class of predicates $\mathcal{P}$ requires the following two efficiently computable algorithms:

- $\mathcal{H}.\mathsf{Samp}(1^\lambda) \to h$ is a randomized p.p.t. algorithm that samples a random hash function from $\mathcal{H}$ with security parameter λ.
- $\mathcal{H}.\mathsf{Eval}(h, P, y)$ is a deterministic polynomial-time algorithm that on input the hash function h, a predicate $P \in \mathcal{P}$ and $y \in Y$ (presumably $h(x)$ for some $x \in X$), outputs a single bit.

Additionally, $\mathcal{H}$ must satisfy the following two properties:

- *η-compressing*, namely, $\log |Y| \leq \eta(\log |X|)$, and
- *robust*, according to one of four definitions that we describe below, leading to four notions of PPH: definition 4 (non-robust PPH), 5 (evaluation-oracle-robust PPH or EO-PPH), 7 (double-oracle-robust PPH or DO-PPH), or 9 (direct-access robust PPH or DA-PPH). We will refer to the strongest form, namely direct-access robust PPH as simply robust PPH when the intent is clear from the context. See also Table 1 for a direct comparison between these.

The Many Types of Robustness

We will next describe four definitions of robustness for PPHs, starting from the weakest to the strongest. Each of these definitions, when plugged into the last bullet of Definition 3, gives rise to a different type of property-preserving hash function. In each of these definitions, we will describe an adversary whose goal is to produce an input and a predicate such that the hashed predicate evaluation disagrees with the truth. The difference between the definitions is in what an adversary has access to, summarized in Table 1.

2.1 Non-Robust PPH

We will start by defining the weakest notion of robustness which we call non-robust PPH. Here, the adversary has no information at all on the hash function h, and is required to produce a predicate P and a valid input x, namely where $P(x) \neq \star$, such that $\mathcal{H}.\mathsf{Eval}(h, P, x) \neq P(x)$ with non-negligible probability. When $\mathcal{P}$ is the family of point functions (or equality functions), this coincides with the notion of 2-universal hash families [5][6].

[5] There is yet a third possibility, namely where there is a *fixed* predicate P that acts on a single input x, and we require that given $h(x)$, one can compute $P(x)$. This makes sense when the computational complexity of h is considerably less than that of P, say when P is the parity function and h is an AC^0 circuit, as in the work of Dubrov and Ishai [10]. We do not explore this third syntax further in this work.

[6] While 2-universal hashing corresponds with a two-input predicate testing equality, the single-input version ($\{P_{x_1}\}$ where $P_{x_1}(x_2) = (x_1 == x_2)$) is more general, and so it is what we focus on.

Table 1 A table comparing the adversary's access to the hash function within different robustness levels of PPHs.

For security parameter λ, fixed predicate class $\mathcal{P}$, and h sampled from $\mathcal{H}.\mathsf{Samp}$

Non-Robust PPH	Adversary has no access to hash function or evaluation.
Evaluation-Oracle PPH	Access to the evaluation oracle $\mathcal{O}_h^{\mathsf{Eval}}(x, P) = \mathcal{H}.\mathsf{Eval}(h, P, h(x))$.
Double-Oracle PPH	Access to both $\mathcal{O}_h^{\mathsf{Eval}}$ (as above) and hash oracle $\mathcal{O}_h^{\mathsf{Hash}}(x) = h(x)$.
Robust PPH “Direct Access”	Direct access to the hash function, description of h.

Here and in the following, we use the notation $\Pr[A_1; \ldots ; A_m \ : \ E]$ to denote the probability that event E occurs following an experiment defined by executing the sequence $A_1, \ldots, A_m$ in order.

▶ **Definition 4.** A family of PPH functions $\mathcal{H} = \{h : X \to Y\}$ for a class of predicates $\mathcal{P}$ is a family of *non-robust* PPH functions if for any $P \in \mathcal{P}$ and $x \in X$ such that for $P(x) \neq \star$,

$$\Pr[h \leftarrow \mathcal{H}.\mathsf{Samp}(1^\lambda) : \ \mathcal{H}.\mathsf{Eval}(h, P, h(x))) \neq P(x)] \leq \operatorname{negl}(\lambda).$$

2.2 Evaluation-Oracle Robust PPH

In this model, the adversary has slightly more power than in the non-robust setting. Namely, she can adaptively query an oracle that has $h \leftarrow \mathcal{H}.\mathsf{Samp}(1^\lambda)$ in its head, on inputs $P \in \mathcal{P}$ and $x \in X$, and obtain as output the hashed evaluation result $\mathcal{H}.\mathsf{Eval}(h, P, h(x))$. Let $\mathcal{O}_h(x, P) = \mathcal{H}.\mathsf{Eval}(h, P, h(x))$.

▶ **Definition 5.** A family of PPH functions $\mathcal{H} = \{h : X \to Y\}$ for a class of predicates $\mathcal{P}$ is a family of *evaluation-oracle robust (EO-robust)* PPH functions if, for any PPT adversary $\mathcal{A}$,

$$\begin{aligned}&\Pr[h \leftarrow \mathcal{H}.\mathsf{Samp}(1^\lambda); \ (x, P) \leftarrow \mathcal{A}^{\mathcal{O}_h}(1^\lambda) : \\ &\quad P(x) \neq \star \wedge \mathcal{H}.\mathsf{Eval}(h, P, h(x)) \neq P(x)] \leq \operatorname{negl}(\lambda).\end{aligned}$$

The reader might wonder if this definition is very weak, and may ask if it follows just from the definition of a non-robust PPH family. In fact, for *total predicates*, we show that the two definitions are the same. At a high level, simply querying the evaluation oracle on (even adaptively chosen) inputs cannot reveal information about the hash function since with all but negligible probability, the answer from the oracle will be correct and thus simulatable without oracle access. The proof of the following lemma is in the full version.

