# Identifying Harmful Media in End-to-End Encrypted Communication: Efficient Private Membership Computation

Anunay Kulshrestha and Jonathan Mayer, *Princeton University*

https://www.usenix.org/conference/usenixsecurity21/presentation/kulshrestha

This paper is included in the Proceedings of the 30th USENIX Security Symposium.

August 11–13, 2021

978-1-939133-24-3

# Identifying Harmful Media in End-to-End Encrypted Communication: Efficient Private Membership Computation

Anunay Kulshrestha
*Princeton University*

Jonathan Mayer
*Princeton University*

## Abstract

End-to-end encryption (E2EE) poses a challenge for automated detection of harmful media, such as child sexual abuse material and extremist content. The predominant approach at present, perceptual hash matching, is not viable because in E2EE a communications service cannot access user content.

In this work, we explore the technical feasibility of privacy-preserving perceptual hash matching for E2EE services. We begin by formalizing the problem space and identifying fundamental limitations for protocols. Next, we evaluate the predictive performance of common perceptual hash functions to understand privacy risks to E2EE users and contextualize errors associated with the protocols we design.

Our primary contribution is a set of constructions for privacy-preserving perceptual hash matching. We design and evaluate client-side constructions for scenarios where disclosing the set of harmful hashes is acceptable. We then design and evaluate interactive protocols that optionally protect the hash set and do not disclose matches to users. The constructions that we propose are practical for deployment on mobile devices and introduce a limited additional risk of false negatives.

## 1 Introduction

The trend toward end-to-end encryption (E2EE) in popular messaging services [1], such as Apple iMessage [2], WhatsApp [3], Facebook Messenger [4], and Signal [5], has immense benefits. E2EE limits access to communications content to just the parties, providing a valuable defense against security threats, privacy risks, and—in some jurisdictions—illegitimate surveillance and other human rights abuses.

Adoption of E2EE does, however, come at a significant societal cost. A small proportion of users share harmful media, such as child sexual abuse material (CSAM), terrorist recruiting imagery, and most recently dangerous disinformation about causes of and cures for COVID-19 [6–9]. Present E2EE deployments do not support the predominant methods for automatically identifying this content.

For over a decade, popular platforms have relied on perceptual hash matching (PHM) to efficiently respond to harmful media [10]. PHM systems use perceptual hash functions (PHFs) to deterministically map media—most commonly images—to a space where proximity reflects perceptual similarity. PHFs are designed to be robust against common transformations, including geometric transformations, noise, and compression [11–19]. When a user shares media, a PHM system computes the perceptual hash and compares the value to a set of known hashes for harmful content. If the computed value is close to a hash in the set, the platform flags the user's media for a content moderation response.

In the United States, the National Center for Missing and Exploited Children (NCMEC) coordinates several datasets of known CSAM perceptual hashes, totaling millions of images [20, 21]. Similar CSAM hash clearinghouses exist in other countries, including the U.K. [22] and Canada [23]. The Global Internet Forum to Counter Terrorism (GIFCT), a coalition of technology firms, facilitates sharing tens of thousands of perceptual hashes for extremist material [24].

Because E2EE services by design do not have access to communications content, they cannot compute and compare perceptual hashes of user media. Law enforcement and civil society stakeholders worldwide have responded by pressing for a moratorium on E2EE adoption and "lawful access" schemes for encrypted communications [25–27].

In this work, we explore the technical feasibility of a middle ground: can an E2EE service take content moderation action against media that matches a perceptual hash set, without learning about non-harmful content, optionally without learning about harmful content, and optionally without disclosing the hash set? Our contributions to the literature, and the structure of the paper, are as follows:

- We formalize the problem of detecting perceptual hash matches in E2EE communications: private exact membership computation (PEMC) and private approximate membership computation (PAMC). We also describe limitations in the problem formulation, including both technical constraints and serious policy concerns that cannot be resolved through technical means (Section 2).
- We evaluate commonly used PHFs for predictive performance, so that we can both characterize the added privacy risk to E2EE communications from PHM false positives and contextualize the additional false negatives associated with certain of our protocol designs (Section 4).
- We evaluate client-side PEMC and PAMC designs, which are straightforward and practical for deployments where the set of perceptual hashes is not sensitive (Section 5).
- We design and evaluate novel interactive protocols for PEMC and PAMC (Section 6). Our protocols consist of four steps: bucketizing PHF values for efficient lookup

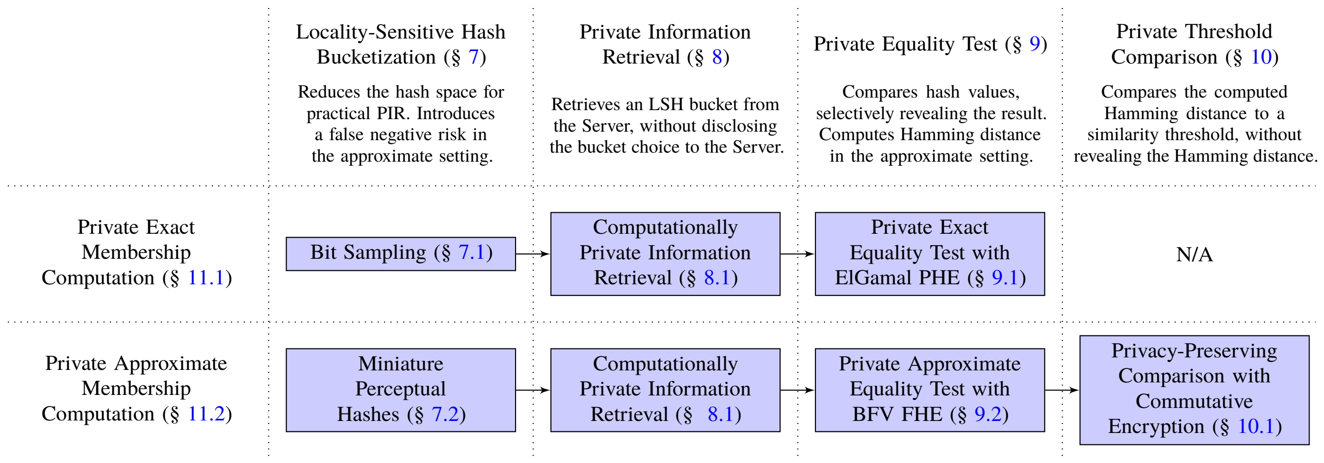
Locality-Sensitive Hash Bucketization (§ 7)
Reduces the hash space for practical PIR. Introduces a false negative risk in the approximate setting.
Private Information Retrieval (§ 8)
Retrieves an LSH bucket from the Server, without disclosing the bucket choice to the Server.
Private Equality Test (§ 9)
Compares hash values, selectively revealing the result. Computes Hamming distance in the approximate setting.
Private Threshold Comparison (§ 10)
Compares the computed Hamming distance to a similarity threshold, without revealing the Hamming distance.
Private Exact Membership Computation (§ 11.1)
Bit Sampling (§ 7.1)
Computationally Private Information Retrieval (§ 8.1)
Private Exact Equality Test with ElGamal PHE (§ 9.1)
N/A
Private Approximate Membership Computation (§ 11.2)
Miniature Perceptual Hashes (§ 7.2)
Computationally Private Information Retrieval (§ 8.1)
Private Approximate Equality Test with BFV FHE (§ 9.2)
Privacy-Preserving Comparison with Commutative Encryption (§ 10.1)

Figure 1: An overview of our protocols for privacy-preserving perceptual hash matching in E2EE communications.

(Section 7), private information retrieval for PHF buckets (Section 8), private equality testing (Section 9), and finally (for PAMC only) private threshold comparison (Section 10). Figure 1 depicts our overall protocol design. After presenting each step, we complete our protocols for PEMC and PAMC (Section 11), then describe our implementation and provide benchmarks (Section 12). The protocols that we propose are practical for deployment on mobile devices and, in PAMC, introduce limited false negatives beyond those inherent in PHFs.

The paper concludes with a discussion of related work (Section 13) and directions for future study.

## 2 Problem Formulation

We begin by explaining current PHM systems, using an abstraction (Section 2.1). Next, we formalize a set of properties for privacy-preserving PHM in E2EE communication (Section 2.2). Finally, we describe limitations of the problem formulation and our protocol designs (Section 2.3).

### 2.1 Perceptual Hash Matching

Current PHM systems function, abstractly, as follows.[1] A user (generically the Client) possesses media that they wish to share via a communications service (generically the Server). The Client transmits the media to the Server with transport encryption, and the Server receives the media in plaintext.

The Server then applies a $k$-bit perceptual hash function $\mathsf{PHF}_k(\cdot)$ to the media, generating the perceptual hash $x = \mathsf{PHF}_k(\mathsf{media}) \in \{0,1\}^k$.[2] The PHF preserves perceptual similarity between images as locality in the hash space, with robustness against common media transformations [11–19]. Most PHFs rely on the Hamming distance metric $d_H(\cdot,\cdot)$ to quantify perceptual similarity between hashes, and the Server selects a fixed similarity threshold $\delta_H < k$ for identifying nearly identical media.

Informally, a perceptual hash is a special type of locality-sensitive hash (LSH) [28, 29] that preserves similarity not only within hash buckets, but also across hash buckets.[3] If the Client's media is perceptually nearly identical to media with hash value $y$, with high probability $d_H(x,y) \leq \delta_H$. Otherwise, with high probability $d_H(x,y) > \delta_H$.

After computing the perceptual hash $x$, the Server compares $x$ to a set of $k$-bit perceptual hashes for media known to be harmful, $\mathcal{B} \subseteq \{0,1\}^k$.[4] Some PHM deployments require an exact hash match: the Server identifies media as harmful if $x \in \mathcal{B}$.[5] In other deployments, the Server uses approximate hash matching: the Server identifies media as harmful if $\exists y \in \mathcal{B}$ such that $d_H(x,y) \leq \delta_H$.[6]

If there is a perceptual hash match, the Server takes a content moderation action in response. The action could range from displaying a warning, to terminating the Client's account, to sharing the Client's identity with law enforcement [30].

**Privacy Properties.** Current PHM implementations depend on Server access to the Client's media, so that the Server can both compute $x$ and compare $x$ to hashes in $\mathcal{B}$. We refer to confidentiality for the Client's media and $x$ as *client privacy*. We consider these values as equivalent in privacy sensitivity because $x$ may have a known preimage (e.g., an image that has been publicly shared) and because PHF constructions typically leak information about media content [31].

[1]We generalize current PHM systems for clarity of presentation. Implementation specifics vary across communications services.

[2]We use $k = 256$ throughout this work, consistent with the state of the art in image PHFs [18].

[3]We do not offer a more formal definition of PHFs as a subset of LSHs, because LSHs usually incorporate a quantitative measure for similarity across inputs and are accompanied by a proof of matching probabilities.

[4]We assume in this abstracted system that $\mathcal{B}$ exclusively contains harmful media. As we discuss in Section 2.3, communications services and governments can use PHM for problematic purposes, and some already do.

[5]In deployments where a Server seeks to identify exact media file matches, it might use a cryptographic hash rather than a perceptual hash. We use perceptual hashes throughout this work for simplicity. The exact matching protocols that we design are compatible with cryptographic hashes.

[6]Exact hash matching is equivalent to approximate matching with $\delta_H = 0$. We treat exact matching separately in this work because more efficient client-side constructions and interactive protocols are available.

PHM systems also conceal the hash set $\mathcal{B}$ from the Client, by comparing hash values exclusively Server-side. We refer to confidentiality for $\mathcal{B}$ as *server privacy*. Maintaining server privacy is especially important for CSAM and other unlawful content, because revealing the hash set could enable evasion, disclose investigative techniques, or create legal liability.[7]

PHM reveals the existence of a perceptual hash match to the Server, because the Server compares $x$ to hashes in $\mathcal{B}$. The Server can then optionally disclose a match to the Client. We refer to a protocol that discloses a match to the Server as *server-revealing*, and we refer to a protocol that discloses a match to the Client as *client-revealing*. Both properties have privacy implications. If the Server learns about a match, that reveals information about the media and $x$. If the Client learns about a match, that reveals information about $\mathcal{B}$.