▶ **Lemma 6.** *Let $\mathcal{P}$ be a class of total predicates on X. A non-robust PPH $\mathcal{H}$ for $\mathcal{P}$ is also an Evaluation-Oracle robust PPH for $\mathcal{P}$ for the same domain X and same codomain Y.*

However, when dealing with *promise* predicates, an EO-robustness adversary has the ability to make queries that do not satisfy the promise, and could get information about the hash function, perhaps even reverse-engineering the entire hash function itself. Indeed, Hardt and Woodruff [12] show that there are no EO-robust *linear* hash functions for a certain promise-ℓ_p distance property; whereas, non-robust linear hash functions for these properties follow from the work of Indyk [14, 13].

2.3 Double-Oracle PPH

We continue our line of thought, giving the adversary more power. Namely, she has access to two oracles, both have a hash function $h \leftarrow \mathcal{H}.\mathsf{Samp}(1^\lambda)$ in their head. The hash oracle $\mathcal{O}_h^{\mathsf{Hash}}$, parameterized by $h \in \mathcal{H}$, outputs $h(x)$ on input $x \in X$. The predicate evaluation oracle $\mathcal{O}_h^{\mathsf{Eval}}$, also parameterized by $h \in \mathcal{H}$, takes as input $P \in \mathcal{P}$ and $y \in Y$ and outputs $\mathcal{H}.\mathsf{Eval}(h, P, y)$. When $\mathcal{P}$ is the family of point functions (or equality functions), this coincides with the notion of psuedo-random functions.

▶ **Definition 7.** A family of PPH functions $\mathcal{H} = \{h : X \to Y\}$ for a class of predicates $\mathcal{P}$ is a family of *double-oracle-robust PPH (DO-PPH)* functions if, for any PPT adversary $\mathcal{A}$,

$$\Pr[h \leftarrow \mathcal{H}.\mathsf{Samp}(1^\lambda);\ (x, P) \leftarrow \mathcal{A}^{\mathcal{O}_h^{\mathsf{Hash}}, \mathcal{O}_h^{\mathsf{Eval}}}(1^\lambda) :$$
$$P(x) \neq \star \wedge \mathcal{H}.\mathsf{Eval}(h, P, h(x)) \neq P(x)] \leq \mathrm{negl}(\lambda).$$

We show that any evaluation-oracle-robust PPH can be converted into a double-oracle-robust PPH at the cost of a computational assumption, namely, one-way functions. In a nutshell, the observation is that the output of the hash function can be encrypted using a symmetric key that is stored as part of the hash description, and the evaluation proceeds by first decrypting. The proof of the following lemma is in the full version.

▶ **Lemma 8.** *Let $\mathcal{P}$ be a class of (total or partial) predicates on X. Assume that one-way functions exist. Then, any EO-robust PPH for $\mathcal{P}$ can be converted into a DO-robust PPH for $\mathcal{P}$.*

2.4 Direct-Access Robust PPH

Finally, we define the strongest notion of robustness where the adversary is given the description of the hash function itself. When $\mathcal{P}$ is the family of point functions (or equality functions), this coincides with the notion of collision-resistant hash families.

▶ **Definition 9.** A family of PPH functions $\mathcal{H} = \{h : X \to Y\}$ for a class of predicates $\mathcal{P}$ is a family of *direct-access robust* PPH functions if, for any PPT adversary $\mathcal{A}$,

$$\Pr[h \leftarrow \mathcal{H}.\mathsf{Samp}(1^\lambda);\ (x, P) \leftarrow \mathcal{A}(h) :$$
$$P(x) \neq \star \wedge \mathcal{H}.\mathsf{Eval}(h, P, h(x))) \neq P(x)] \leq \mathrm{negl}(\lambda).$$

We will henceforth focus on *direct-access-robust* property-preserving hash functions and refer to them simply as robust PPHs.

3 Property-Preserving Hashing and Communication Complexity

In this section, we identify and examine a relationship between property-preserving hash families (in the single-input syntax) and protocols in the one-way communication (OWC) model. A OWC protocol is a protocol between two players, Alice and Bob, with the goal of evaluating a certain predicate on their inputs and with the restriction that only Alice can send messages to Bob.

Our first observation is that non-robust property-preserving hash functions and OWC protocols [31] are equivalent except for two changes. First, PPHs require the parties to be computationally efficient, and second, PPHs also require protocols that incur error negligible in a security parameter. It is also worth noting that while we can reference lower-bounds in the OWC setting, these lower bounds are typically of the form $\Omega(n)$ and are not exact. On the other hand, in the PPH setting, we are happy with getting a single bit of compression, and so an $\Omega(n)$ lower bound still does not tell us whether or not a PPH is possible. So, while we can use previously known lower bounds for some well-studied OWC predicates, we need to refine them to be exactly n in the presence of negligible error. We also propose a framework (for total predicates) that yields exactly n lower bounds for $\textsc{Index}_n$, GreaterThan,and ExactHamming.

3.1 PPH Lower Bounds from One-Way Communication Lower Bounds

In this section, we will review the definition of OWC, and show how OWC lower bounds imply PPH impossibility results.

▶ **Definition 10.** [31, 18] A δ-error public-coin OWC protocol Π for a two-input predicate $P : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ consists of a space R of randomness, and two functions $g_a : X_1 \times R \to Y$ and $g_b : Y \times X_2 \times R \to \{0,1\}$ so that for all $x_1 \in X_1$ and $x_2 \in X_2$,

$$\Pr[r \leftarrow R; y = g_a(x_1; r) : \; g_b(y, x_2; r) \neq P(x_1, x_2)] \leq \delta.$$

A δ-error public-coin OWC protocol Π for a *class of predicates* $\mathcal{P} = \{P : \{0,1\}^n \to \{0,1\}\}$, is defined much the same as above, with a function $g_a : X \times R \to Y$, and another function $g_b : Y \times \mathcal{P} \times R \to \{0,1\}$, which instead of taking a second input, takes a predicate from the predicate class. We say Π has δ-error if

$$\Pr[r \leftarrow R; y = g_a(x; r) : g_b(y, P; r) \neq P(x)] \leq \delta$$

Let $\mathsf{Protocols}_\delta(P)$ denote the set of OWC protocols with error at most δ for a predicate P, and for every $\Pi \in \mathsf{Protocols}_\delta(P)$, let Y_Π be the range of messages Alice sends to Bob (the range of g_a) for protocol Π.