**Security Model.** PHM systems are fundamentally limited to semi-honest security, because they depend on perceptual hashing of media content. Parties could collude to conceal media content (e.g., by encrypting media) or take advantage of how PHFs are imperfect approximations of perceptual similarity (e.g., by applying transformations that PHFs miss). We provide additional explanation of why PHM lacks malicious security in Appendix A. While the security model might strike some readers as lax, PHM has proven valuable for identifying Clients who share harmful content either unknowingly or without technical sophistication. Communications services, law enforcement agencies, and child safety groups all rely on widely deployed PHM today [25–27].

### 2.2 Privacy-Preserving Perceptual Hash Matching

Our goal in this work is to adapt PHM for E2EE communications services. We develop protocols with predictive performance very similar to PHM, additional privacy properties, and an equivalent security model. Formally, we provide efficient solutions to two closely related problems, which we collectively term private membership computation (PMC).

**Definition 2.1. Private Exact Membership Computation (PEMC).** The Client inputs $x$ and the Server inputs $\mathcal{B}$. A $P$-revealing protocol outputs to party $P$ (Client or Server) whether $x \in \mathcal{B}$ while maintaining client privacy.[8]

**Definition 2.2. Private Approximate Membership Computation (PAMC).** The Client inputs $x$ and the Server inputs $\mathcal{B}$. A $P$-revealing protocol outputs to party $P$ whether $\exists y \in \mathcal{B}$ such that $d_H(x,y) \leq \delta_H$ while maintaining client privacy.

Our interactive PEMC protocol fully satisfies the definition above. Our interactive PAMC construction slightly departs from the definition—the design induces a small probability of false negatives, which we evaluate in Section 7.2.

[7]Distribution of certain types of harmful content can be unlawful, and some platforms have expressed concern that disclosing perceptual hashes might give rise to liability for distributing the hashed content. We do not take a position on whether these concerns are substantiated.

[8]A client-revealing PEMC protocol is the same as the private membership test/query protocols that have been studied in prior work (Section 13).

**Privacy Properties.** We guarantee client privacy in the PMC protocols that we propose. The very purpose of E2EE is to maintain the confidentiality of the Client's message content (i.e., the media and $x$) with respect to the Server. We prove client privacy at each step of our protocols.

We develop constructions both with and without server privacy. In some applications, such as providing the Client with disinformation warnings, the Server might not consider the hash set $\mathcal{B}$ sensitive and might be willing to disclose $\mathcal{B}$ to the Client. We evaluate efficient client-side constructions for these applications in Section 5. Our work primarily focuses on interactive protocols that guarantee server privacy, and we prove the property at every step of our protocols.

Each of the interactive protocols that we design is optionally server-revealing or client-revealing, since there may be scenarios (disinformation warnings again being an example) where content moderation logic could be located with the Client and the Server does not need notice of a match. We focus on server-revealing designs in the main text, and we present client-revealing constructions in Appendix E. Our protocol designs, like PHM, are independent of the content moderation action that occurs in response to a match.

**Security Model.** The PMC protocols that we present have, overall, the same semi-honest security model as PHM. As with PHM, parties could evade detection by concealing content or exploiting PHF inaccuracy. We provide further discussion of the security model in Appendix A.

### 2.3 Limitations

The problems that we formulate, and the protocols that we propose to solve those problems, have significant limitations.

First, our problem formulation assumes that E2EE services would use PHM to counter harmful media, such as CSAM and extremist content. But a service could use PHM for other purposes. Platforms could, for example, use the constructions we describe to implement censorship or illegitimate surveillance—and might be compelled by a government that is not committed to free speech and the rule of law. Some platforms *already* rely on PHM for these purposes (e.g., [32]). While future work may be able to partially increase confidence that the hash set $\mathcal{B}$ exclusively contains harmful media (see Section 14), at base *some* entity or entities will have to curate and validate $\mathcal{B}$. This lack of trust may be insurmountable, and readers who consider implementing the constructions we describe should carefully weigh the policy and ethics implications.

Second, there is a risk of false positives inherent in PHM—with low probability, a Client's media will match a value in the hash set even though there is no perceptual similarity. In these instances, an innocent E2EE Client may—depending on the content moderation response—lose communications confidentiality, have their account terminated, or become the subject of a law enforcement inquiry. We evaluate the predictive performance of PHFs in Section 4.[9] While our PMC

[9]The false positive rate depends on the nature of the Client media, the

constructions do not induce *additional* false positives beyond PHM, some readers may find the current level of false positives inadequate for deployment in E2EE communications.[10]

Third, the constructions that we propose would inherently increase the attack surface for E2EE services. Implementations could have security errors, and if an adversary were able to access the backend for a server-revealing protocol, they could manipulate $\mathcal{B}$ to extract a subset of communications content.

Fourth, implementation of the constructions we propose could have adverse consequences for international relations. A choice to implement in a democratic jurisdiction could undermine human rights abroad, by weakening arguments in favor of E2EE services, providing new arguments in favor of PHM systems, or equipping governments with new tools for censorship and surveillance. Implementation could also have economic harms; if a jurisdiction required hash matching, that could undermine the competitiveness of domestic services.

Fifth, our constructions do not protect PHF algorithms. We believe this is a reasonable choice, since most PHF algorithms are public (including Facebook PDQ [18, 33]), and since there is no advantage to an adversary in gaming a PHF rather than encrypting media before sending it with a messaging app.

We emphatically *do not* take a position on whether E2EE services should adopt the constructions we propose, especially given the significant risks to user privacy and security, the challenge for international human rights, and the feasibility of circumvention. We are particularly concerned about whether the level of trust required in $\mathcal{B}$ is attainable through technical or legal means. The goal of our work is to constructively explore the tradeoffs of a possible direction for encryption policy, similar to recent proposals on law enforcement access to encrypted devices [34–36] and messages [37, 38].

## 3 Notation and Primitives

### 3.1 Notation

For ease of exposition, we fix some notation. For positive integer $n$, we use $[n]$ to denote the set $\{0,\ldots,n-1\}$. We denote the Hamming distance between two $k$-bit strings as $d_H : \{0,1\}^k \times \{0,1\}^k \to [k+1]$. For integer $m$, we denote $[\cdot]_m$ as reduction modulo $m$. We denote the next (ceiling), previous (floor) and nearest integer to $x$ as $\lceil x \rceil$, $\lfloor x \rfloor$, and $\lfloor x \rceil$ respectively. These functions are applied coefficient-wise to polynomials. We denote the bitstring of $i$ zeroes as $0^i$.

The protocol notation $(P_o, C_o, S_o) \leftarrow \mathsf{Protocol}(P_i, C_i, S_i)$ denotes that the Server and Client run protocol $\mathsf{Protocol}$ with public input $P_i$, Client input $C_i$ and Server input $S_i$. At the end of the protocol, values in $P_o$ are public while the Client and Server receive outputs $C_o$ and $S_o$ respectively.

The term *privacy* and its derivatives refer to computational privacy, unless otherwise qualified (such as *statistical* privacy).

hash set $\mathcal{B}$, the PHF, and the similarity threshold $\delta_H$.

[10]Our PAMC design slightly decreases PHM false positives, for the same reason that it slightly induces false negatives (see Section 7.2).

### 3.2 Primitives

We briefly introduce primitives here and provide further details in Appendix B.

**Brakerski/Fan-Vercauteren (BFV) Cryptosystem.** The BFV cryptosystem is an asymmetric fully homomorphic encryption (FHE) scheme based on the hardness of the ring learning with errors (RLWE) problem [39, 40]. Plaintexts are polynomials from the quotient ring $\mathbb{Z}[X]/(X^n+1)$ with $n$, a power of 2, and coefficients modulo $t$.

Our proposed protocols also require a distributed variant of the BFV cryptosystem. The two parties can jointly generate an aggregated public key $\mathsf{pk}_{\mathsf{agg}}$ for encryption.[11] Either party can encrypt data using $\mathsf{pk}_{\mathsf{agg}}$ but no single party can decrypt it. At a later time in the protocol, the two parties can jointly re-encrypt a ciphertext encrypted under $\mathsf{pk}_{\mathsf{agg}}$ to one decipherable by $\mathsf{sk}_c$ or $\mathsf{sk}_s$. This collective key switching protocol allows a delegated party (in our case, the Server) to decrypt the final result of the protocol. We recall key aggregation and collective key switching for BFV in Appendix B, Figure 5 [41, 43, 44].

Finally, we define the following packings (embeddings) of $k$-bit strings $a = a_1 \cdots a_k$ into polynomials in $R_t$ [44, 45].

$$\mathsf{FHE.Pack}_1(a) = \sum_{i=0}^{k-1} a_i x^i \quad \mathsf{FHE.Pack}_2(a) = a_0 - \sum_{i=1}^{k-1} a_i x^{n-i}$$

We use these packings to construct a private Hamming distance protocol, which we introduce in Section 9.2.

**ElGamal Cryptosystem.** The ElGamal cryptosystem is an asymmetric partially homomorphic encryption (PHE) scheme [46]. For semantic security, it is defined over cyclic groups in which solving the decisional Diffie–Hellman (DDH) problem is hard. Using elliptic curve groups over finite fields yields both computational and communication efficiency gains [47]. We denote $\mathsf{PHE.Add}$ and $\mathsf{PHE.SMul}$ as $+_E$ and $\times_E$ respectively.

**Privacy-Preserving Comparison.** The millionaires' problem or privacy-preserving comparison (PPC), first formalized by Yao in 1982, is a foundational puzzle of secure multiparty computation. In the canonical formulation of the problem, two millionaires with wealth $w_1$ and $w_2$ respectively wish to find out who is richer (i.e., compute whether $w_1 < w_2$ or $w_1 \geq w_2$) while keeping their wealth values private [48]. We recall a recent PPC protocol based on commutative encryption from Liu et al. in Appendix B [49]. The protocol is marginally slower than the fastest two-party PPC protocol (from Damgård et al. [50]) but requires a third of the total communication.

## 4 PHF Predictive Performance

Before turning to PMC constructions, we evaluate commonly used PHFs for predictive performance.[12] The evaluation

[11]The parties can generate aggregated relinearization keys, as in [41]. We describe a protocol for generating aggregated Galois keys by adapting techniques from Chen et. al and Mouchet et. al in Appendix B [41, 42].

[12]We use the term predictive performance in our evaluation for concision. To be precise, we evaluate the extent to which PHFs cluster perceptually similar images and spread out dissimilar images in hash space. These properties,

Table 1: The content of our image dataset.

| *Type* | *Sources* | *Count* |
|---|---|---|
| Objects | Microsoft COCO [51] | 328,045 |
| Faces | Wikipedia and IMDB [52] | 357,994 |
| Social Media | Instagram, Twitter, and Flickr [53–55] | 2,846,748 |

provides context for both the privacy risk that PHM false positives pose for E2EE communications and the additional false negatives associated with our PAMC construction. We conduct the evaluation by compiling an image dataset, then examining the perceptual hash Hamming distance distributions for perceptually similar and perceptually dissimilar images.

### 4.1 Image Dataset and PHFs

We aggregate a dataset of ~3.5 million distinct images from public sources. The dataset is intended to reflect the diverse types of images that users share via E2EE services. Our sources include existing datasets of social media uploads, images of everyday objects, and images of faces (Table 1). We evaluate PHFs on samples of images from our dataset.