▶ **Definition 11.** The randomized, public-coin *OWC complexity* of a predicate P with error δ, denoted $R_\delta^{A\to B}(P)$, is the minimum over all $\Pi \in \mathsf{Protocols}_\delta(P)$ of $\lceil \log |Y_\Pi| \rceil$.

For a predicate class $\mathcal{P}$, we define the randomized, public-coin OWC complexity with error δ, denoted $R_\delta^{A\to B}(\mathcal{P})$, is the minimum over all $\Pi \in \mathsf{Protocols}_\delta(\mathcal{P})$ of $\lceil \log |Y_\Pi| \rceil$.

A PPH scheme for a two-input predicate[7] P yields a OWC protocol for P with communication comparable to a single hash output size.

▶ **Theorem 12.** *Let P be any two-input predicate P and $\mathcal{P} = \{P_x\}_{x\in\{0,1\}^n}$ be the corresponding predicate class where $P_{x_2}(x_1) = P(x_1, x_2)$. Now, let $\mathcal{H}$ be a PPH in any model for $\mathcal{P}$ that compresses n bits to $m = \eta n$. Then, there exists a OWC protocol Π such that the communication of Π is m and with negligible error.*

Conversely, the amount of possible compression of any (robust or not) PPH family $\mathcal{H} : \{h : X \to Y\}$ is lower bounded by $R_{\mathrm{negl}(\lambda)}^{A\to B}(P)$. Namely, $\log |Y| \geq R_{\mathrm{negl}(\lambda)}^{A\to B}(\mathcal{P})$.

Essentially, the OWC protocol is obtained by using the public common randomness r to sample a hash function $h = \mathcal{H}.\mathsf{Samp}(1^\lambda; r)$, and then Alice simply sends the hash $h(x_1)$ of her input to Bob. (Proof in full version.)

[7] Or rather, for the induced class of single-input predicates $\mathcal{P} = \{P_{x_2}\}_{x_2\in\{0,1\}^n}$, where $P_{x_2}(x_1) = P(x_1, x_2)$; we will use these terminologies interchangeably.

3.2 OWC and PPH lower bounds for Reconstructing Predicates

We next leverage this connection together with OWC lower bounds to obtain impossibility results for PPHs. First, we will discuss the total predicate case; we consider some partial predicates in section 3.3.

As discussed, to demonstrate the impossibility of a PPH, one must give an explicit n-bit communication complexity lower bound (not just $\Omega(n)$) for negligible error. We give such lower bounds for an assortment of predicate classes by a general approach framework we refer to as *reconstructing*. Intuitively, a predicate class is reconstructing if, when given only access to predicates evaluated on an input x, one can, in polynomial time, determine the exact value of x with all but negligible probability.

▶ **Definition 13.** A class $\mathcal{P}$ of total predicates $P : \{0,1\}^n \to \{0,1\}$, is *reconstructing* if there exists a PPT algorithm L (a 'learner') such that for all $x \in \{0,1\}^n$, given randomness r and oracle access to predicates $\mathcal{P}$ on x, denoted $\mathcal{O}_x(P) = P(x)$,

$$\Pr_r[L^{\mathcal{O}_x}(r) \to x] \geq 1 - \mathrm{negl}(n).$$

▶ **Theorem 14.** *If $\mathcal{P}$ is a reconstructing class of predicates on input space $\{0,1\}^n$, then a PPH does not exist for $\mathcal{P}$.*

The full proof appears in the full version. The main idea is to simulate the learner L on two different inputs x_1 and x_2: at some query, L must get a different answer from the oracle, differentiating x_1 from x_2. Since h is compressing, there are many x_1 and x_2 that collide on the same output. We show that we can guess such an x_1 that is paired with an x_2, and the query that they differ on with $1/\mathrm{poly}(n)$ probability. Once we do that, the oracle must answer incorrectly for one of x_1 or x_2, and there is a half chance that we chose the x_i that evaluated incorrectly.

Reconstructing using Index$_n$, GreaterThan, or ExactHamming

We turn to specific examples of predicate classes and sketch why they are reconstructing. For formal proofs, we refer the reader to the full version of this paper.

- The INDEX$_n$ class of predicates $\{P_1, \ldots, P_n\}$ is defined over $x \in \{0,1\}^n$ where $P_i(x) = x_i$, the i'th bit of x. INDEX$_n$ is reconstructing simply because the learner L can just query the each of the n indices of the input and exactly reconstruct: $x_i = P_i(x)$.
- The GREATERTHAN class of predicates $\{P_x\}_{x\in[2^n]}$ is defined over $x \in [2^n] = \{0,1\}^n$ where $P_{x_2}(x_1) = 1$ if $x_1 > x_2$ and 0 otherwise. GREATERTHAN is reconstructing because we can run a binary search on the input space, determining the exact value of x in n queries. GREATERTHAN is an excellent example for how an adaptive learner L can reconstruct.
- The EXACTHAMMING(α) class of predicates $\{P_x\}_{x\in\{0,1\}^n}$ is defined over $x \in \{0,1\}^n$ where $P_{x_2}(x_1) = 1$ if $||x_1 - x_2||_0 > \alpha$ and 0 otherwise. To show that EXACTHAMMING$(n/2)$ is reconstructing requires a little more work. The learner L resolves each index of x independently. For each index, L makes polynomially many random-string queries $\mathbf{r}$ to $\mathcal{O}_x$; if the i'th bit of $\mathbf{r}$ equals x_i, then $\mathbf{r}$ is more likely to be within $n/2$ hamming distance of $\mathbf{x}$, and if the bits are different, $\mathbf{r}$ is more likely to not be within $n/2$ hamming distance of $\mathbf{x}$. The proof uses techniques from [15], and is an example where the learner uses randomness to reconstruct.