We consider 6 open-source PHFs: Average Hash (aHash), blockHash, Difference Hash (dHash), Facebook PDQ, pHash,[13] and Wavelet Hash (wHash) [11–13, 15, 17, 18].[14] Each PHF uses a different set of image features, which we describe in Appendix D. Note that when processing an image, PHFs typically both resize the image to a small square and convert the image to grayscale.[15] Certain PHFs that we analyze produce $k$-bit perceptual hashes with Hamming norm $\frac{k}{2}$ as each bit of the hash reflects a comparison to a median value.[16]

PHFs are designed to produce similar outputs on similar inputs and dissimilar outputs for dissimilar inputs. We empirically evaluate the predictive performance of the 6 PHFs by considering these two cases separately.

### 4.2 Comparing Similar Images

We randomly sample 10,000 images from our dataset and apply 5 transforms to each image. The transforms reflect possible user actions to make images visually appealing (gamma correction), highlight image components (cropping, rescaling, and rotation), and share images (noise from compression).

We uniformly sample the extent of the transform from a range. We consider three sets of ranges, reflecting increasing levels of image transformation (Appendix D, Table 10).

- *Rotation*. Sample $\Delta_r$, rotate by $\Delta_r^\circ$ clockwise.
- *Noise Addition*. Sample $\sigma$, add noise $\sim \mathcal{N}(\mu_g, \sigma)$.[17]
- *Cropping*. Sample $\Delta_c$, crop by $\Delta_c = \frac{\text{cropped dimensions}}{\text{original dimensions}}$.
- *Gamma Correction*. Sample $\gamma$, apply correction $\gamma$.
- *Rescaling*. Sample $\Delta_a$, rescale by $\Delta_a = \frac{\text{width}}{\text{height}}$.

For each of the 6 PHFs, we compare the perceptual hash of each original image in our sample to the hashes of the corresponding transformed images. We normalize the pairwise Hamming distances by $k$ (the hash length) and compute summary statistics for the resulting distance distributions.

Table 2 presents the results, which illustrate the significant effect of both the PHF type and the transform type on predictive performance for perceptually similar images.[18] A robust PHF would correspond to a distribution that is centered close to 0, as it would produce similar outputs on similar images.[19]

The PHFs we evaluate are generally robust to modest extents of specific transformations. For certain transforms (such as rotation), or for greater extents (see the Supplementary Information [56]), PHFs are not very robust at detecting perceptual similarity. If a current PHM system uses one of these PHFs, there may be a significant risk of false negatives.

### 4.3 Comparing Dissimilar Images

We evaluate each PHF's predictive performance on perceptually dissimilar images using an analogous method.[20] We randomly sample 5,000 images of each type, totaling 15,000 images. Then, for each PHF, we generate the perceptual hash of each image and calculate the Hamming distance between each pair of hashes. We finally normalize the Hamming distances and compute summary statistics for each distribution.

Table 2 shows the results of our evaluation.[21] If a PHF generates dissimilar hashes for dissimilar images, the distribution of normalized Hamming distances will likely be centered around the expected distance between two uniformly random bit strings, 0.5. Among distributions centered close to 0.5, a narrower distribution is desirable; a smaller variance implies that perceptual hashes of dissimilar images are consistently spread out in the hash space, which reduces the false positive rate when the PHF is used in a PHM system. We find that the distribution is centered around 0.5 for each PHF, and PDQ is both the closest to 0.5 and exhibits the least variance.

According to our evaluation—and assuming our evaluation approximates real-world distributions of similar and dissimilar images—PDQ would produce the fewest false positives in a PHM system and wHash would produce the fewest false negatives. Note that our constructions are agnostic to an implementer's choice of both PHF and hash length, so long as the PHF supports Hamming distance comparison.

We now turn to privacy-preserving designs for perceptual hash matching, beginning with client-side constructions.

combined with a similarity threshold, determine PHM predictive performance.

[13]We use the DCT-based hash from the pHash C++ library [12].

[14]We omit Microsoft PhotoDNA because the algorithm is not public.

[15]The conversion usually averages color values or computes luminance.

[16]This property is attainable for every PHF that we consider. Certain PHFs involve comparison to a mean value, which could be replaced with a median. In the unusual case where PHF pre-output values equal the median, a PHF could deterministically set bits for those values to achieve Hamming norm $\frac{k}{2}$.

[17]The mean $\mu_g$ of the distribution is the mean gray value of the image.

[18]We provide a normalized Hamming distance histogram for each transform type, transform level, and PHF in the Supplementary Information [56].

[19]Current PHM systems appear to use a similarity threshold of about 0.1.

[20]We assume that distinct images in our dataset are perceptually dissimilar. While this assumption is generally accurate, there are limitations. We find, for example, that PHFs consistently perform worst on distinct images of faces—a result we attribute to multiple images of the same celebrities in our dataset.

[21]The Supplementary Information [56] includes a histogram for each PHF.

Table 2: Results from evaluating PHF predictive performance. Upper rows: Mean ($\mu$) and std. dev. ($\sigma$) of pairwise normalized Hamming distance between 10,000 sampled images and transformed versions for each PHF (Level 1 of all transforms). Bottom row: Mean ($\mu$) and std. dev. ($\sigma$) of pairwise normalized Hamming distances between perceptual hashes of 15,000 distinct images.

| | aHash | blockHash | dHash | pHash | PDQ | wHash |
|---|---|---|---|---|---|---|
| | $\mu$ ($\sigma$) | $\mu$ ($\sigma$) | $\mu$ ($\sigma$) | $\mu$ ($\sigma$) | $\mu$ ($\sigma$) | $\mu$ ($\sigma$) |
| Rotation | 0.164 (0.103) | 0.183 (0.093) | 0.192 (0.083) | 0.229 (0.109) | 0.189 (0.095) | **0.077 (0.056)** |
| Noise Addition | **0.003 (0.006)** | 0.004 (0.009) | 0.026 (0.023) | 0.002 (0.004) | 0.022 (0.021) | 0.009 (0.021) |
| Cropping | **0.058 (0.047)** | 0.091 (0.059) | 0.125 (0.069) | 0.164 (0.095) | 0.193 (0.093) | 0.076 (0.056) |
| Gamma Corr. | 0.336 (0.143) | 0.321 (0.138) | 0.365 (0.120) | 0.363 (0.094) | **0.004 (0.007)** | 0.004 (0.027) |
| Rescaling | **0.037 (0.031)** | 0.059 (0.042) | 0.086 (0.051) | 0.109 (0.066) | 0.132 (0.067) | 0.049 (0.041) |
| Mean of Means | 0.120 | 0.132 | 0.159 | 0.173 | 0.108 | **0.043** |
| Distinct Images | 0.493 (0.100) | 0.498 (0.075) | 0.496 (0.041) | 0.498 (0.032) | **0.500 (0.032)** | 0.493 (0.101) |

Table 3: Benchmarks for data structures with $|\mathcal{B}| = 2^{20}$ and $k = 256$. $t_{pos}$ and $t_{neg}$ are mean query times for positive and negative queries, averaged over $10^4$ runs. Size represents the resulting index size. FPR is the false positive rate. Threshold $\delta_H = 25 \approx k/10$ for multi-index hashing.

| *Data Structure* | $t_{pos}$ *(s)* | $t_{neg}$ *(s)* | *Size (MB)* | *FPR* |
|---|---|---|---|---|
| Binary Search[22] | 0.829 | 0.127 | 32.0 | 0 |
| Bitwise Trie [57] | 0.359 | 0.215 | 26.2 | 0 |
| Bloom Filter [58] | 0.603 | 0.153 | 3.8 | $\sim 10^{-6}$ |
| Cuckoo Filter [59] | 0.100 | 0.108 | 3.01 | $\sim 10^{-6}$ |
| Multi-Index Hashing [60] | 0.066 | 0.009 | 34.2 | 0 |

## 5 Client-Side Matching

In deployments that do not require server privacy, client-side data structures are a straightforward approach to PEMC and PAMC. These constructions use an index stored on the Client device, and they trivially guarantee client privacy because all perceptual hash matching computation is local to the Client.

Table 3 provides performance benchmarks for 4 client-side exact matching constructions and 1 approximate matching construction. We perform all tests on a 6-core Intel i7-10710U@1.10GHz with a 12MB cache and 32GB RAM. We generate a random set $\mathcal{B}$ and random negative queries. We randomly sample from $\mathcal{B}$ for positive queries.

In exact matching applications where a small false positive risk may be acceptable or where memory is especially constrained, cuckoo filters [59] are particularly practical (and outperform Bloom filters [61]).[23] The resulting index size is small and lookups are fast, but we emphasize that the index still leaks information about $\mathcal{B}$: an adversary can easily check whether a hash is in $\mathcal{B}$ without notifying the Server.

Multi-index hashing (MIH) is an approximate search algorithm for Hamming space that exhibits sub-linear runtime behavior for uniformly distributed sets of strings $\mathcal{B}$ [60]. The speedup over a linear search baseline is significant even when $\mathcal{B}$ is not uniformly distributed, which is likely true for a set of perceptual hashes. The size of the MIH index does, however, grow linearly with $\mathcal{B}$. Our benchmarks show that MIH is an efficient solution for client-revealing PAMC without server privacy, and deployment on mobile devices would be practical.

[22] We use the C++ Standard Template Library implementation.

[23] Bloom filters require $1.44\log_2(\text{FPR}^{-1})$ bits per item while cuckoo filters require $\frac{\log_2(\text{FPR}^{-1})+2}{\alpha}$ bits per item for load factor $\alpha$. We present benchmarks for FPR $\approx 10^{-6}$ and $\alpha = 0.955$. Cuckoo filters are smaller than corresponding Bloom filters for expected FPR $< 3\%$ [59].

## 6 Privacy-Preserving Matching

We now turn to PMC constructions that maintain server privacy. Our general approach consists of the following steps:

- The Server partitions the hash set $\mathcal{B}$ into localized buckets, for more efficient retrieval by the Client (Section 7).
- The Client uses private information retrieval to obtain the relevant bucket(s) of hashes, under homomorphic encryption (Section 8).
- The Client obliviously computes a Hamming distance between $x$ and each encrypted hash (Section 9).
- For PAMC, the Client and Server jointly compute a thresholded comparison for each distance (Section 10).

At the conclusion of the PMC protocols, the Server learns whether there is a perceptual hash match.[24] Figure 1 depicts the overall design of the protocols.

We present each step of the protocols, then we assemble the components into complete protocols (Section 11) and provide benchmarks (Section 12).

## 7 Locality-Sensitive Hash Bucketization

Locality-sensitive hashing (LSH) techniques map similar items to the same bin with high probability [28, 29].[25] As the name suggests, LSH techniques are sensitive to locality in the input space, defined according to a similarity metric. We use Hamming distance for comparing perceptual hashes throughout this work, as discussed in Section 2.1. Applying LSH solves the problem of dividing the set of perceptual hashes $\mathcal{B}$ into disjoint localized buckets for efficient retrieval.

**LSH.Setup.** A Server holds a hash set $\mathcal{B}$ of $k$-bit elements and knows a function $\phi(\cdot)$, which the Server applies to buckets before indexing. Both parties know the $l$-hash family $\mathcal{L}$. The Server computes the LSH index Ind for all $w \in \{0,1\}^l$.[26]

[24] See Appendix E for a discussion of client-revealing protocol variants.

[25] LSH function families are precise mathematical objects. We refer readers to the referenced work for formal definitions.

[26] As discussed in the security analysis, if there is no $y \in \mathcal{B}$ such that $\mathcal{L}(y) = w$, the Server should pad the index with dummy $y'$ in order to prevent leaking information about the distribution of hashes in $\mathcal{B}$.

Table 4: FNR induced by 20-bit LSH families in PAMC.Query over a random sample of $10^4$ images with Level 1 transforms. The transformed 256-bit hashes are within normalized Hamming distance 0.1 from the original.

| *LSH Family* | *Transformations* | | | | | *Avg.* |
|---|---|---|---|---|---|---|
| | Rotate | Noise | Crop | Gamma | Rescale | |
| pHash-LSH | 0.2339 | 0.0282 | 0.2695 | 0.0551 | 0.2531 | 0.168 |
| Random-LSH | 0.5034 | 0.0642 | 0.6414 | 0.0866 | 0.5982 | 0.3788 |

$$\mathsf{Ind}[w] \leftarrow \phi(\{y \in \mathcal{B} : \mathcal{L}(y) = w\})$$

$\phi(\cdot)$ returns ElGamal or BFV ciphertext(s) which hide the contents of the LSH bucket from the Client (see Section 11).