We note that it was already known that $\textsc{Index}_n$ and $\textsc{ExactHamming}(n/2)$ had OWC complexity of n-bits for any negligible error [18], though no precise lower bound for randomized OWC protocols was known for $\textsc{GreaterThan}$. What is new here is our unified framework.

3.3 Lower bounds for some partial predicates

In the previous section, we showed how the ability to reconstruct an input using a class of total predicates implied that PPHs for the class cannot exist. This general framework, unfortunately, does not directly extend to the partial-predicate setting, since it is unclear how to define the behavior of an oracle for the predicate. Nevertheless, we can still take existing OWC lower bounds and their techniques to prove impossibility results in this case. We will show that $\textsc{GapHamming}(n, n/2, 1/\sqrt{n})$ (the promise version of $\textsc{ExactHamming}$) cannot admit a PPH, and that while Gap-k $\textsc{GreaterThan}$ has a perfectly correct PPH compressing to $n - \log(k) - 1$ bits, compressing any further results in polynomial error (and thus no PPH with more compression).

First, we define these partial predicates.

▶ **Definition 15.** The definitions for $\textsc{GapHamming}(n, d, \epsilon)$ and Gap-k $\textsc{GreaterThan}$ are:

- The $\textsc{GapHamming}(n, d, \epsilon)$ class of predicates $\{P_x\}_{x\in\{0,1\}^n}$ has $P_{x_2}(x_1) = 1$ if $||x_1 - x_2||_0 \geq d(1+\epsilon)$, 0 if $||x_1 - x_2||_0 \leq d(1-\epsilon)$, and $\star$ otherwise.
- The Gap-k $\textsc{GreaterThan}$ class of predicates $\{P_x\}_{x\in[2^n]}$ has $P_{x_2}(x_1) = 1$ if $x_1 > x_2 + k$, 0 if $x_1 < x_2 - k$, and $\star$ otherwise.

Now, we provide some intuition for why these lower bounds (and the upper bound) exist.

Gap-Hamming

Our lower bound will correspond to a refined OWC lower bound for the Gap-Hamming problem in the relevant parameter regime. Because we want to prove that we cannot even compress by a single bit, we need to be careful with our reduction: we want the specific parameters for which we have a lower bound, and must keep close track of how the error changes within our reduction.

▶ **Theorem 16.** *There does not exist a PPH for* $\textsc{GapHamming}(n, n/2, 1/\sqrt{n})$.

To prove, we show the OWC complexity $R^{A\to B}_{\text{negl}(n)}$ $(\textsc{GapHamming}\ (n, n/2, 1/\sqrt{n})) = n$. A $\Omega(n)$ OWC lower bound for Gap-Hamming in this regime has been proved in a few different ways [29, 30, 15]. Our proof will be a refinement of [15] and is detailed in the full version.

The high-level structure of the proof is to reduce $\textsc{Index}_n$ to $\textsc{GapHamming}$ with the correct parameters. Very roughly, the ith coordinate of an input $x \in \{0,1\}^n$ can be inferred from the bias it induces on the Hamming distance between x and random public vectors. The reduction adds negligible error, but since we require n bits for negligible-error $\textsc{Index}_n$, we also require n bits for a OWC protocol for $\textsc{GapHamming}$.

Notice that this style of proof looks morally as though we are "reconstructing" the input x using $\textsc{Index}_n$. However, the notion of getting a reduction from $\textsc{Index}_n$ to another predicate-class in the OWC model is not the same as being able to query an oracle about the predicate and reconstruct based off of oracle queries. Being able to make a similar reconstructing characterization of partial-predicates as we have for total predicates would be useful and interesting in proving further lower bounds.

Gap-k GreaterThan

This predicate is a natural extension of GreaterThan: we only care about learning that $x_1 < x_2$ if $|x_1 - x_2|$ is larger than k (the gap). Intuitively, a hash function can maintain this information by simply removing the $\log(k)$ least significant bits from inputs and directly comparing: if $h(x_1) = h(x_2)$, they can be at most k apart. We can further remove one additional bit using the fact that we know x_2 when given $h(x_1)$ (considering Gap-k GreaterThan as the corresponding predicate class parameterized by x_2).

For the lower bound, we prove a OWC lower bound, showing $R^{A\to B}_{\mathrm{negl}(n)}(\mathcal{P}) = n - \log(k) - 1$. This will be a proof by contradiction: if we compress to $n - \log(k) - 2$ bits, we obtain many collisions that are more than $3.5k$ apart. These far collisions imply the existence of inputs that the OWC protocol must fail on, even given the gap. We are able to find these inputs the OWC must fail on with polynomial probability, and this breaks the all-but-negligible correctness of the protocol. Our formal theorem statement is below.

▶ **Theorem 17.** *There exists a PPH with perfect correctness for Gap-k* GreaterThan *compressing from n bits to $n - \log(k) - 1$. This is tight: no PPH for Gap-k* GreaterThan *can compress to fewer than $n - \log(k) - 1$ bits.*

For the proof, see the full version. To understand the construction of the PPH, first consider a hash function that just chops off the least-significant $\log(k)$ bits from the inputs. Clearly, comparing two hashed values for which is greater than the other gives the gap-k-GreaterThan result correctly. In order to get the last bit off, we chop off the least-significant $\log(k)$ bits and then round up or down depending on the $\log(k) + 1$'th significant bit before removing it. Evaluation is almost essentially comparing hashes, but we exploit the fact that we know the entire second input (x_2), which is why we are able to remove one extra bit. Proving the lower bound is a bit trickier, but essentially involves simulating a binary search.