**LSH.Query.** For a $k$-bit element $e$, this function returns the $l$-bit locality-sensitive hash $\mathcal{L}(e)$. As the LSH family $\mathcal{L}$ is publicly known, either party can compute LSH.Query. For clarity, we denote LSH.Query by $\mathcal{L}(\cdot)$ in subsequent sections.

### 7.1 Bit Sampling

For a $k$-bit perceptual hash $s = s_1 s_2 \cdots s_k$, the $l$-bit locality-sensitive hash $\mathcal{L}_E(s) = s_1||s_2||\cdots||s_l$ is an $l$-bit prefix. Bit sampling yields a valid LSH family with hashes $h_i(s) = s_i$, which sample individual bits of the input. We can sample and concatenate any $l$ bits of the input to construct an $l$-bit LSH in Hamming space [28]. We choose the first $l$ for simplicity, and we use $\mathcal{L}_E(s)$ to construct the cPIR index in PEMC.

### 7.2 Miniature Perceptual Hashes

LSH bucketization does not induce false negatives in exact hash matching, because if there is a match the Client's hash will deterministically map to the same bucket as the match. Specifically, if there is a hash $y \in B$ such that $d_H(x,y) \le \delta_H$, then $\mathcal{L}_E(x) = \mathcal{L}_E(y)$. In the approximate setting, however, locality-sensitive hashing can induce false negatives.[27] Intuitively, there might be an approximate match in the hash set, but the Client's hash maps to a different bucket. More precisely, it is possible that there exists $y \in B$ such that $d_H(x,y) \le \delta_H$ but $\mathcal{L}(y) \ne \mathcal{L}(x)$, such that the Server erroneously concludes that for all $z \in B$, $\delta_H < d_H(x,z)$. We find that these errors are common when applying bit sampling to perceptual hashes.

We introduce a new LSH family that uses $l$-bit miniature perceptual hashes, significantly lowering the expected FNR in PAMC protocols. Intuitively, we leverage the fact that PHFs are (informally) a type of LSH that preserve locality for similar images. PHFs can operate on lower-resolution grayscale images to produce smaller hashes. If two perceptually similar images have close hash values in a $k$-bit perceptual hash space, the hash values will likely move closer—and eventually become identical—as the perceptual hash space shrinks.

Almost all PHFs produce hashes of perfect square lengths, because input images are resized to small squares before processing. Each bit of the output perceptual hash typically corresponds to a single pixel of the resized square. To construct an $l$-bit LSH, we compute the smallest perfect square greater than $l$, $l' = \min\{i^2 : l < i^2, i \in \mathbb{Z}\}$, and then truncate the $l'$ bit perceptual hash to its first $l$ bits

$$\mathcal{L}_a(x) = \mathsf{PHF}_{l'}(x)_1||\cdots||\mathsf{PHF}_{l'}(x)_l.$$

[27] By false negative, we mean an instance when PHM would have returned a match but PAMC does not. For purposes of identifying harmful content, these are false negatives *in addition to* the false negatives inherent in PHM.

Table 5: Bucket size distribution of LSH indices (for a random sample of $10^4$ images), averaged over five transforms.

| *LSH Family* | *Bucket Count by Bucket Size* | | |
|---|---|---|---|
| | 1 | 2 | 3 |
| pHash-LSH | 9947.4 | 26 | 0.2 |
| Random-LSH | 9845 | 76 | 1 |

Table 4 illustrates the performance improvement (in PAMC) of the miniature PHF-based LSH over bit sampling (Random-LSH) for PHF = pHash and $l = 20$. pHash-LSH decreases the FNR for every transform we considered and led to an improvement of over 50% on average (from 0.3788 to 0.168).[28]

We further validate that pHash-LSH does not lead to an imbalanced LSH index. An imbalanced LSH index can increase expected computation and communication required for both PEMC and PAMC protocols. Table 5 describes the bucket size distributions of the resulting pHash-LSH and Random-LSH indices. The pHash-LSH index mapped 99.47% of samples (on average) to unique buckets in our simulations, compared to 98.45% of samples in the baseline (Random-LSH) index.

The results in Tables 4 and 5 show that the Server may use a miniature PHF for the LSH bucketization step of PAMC, significantly reducing the FNR attributable to PAMC.

### 7.3 Correctness and Security

**Correctness.** It is easy to verify that for all $y, y' \in \mathcal{B}$, the relation $=_{\mathcal{L}} \subseteq \mathcal{B} \times \mathcal{B}$, defined as $y =_{\mathcal{L}} y' \iff \mathcal{L}(y) = \mathcal{L}(y')$, is an equivalence relation. The resulting equivalence classes are partitions of $\mathcal{B}$. The Server constructs LSH index Ind mapping every element in the domain $\{0,1\}^l$ of $\mathcal{L}$ to results of function $\phi(\cdot)$, which operates on an entire equivalence class (LSH bucket) $\subseteq \mathcal{B}$.

As $\mathcal{L}$ is public, a Client with perceptual hash $x$ can reliably determine the equivalence class that $x$ belongs to. The Client can then privately retrieve the element of Ind associated with that equivalence class, represented by $\mathcal{L}(x)$.

**Security.** Encrypted buckets in the LSH index Ind should be padded with dummy ciphertexts to a constant size (bucket size $b$ in PEMC.Setup) to prevent leaking information about the distribution of LSH values for elements of $\mathcal{B}$.

## 8 Private Information Retrieval (PIR)

Using a PIR protocol, a Client can privately obtain a set of LSH buckets that are expected to contain hashes similar to $x$

[28] Assuming the Server is equally likely to encounter every independent transformation of a harmful image.

(if any exist in $\mathcal{B}$) with high probability.[29] In our single-server model, we provide computational client privacy guarantees.[30]

### 8.1 Computationally Private Information Retrieval

A cPIR protocol allows a Client to retrieve an element $e_c$ from a Server that stores a database $D = \{e_i : 1 \le i \le |D|\}$ without revealing choice $c$. Computational privacy of the choice follows from protocol-specific cryptographic hardness assumptions. Since Chor et al. first formalized the problem in 1995, it has been extensively studied due to its ubiquity [62, 63]. In order to answer a cPIR query, a Server must necessarily operate on all database elements [62]. Otherwise, if the Server could omit operating on certain elements, it could infringe on the privacy guarantee of Client choice $c$.

This requirement makes cPIR computationally expensive.[31] Recent advances in cPIR have exploited FHE schemes to significantly lower computation and communication costs in practice, which has led to deployable protocols [65, 66].

The first of these, XPIR, builds on work by Stern [63] and uses the BFV cryptosystem [67]. XPIR computes the dot product of a homomorphically encrypted query vector $\vec{q}$ and the entire database, using FHE.Absorb. Server sends

$$\begin{aligned} r &= (e_1,\dots,e_{|D|})\cdot(q_1,\dots,q_{|D|})^T \\ &= \sum_{i=1}^{|D|} e_i \times_{BFV} q_i = \sum_{i=1}^{|D|} \mathsf{FHE.Absorb}(pk,e_i,q_i) \\ &= \mathsf{FHE.Enc}\Big(pk, 0\cdot(\sum_{i=0}^{c-1} e_i) + 1\cdot e_c + 0\cdot(\sum_{i=c+1}^{|D|} e_i)\Big) \\ &= \mathsf{FHE.Enc}(pk,e_c). \end{aligned}$$

The Client holds the secret key sk and decrypts the response to retrieve $e_c$. While XPIR is the most computationally efficient cPIR protocol, the communication cost of $|D|$ ciphertexts is prohibitively expensive for large databases.

SealPIR is an improvement over XPIR, which seeks to reduce query size at the cost of increased server-side computation [66]. Instead of sending an encrypted query vector of $|D|$ ciphertexts, the Client sends a single ciphertext: the encrypted choice FHE.Enc(pk,$c$). The Server uses FHE.Sub to *expand* the encrypted choice FHE.Enc(pk,$c$) into the equivalent encrypted choice vector $(q_1,\dots q_{|D|})$. Subsequent improvements offer further communication-computation tradeoffs [65].[32]

Notice that Ind (Section 7) is a PIR database indexed by elements of $\{0,1\}^l$. cPIR.Query is a protocol that guarantees computational privacy to a Client retrieving elements from Ind. The retrieved element is a ciphertext in the range of $\phi(\cdot)$.

**cPIR.Query.** LSH family $\mathcal{L}$ is publicly known. Client sends query FHE.Enc(pk,$\mathcal{L}(x)$) to the Server. Server sends reply FHE.Enc(pk, Ind[$\mathcal{L}(x)$])) (as in SealPIR) to the Client.

### 8.2 Privacy

**Lemma 8.1. Client Privacy.** cPIR.Query does not reveal any information about the Client index $\mathcal{L}(x)$ to the Server.

*Proof.* SealPIR relies on the BFV cryptosystem, which is based on the intractability of the RLWE problem.[33] Angel et al. present a proof that we omit here for space [66]. ■

For medium-term security of the BFV cryptosystem, $n = 2048$ and a 60-bit coefficient modulus $q$ are sufficient [44, 68]. In particular, SealPIR uses $t = 2^{23}$ and $q = 2^{60} - 2^{18} + 1$.

## 9 Private Equality Test

After the Client privately retrieves one or more LSH buckets of encrypted hashes, it performs computation on the encrypted hashes. The Client returns the transformed ciphertexts to the Server, which can then learn the result of the equality tests.

For PEMC, we exploit the partial homomorphism of the ElGamal cryptosystem to perform comparisons.[34] We generalize this approach to PAMC by using the BFV cryptosystem for Hamming distance computation. We describe server-revealing protocols here and client-revealing variants in Appendix E.

### 9.1 Private Exact Equality Test

Suppose a Client and a Server hold perceptual hashes $x$ and $y$ respectively. They wish to privately test whether $x = y$. The Server generates an ElGamal key pair $(\mathsf{sk}_s,\mathsf{pk}_s) = \mathsf{PHE.KeyGen}(q,E,G)$ and sends $\mathsf{pk}_s, (C_y,C'_y) = \mathsf{PHE.Enc}(\mathsf{pk}_s,y)$ to the Client. The Client computes

$$(C_{-x},C'_{-x}) = \mathsf{PHE.Enc}(\mathsf{pk}_s,-x)$$

$$\rho_{-x}(C_y,C'_y) = r \times_E \Big[(C_y,C'_y) +_E (C_{-x},C'_{-x})\Big]$$

for randomizer $r \xleftarrow{\$} \mathbb{Z}_q^*$. The randomizer ensures that if $x \ne y$, the decrypted value $r(y-x)$, which is known to the Server, does not reveal anything about $x$ (given that the Server also knows $y$). The Client returns $(C,C') = \rho_{-x}(C_y,C'_y)$ to the Server, which concludes that $\mathsf{PHE.DecChk}(\mathsf{sk}_s,(C,C'),0) \iff x = y$.

#### 9.1.1 Correctness

Notice that the Client response $\rho_{-x}(C_y,C'_y)$ is an encryption of $r(y-x)$ under $\mathsf{pk}_s$.

$$\begin{aligned} \rho_{-x}(C_y,C'_y) &= r \times_E \Big[(C_y,C'_y) +_E (C_{-x},C'_{-x})\Big] \\ &= r \times_E \Big((r_y + r_{-x})\cdot G, (y-x)\cdot G + (r_y + r_{-x})\cdot pk\Big) \\ &= \Big(r(r_y + r_{-x})\cdot G, r(y-x)\cdot G + r(r_y + r_{-x})\cdot pk\Big) \\ &= \mathsf{PHE.Enc}(\mathsf{pk}_s, r(y-x)) \end{aligned}$$

If $x = y$, $\rho_{-x}(C_y,C'_y)$ is an encryption of 0. $\mathsf{PHE.DecChk}(\mathsf{sk}_s, \rho_{-x}(C_y,C'_y),0) = 1$ and the Server correctly concludes that

[29] While PEMC requires retrieving only one LSH bucket, some PAMC implementations might retrieve additional buckets to minimize false negatives.

[30] Other forms of PIR (e.g., differentially private PIR) are compatible with our protocols and offer tradeoffs between performance and privacy.

[31] A cPIR response requires at least $O(|D|/\log\log|D|)$ operations [64].

[32] The computation increase is proportional to $|D|$, so a marginal communication decrease may impose a prohibitively high computation cost in our setting of large $|D|$.

[33] Yasuda et al. present a discussion of distinguishing and decoding attacks against the general LWE problem and related work [44].

[34] In PEMC, the Client decrypts BFV ciphertexts from cPIR to obtain ElGamal ciphertexts. The Client then computes on the ElGamal ciphertexts.

$x = y$. Otherwise if $x \neq y$, then $\rho_{-x}(C_y, C'_y)$ is an encryption of a nonzero random element of $\mathbb{Z}_q^*$, revealing nothing about $x$ to the Server. In this case, $\mathsf{PHE.DecChk}(\mathsf{sk}_s, \rho_{-x}(C_y, C'_y), 0) \neq 1$ and the Server correctly concludes that $x \neq y$.

### 9.1.2 Privacy

Semantic security of the ElGamal cryptosystem relies on the intractability of ECDDH in the elliptic curve group used. We use `secp256k1` for optimized arithmetic.

**Lemma 9.1. Server Privacy.** The private exact equality test protocol reveals no information about $y \in \mathcal{B}$ to a Client.

*Proof.* Without secret key $\mathsf{sk}_s$, the Client cannot decrypt $(C_y, C'_y)$. Semantic security of the ElGamal cryptosystem ensures that $(C_y, C'_y)$ reveals no information about $y$. ∎

**Lemma 9.2. Client Privacy.** The private exact equality test protocol reveals no information about $x$ to a Server that decrypts $\rho_{-x}(C_y, C'_y)$ if $x \notin \mathcal{B}$.

*Proof.* If $x \neq y$, the Server could learn $r(y - x)$ by decrypting $\rho_{-x}(C_y, C'_y)$. Decryption in our scheme is extremely inefficient, however, especially for a large randomizer $r$ as it amounts to solving the ECDLP. In any event, $r(y - x)$ does not reveal anything about $x$ to the Server without the knowledge of randomizer $r$, which is only known to the Client. ∎

## 9.2 Private Approximate Equality Test

Now suppose a Client and a Server hold $k$-bit perceptual hashes $x$ and $y$ respectively. They wish to privately test whether $d_H(x, y) \leq \delta_H$ for a public threshold $\delta_H \in [k]$. We adapt the server-revealing variant of a protocol proposed by Yasuda et al. for biometric authentication, using BFV FHE rather than somewhat homomorphic encryption (SHE) as in the original [44, 45]. Using FHE, we are able to harness SealPIR's performance gains (via $\mathsf{FHE.Expand}$ and $\mathsf{FHE.Absorb}$) over SHE-based PIR protocols like MulPIR [65].[35]

The protocol operates on encryptions of $\mathsf{FHE.Pack}_1(y)$ and $\mathsf{FHE.Pack}_2(x)$. The Client homomorphically transforms a packed encryption of $y$ using its own packed ciphertext. The Server can decrypt the transformed ciphertext to learn the required Hamming distance.

The Server generates a BFV key pair $(\mathsf{sk}_s, \mathsf{pk}_s)$ and sends $\mathsf{pk}_s, c_y = \mathsf{FHE.Enc}(\mathsf{pk}_s, \mathsf{FHE.Pack}_1(y))$. For $J_x, J_y \in R_t$,

$$J_x = -\sum_{i=0}^{l_b-1} x^{n-i} \qquad\qquad J_y = \sum_{i=0}^{l_a-1} x^i$$

the Client computes $c_x = \mathsf{FHE.Enc}(\mathsf{pk}_s, \mathsf{FHE.Pack}_2(x))$

$$\zeta(c_x, c_y) = -2^{-1}\{(2 \cdot c_y - J_y) \cdot (2 \cdot c_x - J_x)\} + 2^{-1}\{J_y \cdot J_x\}$$

and returns $\zeta(c_x, c_y)$ to the Server, which decrypts it. If $p_{x,y} \in R_t$ such that $p_{x,y} = \mathsf{FHE.Dec}(\mathsf{sk}_s, \zeta(c_x, c_y))$, the Server concludes $d_H(x, y) \leq \delta_H \iff p_{x,y}(0) \leq \delta_H$.

[35] Compared to SealPIR, MulPIR reduces communication but increases Server computation. In our setting, the communication cost is already low enough to be practical (see Table 8). We could also use FHE for exact equality testing, but PHE incurs much lower communication and computation costs.

### 9.2.1 Correctness

Protocol correctness follows from Lemma C.1 [44, 45].

### 9.2.2 Privacy

**Lemma 9.3.** $p_{x,y}$ only reveals the Hamming distance $d_H(x, y)$ and no other information about $x$ to a Server.

*Proof.* Yasuda et. al. provide a proof in the original text that we omit here for space [44, 45]. ∎

The privacy guarantee is weaker than client privacy, which requires that the Server learn $d_H(x, y) \overset{?}{\leq} \delta_s$ without learning $d_H(x, y)$. We next describe how to achieve that property.

## 10 Private Threshold Comparison

Suppose a Client and a Server hold values $v_c, v_s \in [t]$ for some modulus $t$. A private threshold comparison (PTC) protocol allows the two parties to privately decide whether $v_s - v_c \leq \delta$ for some threshold $\delta \in [t]$.

We construct a server-revealing PTC protocol using a privacy-preserving comparison (PPC) protocol. Recall that a server-revealing PPC protocol allows a Server to learn whether its private value exceeds a Client's private value. For Client and Server inputs $a$ and $b$ respectively, $\mathsf{PPC}(\cdot, \{a\}, \{b\})$ returns `true` if and only if $a < b$. Notice that $\mathsf{PPC}(\cdot, \{v_c + \delta\}, \{v_s\})$ returns `false` $\iff v_s \leq v_c + \delta \iff v_s - v_c \leq \delta$.

We use this property to hide the computed Hamming distance in PAMC from both parties. At the end of the protocol, the Server only learns whether the Hamming distance is $\leq \delta_H$.

## 10.1 Hiding the Hamming Distance

We return to private approximate equality testing (PAET). Recall that the Client sends $\zeta(c_x, c_y)$ to the Server (Section 9.2). If the Client offsets $\zeta(c_x, c_y)$ by a random polynomial, the Server will not learn the Hamming distance upon decryption. Osadchy et al. propose a similar technique but use computationally expensive oblivious transfer (OT) to undo the offset [69]. We show how the two parties can jointly undo the offset and only reveal whether $d_H(x, y) \leq \delta_H$ using a much more efficient PPC protocol.

Suppose the Client samples $r \in R_t$ such that $r(0) < t - k$.[36] The Client computes $\mathsf{FHE.Enc}(\mathsf{pk}_s, r)$ and sends $\zeta(c_x, c_y) + c_r$. The Server can then decrypt with $\mathsf{sk}_s$ to obtain $p_{x,y} = \mathsf{FHE.Dec}(\mathsf{sk}_s, \zeta(c_x, c_y) + c_r) \in R_t$.

As an immediate consequence of Lemma C.1 for $l_a = l_b = k$, if $r(0) < t - k$, the constant term of the decrypted polynomial $p_{x,y} \in R_t$ equals the difference of $d_H(x, y)$ and the constant term of the Client randomizer $r \in R_t$: $d_H(x, y) = p_{x,y}(0) - r(0)$.

The Client knows $r(0)$ but does not know $p_{x,y}(0)$ while the Server knows $p_{x,y}(0)$ but does not know $r(0)$. Neither party can compute $d_H(x, y)$. Using the PPC protocol described above, with Client input $r(0)$ and Server input $p_{x,y}(0) - \delta_H$, the two

[36] This restriction can be relaxed by using two PPCs. If $r(0) \geq t - k$, there is a possibility that $r(0) + d_H(x, y) \geq t$. It follows that $p_{x,y}(0) < r(0)$ and $\mathsf{PPC}(\cdot, \{r(0) + \delta_H\}, \{p_{x,y}(0)\})$ always returns `false`. If $d_H(x, y) > \delta_H$, the PAMC protocol would result in a PTC-induced false positive.

parties can jointly determine whether $d_H(x,y) \le \delta_H$. Recall that PPC returns `false` $\iff p_{x,y}(0) - \delta_H \le r(0) \iff p_{x,y}(0) - r(0) \le \delta_H \iff d_H(x,y) \le \delta_H$. The Server concludes $d_H(x,y) \le \delta_H \iff$ PPC returns `false`.

### 10.2 Privacy

**Lemma 10.1. Client Privacy.** The PTC protocol only reveals bit $d_H(x,y) \overset{?}{\le} \delta_H$ and nothing else about $x$ to a Server.

*Proof.* Without knowledge of Client randomizer $r$, a Server cannot undo the offset to learn $d_H(x,y)$. The security of the PPC protocol guarantees that the Server cannot learn randomizer $r$. Liu et al. provide a security proof for the PPC protocol that we omit here for space [49]. ■

## 11 Private Membership Computation

We now combine the primitives from the preceding sections to construct PEMC and PAMC protocols. Recall that our overall protocols include four steps, as shown in Figure 1 and described in Section 6. First, the Server partitions the hash set into buckets with LSH. Second, the Client retrieves one or more relevant buckets using PIR. Each bucket contains homomorphic (PHE or FHE) encryptions of perceptual hashes. Third, the Client obliviously computes the Hamming distance between $x$ and each hash in the retrieved bucket(s). Finally, in the approximate case, the Client and Server jointly compute a thresholded comparison for each Hamming distance value.

### 11.1 Private Exact Membership Computation

Recall that the Client holds a $k$-bit string $x$ and the Server holds a set of $k$-bit strings $\mathcal{B}$. The Server seeks to learn whether $x \in \mathcal{B}$ without the Client revealing $x$ (client privacy) or the Server revealing $\mathcal{B}$ (server privacy). We construct a server-revealing PEMC protocol using bit-sampling LSH, cPIR, and ElGamal PHE that is practical for large sets $\mathcal{B}$.

The protocol requires an initial Server setup phase, PEMC.Setup, before queries can be performed. The Server generates an ElGamal key pair and constructs an LSH index over $\mathcal{B}$ using any bit-sampling LSH family $\mathcal{L}_E$. The keys of the LSH index are mapped to element-wise encryptions of the corresponding bucket (equivalence class of $=_{\mathcal{L}}$).

**PEMC.Setup.** The Server generates ElGamal key pair $(\mathsf{sk}_s, \mathsf{pk}_s)$ and sets function $\phi(\{s_1,\ldots,s_b\}) = \{\mathsf{PHE.Enc}(\mathsf{pk}_s, s_i) : 1 \le i \le b\}$. The Server then runs $(\{\mathcal{L}\},\{\},\{\mathsf{Ind}\}) \leftarrow \mathsf{LSH.Setup}(\{k,l\},\{\},\{\mathcal{B},\phi_e\})$ (Section 7). The Client generates $(\mathsf{sk}_c, \mathsf{pk}_c) \leftarrow \mathsf{FHE.Keygen}(q,t,n,\sigma)$.