4 A Gap-Hamming PPH from Collision Resistance

In this section, we present one of our constructions for a PPH for the GapHamming problem. Recall from section 3.3 that the gap-Hamming property $P = \textsc{GapHamming}(n, d, \epsilon)$ is parameterized by the input domain $\{0,1\}^n$, an integer $d \in [n]$ and a parameter $\epsilon \in \mathbb{R}^{\geq 0}$, so that $P(\mathbf{x}_1, \mathbf{x}_2) = 1$ if $||\mathbf{x}_1 \oplus \mathbf{x}_2||_0 \geq d(1+\epsilon)$ and 0 if $||\mathbf{x}_1 \oplus \mathbf{x}_2||_0 \leq d(1-\epsilon)$. Both of our constructions will distinguish between $d(1-\epsilon)$-close and $d(1+\epsilon)$-far vectors for $d \approx O(n/\log n)$. This means that the gap is quite large, approximately $O(n/\log n)$.

This construction is a robust m/n-compressing $\textsc{GapHamming}(n, d, \epsilon)$ PPH for any $m = n^{\Omega(1)}$, $d = o(n/\log \lambda)$ and any constant $\epsilon > 0$. Security of the construction holds under the (standard) assumption that collision-resistant hash function families (CRHFs) exist.

We now informally describe the idea of the construction which, in one word, is "subsampling". In slightly more detail, the intuition is to notice that if $\mathbf{x}_1 \in \{0,1\}^n$ and $\mathbf{x}_2 \in \{0,1\}^n$ are *close*, then *most small enough subsets* of indices of $\mathbf{x}_1$ and $\mathbf{x}_2$ will match identically. On the other hand, if $\mathbf{x}_1$ and $\mathbf{x}_2$ are *far*, then most *large enough subsets* of indices will differ. This leads us to the first idea for the construction, namely, fix a collection of sets $\mathcal{S} = \{S_1, \ldots, S_k\}$ where each $S_i \subseteq [n]$ is a subset of appropriately chosen size s. On input $\mathbf{x} \in \{0,1\}^n$, output $\mathbf{y} = (\mathbf{x}|_{S_1}, \ldots, \mathbf{x}|_{S_k})$ where $\mathbf{x}|_S$ denotes the substring of $\mathbf{x}$ indexed by the set S. The observation above tells us that if $\mathbf{x}_1$ and $\mathbf{x}_2$ are close (resp. far), so are $\mathbf{y}_1$ and $\mathbf{y}_2$.

However, this does not compress the vector $\mathbf{x}$. Since the union of all the sets $\bigcup_{i\in[k]} S_i$ has to be the universe $[n]$ (or else, finding a collision is easy), it turns out that we are just comparing the vectors index-by-index. Fortunately, it is not necessary to output $\mathbf{x}|_{S_i}$ by themselves; rather we can simply output the collision-resistant hashes. That is, we will let the PPH hash of $\mathbf{x}$, denoted $\mathbf{y}$, be $(g(\mathbf{x}|_{S_1}), \ldots, g(\mathbf{x}|_{S_k}))$ where g is a collision resistant hash function randomly drawn from a CRHF family.

This simple construction works as long as s, the size of the sets S_i, is $\Theta(n/d)$, and the collection $\mathcal{S}$ satisfies that any subset of disagreeing input indices $T \subseteq [n]$ has nonempty intersection with roughly the corresponding fraction of subsets S_i. The latter can be achieved by selecting the S_i of size $\Theta(n/d)$ at random, or alternatively as defined by the neighbor sets of a bipartite expander. We are additionally constrained by the fact that the CRHF must be secure against adversaries running in time $\text{poly}(\lambda)$. So, let $t = t(\lambda)$ be the smallest output size of the CRHF such that it is secure against $\text{poly}(\lambda)$-time adversaries. Since the input size s to the CRHF must be $\omega(t)$ so that g actually compresses, this forces $n/d = \omega(t)$, and thus $d = o(n/t)$.

Before presenting our construction more formally, we define our tools.

- We will use a family of CRHFs that take inputs of variable size and produce outputs of t bits and denote it by $\mathcal{H}_t = \{h : \{0,1\}^* \to \{0,1\}^t\}$. We implicitly assume a procedure for sampling a seed for the CRHF given a security parameter 1^λ. One could set $t = \omega(\log \lambda)$ and assume the exponential hardness of the CRHF, or set $t = \lambda^{O(1)}$ and assume polynomial hardness. These choices will result in different parameters of the PPH hash function.
- We will use an $(n, k, D, \gamma, \alpha)$-*bipartite expander* $G = (L \cup R, E)$ which is a D-left-regular bipartite graph, with $|L| = n$ and $|R| = k$ such that for every $S \subset L$ for which $|S| \leq \gamma n$, we have $|N(S)| \geq \alpha |S|$, where $N(S)$ is the set of neighbors of S. For technical reasons, we will need the expander to be δ-*balanced* on the right, meaning that for every $v \in R$, $|N(v)| \geq (1-\delta)nD/k$.

  A simple probabilistic construction shows that for every $n \in \mathbb{N}$, $k = o(n)$ and constant $a \in (0,1)$, and any $\gamma = o(\frac{k}{n \log(n/k)})$ and $D = \Theta(\log(1/\gamma))$ so that for every $\delta > 0$, δ-balanced $(n, k, D, \gamma, \alpha)$-bipartite expanders exist. In fact, there are even explicit efficient constructions that match these parameters [4].

The construction is in Table 2. We first discuss explicit parameter settings.