Queries are performed using the PEMC.Query protocol (Figure 2). The Client computes $\mathcal{L}_E(x)$, and uses cPIR to retrieve element-wise encryptions in the associated bucket. The Client then subtracts an encryption of $x$ from each ciphertext and randomizes the result. The Client sends a permutation of the transformed encrypted bucket to the Server, which checks whether any of the returned ciphertexts decrypt to 0.[37] At the end of PEMC.Query, the Server learns whether $x \in \mathcal{B}$.

PEMC.Query

**Inputs:**

Public: LSH family $\mathcal{L}$, ElGamal public key $\mathsf{pk}_s$, and BFV public key $\mathsf{pk}_c$.

Server: PIR database Ind.

Client: $k$-bit string $x$, $\mathsf{sk}_c$.

**Protocol:**

1. Client and Server run
   $\gamma_s \leftarrow \mathsf{cPIR.Query}(\{\mathsf{pk}_c, \mathcal{L}\},\{x\},\{\mathsf{Ind}\})$.
2. Client decrypts reply $\gamma_s$ to obtain
   $\mathsf{Ind}[\mathcal{L}(x)] = \mathsf{FHE.Dec}(\mathsf{sk}_c, \gamma_s)$ and computes
   $(C_{-x}, C'_{-x}) \leftarrow \mathsf{PHE.Enc}(\mathsf{pk}_s, -x)$
   $\rho_{-x}(C_y, C'_y) = r \times_E \left[(C_y, C'_y) +_E (C_{-x}, C'_{-x})\right]$
   $\gamma_c \leftarrow \{\rho_{-x}(C_y, C'_y) : (C_y, C'_y) \in \mathsf{Ind}[\mathcal{L}(s)]\}$
   for randomizers $r \leftarrow_{\$} \mathbb{Z}_q^*$.
   Client sends random permutation $\pi(\gamma_c)$ to the Server.
3. Server checks whether
   $$\bigvee_{(C,C') \in \pi(\gamma_c)} \mathsf{PHE.DecChk}(\mathsf{sk}_s, (C,C'), 0) \overset{?}{=} 1.$$

**Outputs:**

Server: $x \overset{?}{\in} \mathcal{B}$.

Figure 2: PEMC.Query

The correctness of the PEMC protocol follows from the correctness of each component primitive (discussed in the preceding sections).

#### 11.1.1 Privacy

Following Lemmas 8.1, 9.1, and 9.2, our PEMC construction provides client and server privacy.

**Theorem 11.1. Client Privacy.** After the PEMC protocol, the Server learns whether $x \in \mathcal{B}$ and nothing more about $x$ or the Client's media.

**Theorem 11.2. Server Privacy.** After the PEMC protocol, the Client learns no information about $\mathcal{B}$.

#### 11.1.2 Efficiency

Table 6 shows the computation and communication cost of our PEMC protocol. We present benchmarks for our implementation in Section 12.

### 11.2 Private Approximate Membership Computation

We now extend our PEMC protocol to the approximate setting. Recall that the Client holds $k$-bit string $x$ and the Server holds a set of $k$-bit strings $\mathcal{B}$. At the end of a PAMC protocol, the Server should learn whether there exists $y \in B$

[37]Permuting the bucket defends against a malicious Server that tries to encode information into the order of bucket contents.

PAMC.Query

**Inputs:**

Public: Hash length $k$, LSH family $\mathcal{L}$, threshold $\delta_H$, bucket size $b$, BFV parameters $q,t,n$, and $\sigma$, and public keys $\mathsf{pk}_c$ and $\mathsf{pk}_s$.

Server: PIR database $\mathsf{Ind}$ and BFV secret key $\mathsf{sk}_s$.

Client: $k$-bit string $x$ and BFV secret key $\mathsf{sk}_c$.

**Protocol:**

1. Client and Server choose $p_1$ and run $\mathsf{pk}_{\mathrm{agg}} \leftarrow \mathsf{FHE.KeyAgg}(p_1)$.
2. Client and Server run $\mathsf{cPIR.Query}(\{\mathsf{pk}_{\mathrm{agg}},\mathcal{L}\},\{s\},\{\mathsf{Ind}\})$.
3. Client and Server run FHE.CKS and the Client finally obtains $c_y \leftarrow \mathsf{FHE.Enc}(\mathsf{pk}_{\mathrm{agg}},\mathsf{Ind}[\mathcal{L}(s)])$.
4. Client computes
   $c_{x,i} \leftarrow \mathsf{FHE.Enc}(\mathsf{pk}_s,\mathsf{FHE.Pack}_2(\underbrace{0^k \,\|\cdots\|\, 0^k}_{i \text{ times}} \| x))$
   $c_{r,i} \leftarrow \mathsf{FHE.Enc}(\mathsf{pk}_s,r_i)$, $\gamma_c \leftarrow \{\zeta(c_{x,i},c_y,c_{r,i}) : 1 \le i \le b\}$ for randomizers $r_i \in R_t$ with $r_i(0) < t-k$, sends $\gamma_c$ to Server.
5. Client and Server run FHE.CKS and Server decrypts $\gamma_c$ to obtain $v_i = d_H(x,y_i) + r_i(0) + \sum_{j=1}^{i-1} \|y_j\|_H$ for $1 \le i \le b$.
6. Client and Server run, for $1 \le i \le b$:
   $\alpha_i \leftarrow \mathsf{PPC}(\{\ldots\},\{r_i(0)+(i-1)\cdot\|x\|_H\},\{v_i-\delta_H\})$.
7. Server computes the result $\bigvee_{i=1}^{b} \widetilde{\alpha_i}$.

**Outputs:**

Server: $y \overset{?}{\in} \mathcal{B}$ such that $d_H(x,y) \le \delta_H$ (with high probability) or false (with low probability).

Figure 3: PAMC.Query

such that $d_H(x,y) \le \delta_H$ for some threshold $0 \le \delta_H \le k$. The Server should not learn any other information.

Similar to PEMC, the PAMC protocol also requires a setup phase (PAMC.Setup) before queries can be performed. Unlike the ElGamal ciphertexts in PEMC.Setup, however, index Ind in PAMC.Setup contains plaintext packings of LSH buckets.

**PAMC.Setup.** The Server sets $\phi_c(\{s_1,\ldots,s_b\}) = \mathsf{FHE.Pack}_1(s_1 \| \cdots \| s_b)$. The Server runs $(\{\mathcal{L}\},\{\},\{\mathsf{Ind}\}) \leftarrow \mathsf{LSH.Setup}(\{k,l\},\{\},\{\mathcal{B},\phi_c\})$ (Section 7).

We extend Lemma C.1 in order to pack $b = n/k$ perceptual hashes in a single BFV plaintext, where $n$ is the BFV polynomial degree. This step allows the Client to retrieve an entire encrypted bucket (equivalence class of $=_{\mathcal{L}_A}$) efficiently using cPIR.[38] We prove the extension, Lemma C.2, in Appendix C. Queries are performed using PAMC.Query (Figure 3).

Note that the cPIR buckets in PAMC are plaintext packings of perceptual hashes. Thus, unlike cPIR in PEMC, cPIR in PAMC requires a distributed BFV public key $\mathsf{pk}_{\mathrm{agg}}$ because the Client should not be able to completely decrypt the cPIR response. For cPIR recursion parameter $d$, the SealPIR response is of the form $\mathsf{FHE.Enc}^d(\mathsf{pk}_{\mathrm{agg}},\cdot)$. After $d-1$ joint decryptions (using FHE.CKS), the Client finally obtains an encryption of the packed bucket $c_y = \mathsf{FHE.Enc}(\mathsf{pk}_{\mathrm{agg}},\mathsf{FHE.Pack}_1(s_1 \| \cdots \| s_b))$.

The Client transforms the retrieved ciphertext in $b$ different ways (using results of Lemmas C.1 and C.2), which allows the Server to compute the result of $b$ private approximate equality tests (Section 9.2) in parallel. Another instance of FHE.PKS enables the Server to decrypt the result of the PAETs. The Server does not, however, learn the true Hamming distances (corresponding to each of the $b$ elements of the bucket) due to Client randomization. This randomization allows the two parties to run $b$ instances of a private threshold comparison (Section 10) protocol. The Server aggregates the results from all $b$ instances to compute the result of the PAMC protocol. At the end of PAMC.Query, the Server learns whether $y \in B$ such that $d_H(x,y) \le \delta_H$ but nothing more.

The correctness of the PAMC protocol follows from the correctness of each component primitive (discussed in the preceding sections).

### 11.2.1 Privacy

Following Lemmas 8.1, 9.3, and 10.1, our PAMC construction provides client and server privacy.

**Theorem 11.3. Client Privacy.** After the PAMC protocol, the Server learns whether whether $y \in B$ such that $d_H(x,y) \le \delta_H$ and nothing more about $x$ or the Client's media.

**Theorem 11.4. Server Privacy.** After the PAMC protocol, the Client learns no information about $\mathcal{B}$.

### 11.2.2 Efficiency

Table 6 shows the computation and communication cost of our PAMC protocol. We present benchmarks for PAMC operations in Section 12.

## 12 Implementation and Benchmarks

We implemented our PEMC and PAMC protocols in C++ using the SEAL, SealPIR, NTL, and Botan libraries [66, 70]. We then ran benchmarks with both parties running locally, using the same hardware as in Section 5.

We present benchmark results in Tables 7, 8, and 9.[39] As in Section 5, we generate a random set $\mathcal{B}$ and random queries for testing. We find that both the PEMC and PAMC protocols are practical for deployment on mobile devices. We also find that computation costs are lower (all else equal) with increasing bucket count. Table 9 provides per-operation benchmarks for PAMC.[40] cPIR.Query remains the most expensive operation, both in terms of computation and communication.[41]

[38] Lemma C.2 uses the fact that $k$-bit PHFs typically produce—or could be adapted to produce—hashes with constant Hamming norm $k/2$. Otherwise, our PAMC protocol follows Lemma C.1 and supports 1 hash per FHE value.

[39] PAMC and PEMC.Query benchmarks are averaged over 100 runs while PEMC.Setup benchmarks are averaged over 10 runs.

[40] PAMC per-operation benchmarks are averaged over 100 runs.

[41] Our implementation of cPIR.Query is single-threaded and does not benefit from recent SEAL optimizations, including more efficient packing.