Setting the Parameters

The parameters required for this construction to be secure and constructible are as follows.

- Let $n \in \mathbb{N}$ and constant $\epsilon > 0$.
- We require two building blocks: a CRHF and an expander. So, let $\mathcal{H}_t = \{g : \{0,1\}^* \to \{0,1\}^t\}$ be a family of CRHFs secure against $\text{poly}(\lambda)$-time adversaries. Let G be a δ-balanced $(n, k, D, \gamma, \alpha)$-expander for a constant δ bounding the degree of the right-nodes, where n is the size of the left side of the graph, k is the size of the right, D is the left-degree, γn is the upper bound for an expanding set on the right that expands to a set of size α times the original size.
- These building blocks yield the following parameters for the construction: compression is $\eta = kt/n$ and our center for the Gap-Hamming problem is bounded by $\frac{\gamma n}{(1+\epsilon)} \leq d < \frac{k}{D(1-\epsilon)}$.

If we consider what parameter settings yield secure CRHFs with output size t and for what parameters we have expanders, we have a PPH construction where given any n and ϵ, there exists a $d = o(n)$ and $\eta = O(1)$ such that Construction 2 is a robust PPH for gap-Hamming. We will see that the smaller t is, the stronger the CRHF security assumption, but the better our compression.

Now, given these settings of parameters, we have the following theorem that Construction 2 is a robust Gap-Hamming PPH.

Table 2 Construction 2 of a robust GapHamming(n, d, ϵ) PPH family from CRHFs. Resulting parameters (n, m, d, ϵ) discussed in text.

Robust GapHamming(n, d, ϵ) PPH family $\mathcal{H}$ from any CRHF

Our (n, m, d, ϵ)-robust PPH family $\mathcal{H} = (\mathcal{H}.\mathsf{Samp}, \mathcal{H}.\mathsf{Eval})$ is defined as follows.

- $\mathcal{H}.\mathsf{Samp}(1^\lambda, n)$. Fix a δ-balanced $(n, k, D, \gamma, \alpha)$-bipartite expander $G = (L \cup R, E)$ (either deterministically or probabilistically). Sample a CRHF $g \leftarrow \mathcal{H}_t$. Output $h = (G, g)$.
- $\mathcal{H}.\mathsf{Hash}(h = (G, g), \mathbf{x})$. For every $i \in [k]$, compute the (ordered) set of neighbors of the i-th right vertex in G, denoted $N(i)$. Let $\hat{\mathbf{x}}^{(i)} := \mathbf{x}|_{N(i)}$ be $\mathbf{x}$ restricted to the set $N(i)$. Output

  $$h(\mathbf{x}) := \left(g(\hat{\mathbf{x}}^{(1)}), \ldots, g(\hat{\mathbf{x}}^{(k)})\right)$$

  as the hash of $\mathbf{x}$.
- $\mathcal{H}.\mathsf{Eval}(h = (G, g), \mathbf{y}_1, \mathbf{y}_2)$. Compute the threshold $\tau = D \cdot d \cdot (1 - \epsilon)$. Parse $\mathbf{y}_1 = (\hat{\mathbf{y}}_1^{(1)}, \ldots, \hat{\mathbf{y}}_1^{(k)})$ and $\mathbf{y}_2 = (\hat{\mathbf{y}}_2^{(1)}, \ldots, \hat{\mathbf{y}}_2^{(k)})$. Compute

  $$\Delta' = \sum_{i=1}^{k} \mathrm{Ind}(\hat{\mathbf{y}}_1^{(i)} \neq \hat{\mathbf{y}}_2^{(i)}),$$

  where Ind denotes the indicator predicate. If $\Delta' \leq \tau$, output **CLOSE**. Otherwise, output **FAR**.

▶ **Theorem 18.** *Let λ be a security parameter. Assuming that exponentially secure CRHFs exist, then for any polynomial $n = n(\lambda)$, and any constants $\epsilon, \eta > 0$, Construction 2 is an η-compressing robust property preserving hash family for* GapHamming(n, d, ϵ) *where $d = o(n/\log\lambda \log\log\lambda)$. Assuming only that polynomially secure CRHFs exist, for any constant $c > 0$, we achieve $d = o(n/\lambda^c)$.*

Proof. Before getting into the proof, we more explicitly define the parameters, including parameters associated with the expander in our construction:

1. Let $n \in \mathbb{N}$ be the input size and let $\epsilon > 0$ be any constant.
2. Our CRHF is $\mathcal{H}_t = \{g : \{0,1\}^* \to \{0,1\}^t\}$.
3. Our expander will be a δ-balanced $(n, k, D, \gamma, \alpha)$-expander, where $k < n/t$, $\gamma = o(\frac{k}{n \log(n/k)})$, $D = \Theta(\log(1/\gamma))$, and $\alpha > D \cdot \frac{d(1-\epsilon)}{\gamma n}$.
4. Our center for the gap-hamming problem is d, and is constrained by $\frac{\gamma n}{1+\epsilon} \leq d < \frac{k}{D(1-\epsilon)}$.
5. Constraint 4 implies that $k = n^{\Omega(1)}$, since $\frac{\gamma n \cdot D(1-\epsilon)}{1+\epsilon} < k$.

Now, we prove our construction is well-defined and efficient. Fix any $\delta, a \in (0, 1)$. In the full version, we explicitly prove that δ-balanced $(n, k, D, \gamma, \alpha = an)$-bipartite expanders exist and can be efficiently sampled for $k = o(n/\log n)$, $D = \Theta(\log\log n)$, and $\gamma = \tilde{\Theta}(1/\log n)$. Thus, sampling the graph G before running the construction is efficient. Once we have a G, sampling and running a CRHF $k = O(n)$ times is efficient. Comparing k outputs of the hash function is also efficient. Therefore, each of $\mathcal{H}.\mathsf{Samp}$, $\mathcal{H}.\mathsf{hash}$, and $\mathcal{H}.\mathsf{Eval}$ is efficient in $\lambda = \mathrm{poly}(n)$.

Now, we prove that Construction 2 is compressing. Points 2 and 3 mean that $m = k \cdot t < n/t \cdot t = n$, as required.

Lastly, we will prove our construction is robust. Let $\mathcal{A}$ be a PPT adversary. We will show that $\mathcal{A}$ (in fact, even an unbounded adversary) cannot find $\mathbf{x}_1$ and $\mathbf{x}_2$ such that $||\mathbf{x}_1 - \mathbf{x}_2|| \leq d(1-\epsilon)$ but $\mathcal{H}.\mathsf{Eval}(h, h(\mathbf{x}_1), h(\mathbf{x}_2))$ evaluates to **FAR**, and that $\mathcal{A}$ must break the collision-resistance of $\mathcal{H}_t$ in order to find $\mathbf{x}_1$ and $\mathbf{x}_2$ where $||\mathbf{x}_1 - \mathbf{x}_2|| \geq d(1+\epsilon)$ but $\mathcal{H}.\mathsf{Eval}(h, h(\mathbf{x}_1), h(\mathbf{x}_2))$ evaluates to **CLOSE**.

- First, consider any $\mathbf{x}_1, \mathbf{x}_2 \in \{0,1\}^n$ where $||\mathbf{x}_1 - \mathbf{x}_2||_0 \leq d(1-\epsilon)$. Let $\Delta = ||\mathbf{x}_1 - \mathbf{x}_2||_0$. So, consider the set $S \subset L$ corresponding to the indices that are different between $\mathbf{x}_1$ and $\mathbf{x}_2$, and $T = N(S) \subset R$. The maximum size of T is $|S| \cdot D$, the degree of the graph.

  For every $i \in T$, we get that the intermediate computation has $\hat{\mathbf{x}}_1^{(i)} \neq \hat{\mathbf{x}}_2^{(i)}$, but for every $j \notin T$, we have $\hat{\mathbf{x}}_1^{(j)} = \hat{\mathbf{x}}_2^{(j)}$ which implies $\hat{\mathbf{y}}_1^{(j)} = \hat{\mathbf{y}}_2^{(j)}$ after applying g. Therefore $\sum_{i=1}^{k} \mathrm{Ind}(\hat{\mathbf{y}}_1^{(i)} \neq \hat{\mathbf{y}}_2^{(i)}) \leq \sum_{i \in S} \mathrm{Ind}(\hat{\mathbf{y}}_1^{(i)} \neq \hat{\mathbf{y}}_2^{(i)}) + \sum_{j \notin S} \mathrm{Ind}(\hat{\mathbf{y}}_1^{(j)} \neq \hat{\mathbf{y}}_2^{(j)}) \leq \Delta \cdot D$.

  We set the threshold $\tau = D \cdot d \cdot (1-\epsilon)$ in the evaluation. Point 4 guarantees that $\tau < k$ (and implicitly implies $k > D(1-\epsilon)$), so because $D \cdot \Delta \leq D \cdot d(1-\epsilon) = \tau < k$, $\mathcal{H}.\mathsf{Eval}$ will evaluate $\Delta' \leq \tau$. Thus, $\mathcal{H}.\mathsf{Eval}$ will always evaluate to **CLOSE** in this case, regardless of the choice of CRHF.

- Now consider $||\mathbf{x}_1 - \mathbf{x}_2||_0 \geq d(1+\epsilon)$, and again, let $\Delta = ||\mathbf{x}_1 - \mathbf{x}_2||_0$ and define $S \subset L$ and $T \subset R$ as before.

  By point 4 again ($\gamma n \leq d(1+\epsilon)$), we can restrict S to S' where $|S'| = \gamma n$, and by the properties of expanders $|N(S')| \geq \gamma n \cdot \alpha$. Now, point 3 guarantees that $\tau = D \cdot d \cdot (1-\epsilon) < \alpha \cdot \gamma n$. So, for every $i \in T'$, $\hat{\mathbf{x}}_1^{(i)} \neq \hat{\mathbf{x}}_2^{(i)}$, and $|T'| \geq \alpha \cdot \gamma n > \tau$. Now we want to argue that with all but negligible probability over our choice of g, g will preserve this equality relation, and so $\Delta' = |T'|$. Given that our expander is δ-balanced for some constant $\delta > 0$, we have that $|\hat{\mathbf{x}}_1^{(i)}| = |\hat{\mathbf{x}}_2^{(i)}| = |N(r_i)| \geq (1-\delta)nD/k$. Now, point 3 states that the constraints have $k < n/t$, implying $n/k > t$. So, $(1-\delta)D \cdot n/k > (1-\delta)D \cdot t$.

  This means that every input to g will be larger than the output ($(1-\delta)$ is a constant and $D = \omega(1)$), and so if $g(\hat{\mathbf{x}}_1^{(i)}) = g(\hat{\mathbf{x}}_2^{(i)})$ but $\hat{\mathbf{x}}_1^{(i)} \neq \hat{\mathbf{x}}_2^{(i)}$ for any i, then our adversary has found a collision, which happens with all but negligible probability for adversaries running in time poly(λ).

  Therefore, with all but negligible probability over the choice of g and adversarially chosen $\mathbf{x}_1$ and $\mathbf{x}_2$ in this case, $\Delta' = \sum_{i=1}^{m'} \mathrm{Ind}(\hat{\mathbf{y}}_1^{(i)} \neq \hat{\mathbf{y}}_2^{(i)}) \geq \alpha \cdot \gamma n = \tau$, and $\mathcal{H}.\mathsf{Eval}$ outputs **FAR**. ◀

References

1 Noga Alon, Yossi Matias, and Mario Szegedy. The Space Complexity of Approximating the Frequency Moments. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, pages 20–29, 1996.