Table 6: Compute and communication cost of our PEMC and PAMC protocols. $\mathcal{B}$ is the Server hash set, $d$ is the cPIR recursion parameter, $l$ is the LSH family size, $(n,t,q_1)$ are BFV parameters, $q_2$ is the ElGamal modulus, and $F$ is the BFV expansion factor.

| *Operation* | *Computation* | | *Communication* |
|---|---|---|---|
| | *Client* | *Server* | |
| PEMC.Setup | 1 polynomial mul. | $3\,\lvert\mathcal{B}\rvert+1$ EC point mul. | Only required to fix public parameters. |
| PAMC.Setup | – | $(\lvert\mathcal{B}\rvert/2^l)$ FHE.Pack$_1$ op. | Only required to fix public parameters. |
| PEET | $3+\lvert\mathcal{B}\rvert/2^l$ EC point mul. | $\lvert\mathcal{B}\rvert/2^{l-1}$ EC point mul. | $(\lvert\mathcal{B}\rvert/2^{l-1})(\log q_2+1)$ bits. |
| FHE.KeyAgg | 1 polynomial mul. | 1 polynomial mul. | $n\log q_1$ bits. |
| cPIR.Query | $4d\lceil\sqrt[d]{2^l}/n\rceil+F^{d-1}$ polynomial mul. | $d\sqrt[d]{2^l}$ FHE.Expand op., $2^l$ FHE.Absorb op. | $2n(d\lceil\sqrt[d]{\lvert\mathcal{B}\rvert}/n\rceil+F^{d-1})\log q_1$ bits. |
| FHE.CKS | $b+d-1$ polynomial mul. | $b+d-1$ polynomial mul. | $(b+d-1)n\log q_1$ bits. |
| PAET | $6(\lvert\mathcal{B}\rvert/2^l)$ polynomial mul. | $(\lvert\mathcal{B}\rvert/2^l)$ polynomial mul. | $n(\lvert\mathcal{B}\rvert/2^{l-1})(\log q_1)$ bits. |
| PTC | $(\lvert\mathcal{B}\rvert/2^{l-1})\log t$ EC point mul. | $(\lvert\mathcal{B}\rvert/2^{l-1})\log t$ EC point mul. | $3(\lvert\mathcal{B}\rvert/2^l)(\log t)(\log q_2+1)$ bits. |

Table 7: PEMC benchmarks with $k=256$ and SealPIR parameters $(n,d)=(2048,2)$ for varying set sizes $|\mathcal{B}|$ and LSH buckets.

| $\lvert\mathcal{B}\rvert$ | $2^{19}$ *buckets* | | | $2^{20}$ *buckets* | | | $2^{21}$ *buckets* | | |
|---|---|---|---|---|---|---|---|---|---|
| | *Setup (s)* | *Query (s)* | *Comm. (KB)* | *Setup (s)* | *Query (s)* | *Comm. (KB)* | *Setup (s)* | *Query (s)* | *Comm. (KB)* |
| $2^{20}$ | 175.4 | 0.75 | 394.45 | 175 | 0.75 | 394.32 | – | – | – |
| $2^{21}$ | 357.1 | 1.41 | 394.71 | 352.3 | 1.34 | 394.45 | 355.7 | 1.36 | 394.32 |
| $2^{22}$ | 735.3 | 2.88 | 395.25 | 698.9 | 2.6 | 394.71 | 703 | 2.52 | 394.45 |
| $2^{23}$ | 1480 | 6.73 | 396.3 | 1421 | 5.37 | 395.25 | 1408 | 4.97 | 394.71 |
| $2^{24}$ | – | – | – | 2841 | 13 | 396.3 | 2831 | 10.5 | 395.25 |

Table 8: PAMC benchmarks with $k=256$ and SealPIR parameters $(n,d)=(2048,2)$ for varying set sizes $|\mathcal{B}|$.

| $\lvert\mathcal{B}\rvert$ | $2^{20}$ *buckets* | | |
|---|---|---|---|
| | *Setup (s)* | *Query (s)* | *Comm. (KB)* |
| $2^{20}$ | 37.2 | 27.5 | 508.07 |
| $2^{21}$ | 37.4 | 27.5 | 586.06 |
| $2^{22}$ | 37.4 | 27.7 | 742.03 |
| $2^{23}$ | 37.7 | 28.3 | 1053.98 |

Table 9: PAMC benchmarks (per op.) with $|\mathcal{B}|=2^{20}$, $l=20$, $k=256$, $n=2048$, and $t=12$. PAMC requires $b+d-1$ FHE.CKS ops. as well as $b$ PAET and PTC ops. for bucket size $b$ and recursion level $d$.

| *Operation* | *Computation* (s) | *Communication* (kB) |
|---|---|---|
| PAMC.Setup | 37.2 | — |
| FHE.KeyAgg | 0.0004 | 15.36 |
| cPIR.Query | 27.4 | 368.64 |
| FHE.CKS | 0.012 | 46.08 |
| PAET | 0.004 | 30.72 |
| PTC | 0.016 | 1.188 |

## 13 Related Work

In this section, we contextualize our work in the literature on content moderation for E2EE services, private membership tests, and private biometric authentication. We do not recount the literature on locality-sensitive hashing [28, 29], private information retrieval [62–67, 71], or privacy-preserving comparison [48–50, 72], since those are well-known primitives. We refer the reader to the cited works for detail.

**Content Moderation for E2EE Services.** In the work closest to our own, Singh and Farid describe a Client-Server protocol for computing the perceptual hash of an image under Paillier PHE [73]. The Client encrypts the original image and partially computes the hash. The Server then decrypts the partial output and completes the hash computation. The protocol's security model is very different from ours: it attempts to keep the original image private, but it reveals the perceptual hash to the Server. We treat images and perceptual hashes as equivalent in privacy (Section 2). The paper's use of PHE, curiously, has no relation to its security model. The protocol provides equivalent security under plaintext and PHE, because it does not use PHE for joint computation. Rather, the PHE ciphertexts encrypted by the Client are immediately decrypted by the Server. The protocol's performance is also very different: it requires the Client to send 276MB per image,[42] and its computation requirements are also impractical [73].

Recent work by Facebook [4], Grubbs et al. [74], Dodis et al. [75], Tyagi et al. [76], and Mayer [77] addresses how an E2EE service can verify that a reported message is authentic, without initially having access to its content. These "message franking" schemes could be combined with our protocols, providing proof that a user actually sent harmful media.

Tyagi et al. propose content traceback methods for E2EE services [78]. Traceback could integrate with our protocols, enabling services to identify users who forward harmful media.

**Private Membership Test (PMT).** Our PMC problem formulation is a generalization of PMT, which is equivalent to client-revealing PEMC. We generalize by considering protocols for approximate matching, as well as protocols with server-revealing and server privacy properties.

PMTs have been studied extensively for malware detection: a client hashes an untrusted file (or obtains an application signature) and wishes to test membership of the value in a malware database held by an untrusted server. Unlike in our

We anticipate that significant performance gains are available for deployment.

[42]For 112-bit security and using no packing, the payload consists of 540K Paillier ciphertexts of 4096 bits each.

setting, some false positives are considered acceptable.

Tamrakar et al. use hardware security architectures to implement a scalable PMT for malware detection [79]. The work assumes a trusted execution environment with remote attestation and focuses on server-side optimizations like query batching. Recent side-channel attacks on ARM TrustZone and Intel SGX have decreased confidence in these architectures [80–82]. Our protocols do not rely on trusted hardware.

Ramezanian et al. propose a PMT for malware detection using Bloom and cuckoo filters encrypted under Paillier PHE [83]. The protocol requires low communication, but it induces false positives and compromises both client and server privacy. Clients reveal bits of $x$ and may learn about $\mathcal{B}$ from the filters.

PMTs have also been studied for compromised credential checking, where the Client tests membership of a password hash in a database of compromised credentials held by an untrusted server. Services like HaveIBeenPwned and Google Password Checkup provide APIs for this purpose [84, 85]. Current API implementations require a Client to reveal a prefix of a password's hash, violating the client privacy guarantee.

Li et al. show that knowledge of a hash prefix can increase guessing attack efficacy up to 12x [85]. The work proposes a frequency-smoothing bucketization method, which provides greater client privacy—but limits server privacy, by assuming Client knowledge of the relative frequency of hashes in $\mathcal{B}$.

Ali et al. construct a credential checking PMT from MulPIR, a cPIR protocol based on SHE [65]. The design guarantees computational client privacy and is practical for deployment.

**Private Biometric Authentication.** Yasuda et al. and Yasuda use SHE to build server-revealing Hamming distance protocols for biometric authentication, which inspire our PET design [44, 45]. Osadchy et al. use oblivious transfer (OT) and Paillier PHE to build a threshold protocol for face identification, which motivates our more efficient PTC design [69].

## 14 Conclusion

In this work, we explored the technical feasibility of privacy-preserving perceptual hash matching in E2EE services. We formalized the problem space, and we designed and evaluated protocols that optionally protect the hash set and optionally do not disclose matches to users. Our protocols are practical for deployment on modern mobile devices and introduce limited false negatives beyond those inherent in PHM.

As discussed in Section 2.3, our problem formulation and protocols have significant limitations. Future work could partially address those limitations and provide new capabilities. We briefly describe several promising directions.

**Improved Performance.** Recent work has improved on SealPIR, including in multi-query settings [65, 66]. There may be opportunities for performance gains in the cPIR step of our protocols, and it may be possible to reduce false negatives by querying for nearby LSH buckets in cPIR or combining the content of nearby buckets into each bucket.

**Increasing Trust in $\mathcal{B}$.** It may also be possible to increase (but not definitively establish) trust in $\mathcal{B}$. A server could, for example, publicly commit to LSH bucket contents (e.g., with a Merkle tree [86]) and each bucket could contain a proof of inclusion, giving users confidence that the server has not manipulated the bucket(s) returned by PIR or recently modified the hash set. The server could also collaborate with trusted third parties (e.g., civil society groups) to validate the hash set and additionally commit to LSH bucket contents.

**Zero-Knowledge Proofs.** A zero-knowledge proof of non-exact (PEMC) or non-approximate (PAMC) membership in $\mathcal{B}$, coupled with a zero-knowledge proof that $x$ is the perceptual hash of the same content encrypted under a session key (used for E2EE) could provide security against malicious Clients that spoof $x$. State-of-the-art constructions (via accumulators) do not yet support non-approximate membership, nor efficient non-exact membership while fully hiding $\mathcal{B}$ [87, 88].

**Harmful Media Disclosure.** While our constructions are agnostic to the content moderation action that the Server will take, one action that might be commonly required is disclosing media plaintext without notifying the client of a match. Future work could explore how to provide this property, such as by encrypting media with an ephemeral key and using a custom circuit for PTC that returns the key in the event of a match.

Additional directions for future work include private harmful media detection with machine learning, which is gaining traction (e.g., Google Content Safety [89]), and building protocols that protect the PHF.

In closing, we emphasize that the goal of this project is to contribute technical analysis about a *possible* path forward that maintains the benefits of E2EE communications while addressing the serious societal challenges posed by harmful media. We do not take a position on whether E2EE services should implement the protocols that we propose, and we have both technical and non-technical reservations ourselves. But we encourage the information security community to continue its earnest exploration of potential middle ground designs for storage and communications encryption that address tensions with longstanding societal priorities. There is value in understanding the space of possible designs and associated tradeoffs, even if the best option is to maintain the status quo.

## Acknowledgments

We are grateful to Seny Kamara, Hany Farid, and the Microsoft PhotoDNA team for valuable conversations that informed this work. We also thank Susan E. McGregor for thoughtful shepherding. All views are solely our own.

## References

[1] Ksenia Ermoshina, Francesca Musiani, and Harry Halpin. "End-to-End Encrypted Messaging Protocols: An Overview". In: *International Conference on Internet Science*. 2016.

[2] Apple. *Apple Platform Security*. Tech. rep. Apr. 2020. URL: `https://manuals.info.apple.com/MANUALS/1000/MA1902/en_US/apple-platform-security-guide.pdf`.

[3] WhatsApp. *WhatsApp Encryption Overview*. Tech. rep. Dec. 2017. URL: https://whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf.

[4] Facebook. *Messenger Secret Conversations*. Tech. rep. May 2017. URL: https://about.fb.com/wp-content/uploads/2016/07/messenger-secret-conversations-technical-whitepaper.pdf.

[5] Katriel Cohn-Gordon et al. "A Formal Security Analysis of the Signal Messaging Protocol". In: *IEEE European Symposium on Security and Privacy*. 2017.

[6] Facebook. *Community Standards Enforcement Report*. URL: https://transparency.facebook.com/community-standards-enforcement.

[7] Roshan Sumbaly et al. *Using AI to Detect COVID-19 Misinformation and Exploitative Content*. May 2020. URL: https://ai.facebook.com/blog/using-ai-to-detect-covid-19-misinformation-and-exploitative-content.

[8] Google. *YouTube Community Guidelines Enforcement*. URL: https://transparencyreport.google.com/youtube-policy/removals.

[9] Twitter. *Twitter Rules Enforcement*. URL: https://transparency.twitter.com/en/twitter-rules-enforcement.html.

[10] Microsoft. *New Technology Fights Child Porn by Tracking Its "PhotoDNA"*. URL: https://news.microsoft.com/2009/12/15/new-technology-fights-child-porn-by-tracking-its-photodna/.

[11] Bian Yang, Fan Gu, and Xiamu Niu. "Block Mean Value Based Image Perceptual Hashing". In: *International Conference on Intelligent Information Hiding and Multimedia*. 2006.