2 Benny Applebaum, Naama Haramaty, Yuval Ishai, Eyal Kushilevitz, and Vinod Vaikuntanathan. Low-Complexity Cryptographic Hash Functions. In *8th Innovations in Theoretical Computer Science Conference, ITCS 2017, January 9-11, 2017, Berkeley, CA, USA*, pages 7:1–7:31, 2017.

3 László Babai, Anna Gál, Peter G. Kimmel, and Satyanarayana V. Lokam. Communication Complexity of Simultaneous Messages. *SIAM J. Comput.*, 33(1):137–166, 2003.

4 Michael Capalbo, Omer Reingold, Salil Vadhan, and Avi Wigderson. Randomness Conductors and Constant-degree Lossless Expanders. In *Proceedings of the Thiry-fourth Annual ACM Symposium on Theory of Computing*, STOC '02, pages 659–668, New York, NY, USA, 2002. ACM. `doi:10.1145/509907.510003`.

5 Larry Carter and Mark N. Wegman. Universal Classes of Hash Functions (Extended Abstract). In *Proceedings of the 9th Annual ACM Symposium on Theory of Computing, May 4-6, 1977, Boulder, Colorado, USA*, pages 106–112, 1977.

6 Moses Charikar, Kevin C. Chen, and Martin Farach-Colton. Finding frequent items in data streams. *Theor. Comput. Sci.*, 312(1):3–15, 2004.

7 Scott Shaobing Chen, David L. Donoho, and Michael A. Saunders. Atomic Decomposition by Basis Pursuit. *SIAM Rev.*, 43(1):129–159, 2001.

8 Graham Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *J. Algorithms*, 55(1):58–75, 2005. doi:10.1016/j.jalgor.2003.12.001.

9 Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam Smith. Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data. *SIAM J. Comput.*, 38(1):97–139, March 2008. doi:10.1137/060651380.

10 Bella Dubrov and Yuval Ishai. On the randomness complexity of efficient sampling. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing, Seattle, WA, USA, May 21-23, 2006*, pages 711–720, 2006.

11 Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986.

12 Moritz Hardt and David P. Woodruff. How robust are linear sketches to adaptive inputs? In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 121–130. ACM, 2013. doi:10.1145/2488608.2488624.

13 Piotr Indyk. Stable Distributions, Pseudorandom Generators, Embeddings and Data Stream Computation. In *41st Annual Symposium on Foundations of Computer Science, FOCS 2000, 12-14 November 2000, Redondo Beach, California, USA*, pages 189–197, 2000.

14 Piotr Indyk and Rajeev Motwani. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 604–613, 1998.

15 T. S. Jayram, Ravi Kumar, and D. Sivakumar. The One-Way Communication Complexity of Hamming Distance. *Theory of Computing*, 4(1):129–135, 2008. doi:10.4086/toc.2008.v004a006.

16 Harini Kannan, Alexey Kurakin, and Ian J. Goodfellow. Adversarial Logit Pairing. *CoRR*, abs/1803.06373, 2018. arXiv:1803.06373.

17 J. Zico Kolter and Eric Wong. Provable defenses against adversarial examples via the convex outer adversarial polytope. *CoRR*, abs/1711.00851, 2017. arXiv:1711.00851.

18 Ilan Kremer, Noam Nisan, and Dana Ron. On Randomized One-round Communication Complexity. In *Proceedings of the Twenty-seventh Annual ACM Symposium on Theory of Computing*, STOC '95, pages 596–605, New York, NY, USA, 1995. ACM. doi:10.1145/225058.225277.

19 Eyal Kushilevitz, Rafail Ostrovsky, and Yuval Rabani. Efficient Search for Approximate Nearest Neighbor in High Dimensional Spaces. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, pages 614–623, New York, NY, USA, 1998. ACM. doi:10.1145/276698.276877.

20 Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards Deep Learning Models Resistant to Adversarial Attacks. *CoRR*, abs/1706.06083, 2017. arXiv:1706.06083.

21 Ilya Mironov, Moni Naor, and Gil Segev. Sketching in adversarial environments. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 651–660, 2008.

22 Jayadev Misra and David Gries. Finding Repeated Elements. *Sci. Comput. Program.*, 2(2):143–152, 1982.

23 J. Ian Munro and Mike Paterson. Selection and Sorting with Limited Storage. *Theor. Comput. Sci.*, 12:315–323, 1980.

24 Moni Naor and Eylon Yogev. Bloom Filters in Adversarial Environments. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*, pages 565–584, 2015.

25 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, May 14-17, 1989, Seattle, Washigton, USA*, pages 33–43, 1989.

26 Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. Certified Defenses against Adversarial Examples. *CoRR*, abs/1801.09344, 2018. arXiv:1801.09344.

27 Sivaramakrishnan Natarajan Ramamoorthy and Makrand Sinha. On the communication complexity of greater-than. In *53rd Annual Allerton Conference on Communication, Control, and Computing, Allerton 2015, Allerton Park & Retreat Center, Monticello, IL, USA, September 29 - October 2, 2015*, pages 442–444, 2015.

28 Aman Sinha, Hongseok Namkoong, and John C. Duchi. Certifiable Distributional Robustness with Principled Adversarial Training. *CoRR*, abs/1710.10571, 2017. arXiv:1710.10571.

29 David Woodruff. Optimal Space Lower Bounds for All Frequency Moments. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '04, pages 167–175, Philadelphia, PA, USA, 2004. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=982792.982817.

30 David P. Woodruff. *Efficient and private distance approximation in the communication and streaming models*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 2007. URL: http://hdl.handle.net/1721.1/42243.

31 Andrew Chi-Chih Yao. Some Complexity Questions Related to Distributive Computing (Preliminary Report). In *Proceedings of the 11h Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1979, Atlanta, Georgia, USA*, pages 209–213, 1979.