[12] Evan Klinger and David Starkweather. *pHash: The Open Source Perceptual Hash Library*. 2010. URL: https://www.phash.org.

[13] Neal Krawetz. *Looks Like It*. 2011. URL: https://www.hackerfactor.com/blog/index.php?/archives/432-Looks-Like-It.html.

[14] Christoph Zauner, Martin Steinebach, and Eckehard Hermann. "Rihamark: Perceptual Image Hash Benchmarking". In: *IS&T/SPIE Electronic Imaging*. 2011.

[15] Neal Krawetz. *Kind of Like That*. 2013. URL: https://www.hackerfactor.com/blog/index.php?/archives/529-Kind-of-Like-That.html.

[16] Arambam Neelima and Kh Manglem Singh. "Perceptual Hash Function Based on Scale-Invariant Feature Transform and Singular Value Decomposition". In: *The Computer Journal* 59.9 (2016).

[17] Dmitry Petrov. *Wavelet Image Hash in Python*. 2016. URL: https://fullstackml.com/wavelet-image-hash-in-python-3504fdd282b5.

[18] Facebook. *The TMK+PDQF Video-Hashing Algorithm and the PDQ Image-Hashing Algorithm*. Tech. rep. Aug. 2019. URL: https://github.com/facebook/ThreatExchange/blob/master/hashing/hashing.pdf.

[19] Microsoft. *Microsoft PhotoDNA Cloud Service*. 2009. URL: https://microsoft.com/en-us/photodna.

[20] John F. Clark. *Statement for the United States Senate Committee on the Judiciary*. July 2019. URL: https://www.judiciary.senate.gov/imo/media/doc/Clark%20Testimony.pdf.

[21] Elie Bursztein et al. "Rethinking the Detection of Child Sexual Abuse Imagery on the Internet". In: *The World Wide Web Conference*. 2019.

[22] Internet Watch Foundation. *Hash List*. URL: https://www.iwf.org.uk/our-services/hash-list.

[23] Canadian Centre for Child Protection. *Project Arachnid*. URL: https://projectarachnid.ca/en/#how-does-it-work.

[24] Global Internet Forum to Counter Terrorism. *Joint Tech Innovation*. URL: https://www.gifct.org/joint-tech-innovation/.

[25] Priti Patel et al. *Open Letter: Facebook's "Privacy First" Proposals*. Oct. 2019. URL: https://www.justice.gov/opa/press-release/file/1207081/download.

[26] National Center for Missing and Exploited Children. *NCMEC's Statement Regarding End-to-End Encryption*. Oct. 2019. URL: https://missingkids.org/blog/2019/post-update/end-to-end-encryption.

[27] Priti Patel et al. *International Statement: End-to-End Encryption and Public Safety*. Oct. 2020. URL: https://www.justice.gov/opa/pr/international-statement-end-end-encryption-and-public-safety.

[28] Piotr Indyk and Rajeev Motwani. "Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality". In: *ACM Symposium on Theory of Computing*. 1998.

[29] Moses S. Charikar. "Similarity Estimation Techniques from Rounding Algorithms". In: *ACM Symposium on Theory of Computing*. 2002.

[30] Eric Goldman. *Content Moderation Remedies*. Apr. 2019. URL: https://citp.princeton.edu/event/goldman/.

[31] Baris Coskun and Nasir Memon. "Confusion/Diffusion Capabilities of Some Robust Hash Functions". In: *Conference on Information Sciences and Systems*. 2006.

[32] Jeffrey Knockel, Lotus Ruan, and Masashi Crete-Nishihata. "An Analysis of Automatic Image Filtering on WeChat Moments". In: *USENIX Workshop on Free and Open Communications on the Internet*. 2018.

[33] Janis Dalins, Campbell Wilson, and Douglas Boudry. *PDQ & TMK + PDQF – A Test Drive of Facebook's Perceptual Hashing Algorithms*. 2019. arXiv: 1912.07745 [cs.CV].

[34] Ray Ozzie. *CLEAR*. Tech. rep. Jan. 2017. URL: https://github.com/rayozzie/clear/blob/master/clear-rozzie.pdf.

[35] Steven M. Bellovin et al. *Analysis of the CLEAR Protocol per the National Academies' Framework*. Tech. rep. CUCS-003-18. Department of Computer Science, Columbia University, May 2018. URL: https://mice.cs.columbia.edu/getTechreport.php?techreportID=1637.

[36] Stefan Savage. "Lawful Device Access Without Mass Surveillance Risk: A Technical Design Discussion". In: *ACM Conference on Computer and Communications Security*. 2018.

[37] Charles V. Wright and Mayank Varia. "Crypto Crumple Zones: Enabling Limited Access without Mass Surveillance". In: *IEEE European Symposium on Security and Privacy*. 2018.

[38] Charles V. Wright and Mayank Varia. "A Cryptographic Airbag for Metadata: Protecting Business Records Against Unlimited Search and Seizure". In: *USENIX Workshop on Free and Open Communications on the Internet*. 2018.

[39] Zvika Brakerski and Vinod Vaikuntanathan. "Fully Homomorphic Encryption from Ring-LWE and Security for Key Dependent Messages". In: *International Cryptology Conference*. 2011.

[40] Junfeng Fan and Frederik Vercauteren. *Somewhat Practical Fully Homomorphic Encryption*. Cryptology ePrint Archive, Report 2012/144. 2012.

[41] Christian Mouchet, Juan Troncoso-Pastoriza, and Jean-Pierre Hubaux. *Computing Across Trust Boundaries Using Distributed Homomorphic Cryptography*. Cryptology ePrint Archive, Report 2019/961. 2019.

[42] Hao Chen et al. *Efficient Homomorphic Conversion Between (Ring) LWE Ciphertexts*. Cryptology ePrint Archive, Report 2020/015. 2020.

[43] Ivan Damgård et al. “Multiparty Computation from Somewhat Homomorphic Encryption”. In: *International Cryptology Conference*. 2012.

[44] Masaya Yasuda et al. “New Packing Method in Somewhat Homomorphic Encryption and Its Applications”. In: *Security and Communication Networks* 8.13 (2015).

[45] Masaya Yasuda. “Secure Hamming Distance Computation for Biometrics Using Ideal-Lattice and Ring-LWE Homomorphic Encryption”. In: *Information Security Journal: A Global Perspective* 26.2 (2017).

[46] Taher ElGamal. “A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms”. In: *IEEE Transactions on Information Theory* 31.4 (1985).

[47] Neal Koblitz. “Elliptic Curve Cryptosystems”. In: *Mathematics of Computation* 48.177 (1987).

[48] Andrew C. Yao. “Protocols for Secure Computations”. In: *IEEE Symposium on Foundations of Computer Science*. IEEE. 1982.

[49] Meng Liu et al. “Efficient Solution to the Millionaires’ Problem Based on Asymmetric Commutative Encryption Scheme”. In: *Computational Intelligence* 35.3 (2019).

[50] Ivan Damgård, Martin Geisler, and Mikkel Kroigard. “Homomorphic Encryption and Secure Comparison”. In: *International Journal of Applied Cryptography* 1.1 (2008).

[51] Tsung-Yi Lin et al. “Microsoft COCO: Common Objects in Context”. In: *European Conference on Computer Vision*. 2014.

[52] Rasmus Rothe, Radu Timofte, and Luc Van Gool. “DEX: Deep EXpectation of Apparent Age from a Single Image”. In: *IEEE International Conference on Computer Vision Workshops*. 2015.

[53] Lucia Vadicamo et al. “Cross-Media Learning for Image Sentiment Analysis in the Wild”. In: *IEEE International Conference on Computer Vision Workshops*. 2017.

[54] Baoyuan Wu et al. “Tencent ML-Images: A Large-Scale Multi-Label Image Database for Visual Representation Learning”. In: *IEEE Access* 7 (2019).

[55] Raul Gomez et al. “Learning to Learn from Web Data Through Deep Semantic Embeddings”. In: *European Conference on Computer Vision*. 2018.

[56] Anunay Kulshrestha and Jonathan Mayer. *Identifying Harmful Media in End-to-End Encrypted Communication: Efficient Private Membership Computation (Supplementary Information)*. 2020. URL: `https://github.com/citp/private-membership-computation`.

[57] Shunsuke Kanda, Kazuhiro Morita, and Masao Fuketa. “Compressed Double-Array Tries for String Dictionaries Supporting Fast Lookup”. In: *Knowledge and Information Systems* 51.3 (2017).

[58] Arash Partow. *C++ Bloom Filter Library*. 2000. URL: `http://www.partow.net/programming/bloomfilter/index.html`.

[59] Bin Fan et al. “Cuckoo Filter: Practically Better than Bloom”. In: *ACM International Conference on Emerging Networking Experiments and Technologies*. 2014.

[60] Mohammad Norouzi, Ali Punjani, and David J. Fleet. “Fast Search in Hamming Space with Multi-Index Hashing”. In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2012.

[61] Burton H. Bloom. “Space/Time Trade-Offs in Hash Coding with Allowable Errors”. In: *Communications of the ACM* 13.7 (1970).

[62] Benny Chor et al. “Private Information Retrieval”. In: *IEEE Annual Symposium on Foundations of Computer Science*. 1995.

[63] Julien P. Stern. “A New and Efficient All-or-Nothing Disclosure of Secrets Protocol”. In: *International Conference on the Theory and Application of Cryptology and Information Security*. 1998.

[64] Helger Lipmaa. “First CPIR Protocol with Data-Dependent Computation”. In: *International Conference on Information Security and Cryptology*. 2009.

[65] Asra Ali et al. *Communication–Computation Trade-Offs in PIR*. Cryptology ePrint Archive, Report 2019/1483. 2019.

[66] Sebastian Angel et al. “PIR with Compressed Queries and Amortized Query Processing”. In: *IEEE Symposium on Security and Privacy*. 2018.

[67] Carlos Aguilar-Melchor et al. “XPIR: Private Information Retrieval for Everyone”. In: *Proceedings on Privacy Enhancing Technologies* 2016.2 (2016).

[68] Martin R. Albrecht, Rachel Player, and Sam Scott. “On the Concrete Hardness of Learning with Errors”. In: *Journal of Mathematical Cryptology* 9.3 (2015).

[69] Margarita Osadchy et al. “SCiFI – A System for Secure Face Identification”. In: *IEEE Symposium on Security and Privacy*. 2010.

[70] Hao Chen, Kim Laine, and Rachel Player. “Simple Encrypted Arithmetic Library - SEAL v2.1”. In: *International Conference on Financial Cryptography and Data Security*. 2017.

[71] Raphael R. Toledo, George Danezis, and Ian Goldberg. “Lower-Cost $\varepsilon$-Private Information Retrieval”. In: *Proceedings on Privacy Enhancing Technologies* 2016.4 (2016).

[72] Hsiao-Ying Lin and Wen-Guey Tzeng. “An Efficient Solution to the Millionaires’ Problem Based on Homomorphic Encryption”. In: *International Conference on Applied Cryptography and Network Security*. 2005.

[73] Priyanka Singh and Hany Farid. “Robust Homomorphic Image Hashing”. In: *IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2019.

[74] Paul Grubbs, Jiahui Lu, and Thomas Ristenpart. *Message Franking via Committing Authenticated Encryption*. Cryptology ePrint Archive, Report 2017/664. 2017.

[75] Yevgeniy Dodis et al. “Fast Message Franking: From Invisible Salamanders to Encryptment”. In: *International Cryptology Conference*. 2018.

[76] Nirvan Tyagi et al. “Asymmetric Message Franking: Content Moderation for Metadata-Private End-to-End Encryption”. In: *International Cryptology Conference*. 2019.

[77] Jonathan Mayer. *Content Moderation for End-to-End Encrypted Messaging*. Oct. 2019. URL: `https://www.cs.princeton.edu/~jrmayer/papers/Content_Moderation_for_End-to-End_Encrypted_Messaging.pdf`.

[78] Nirvan Tyagi, Ian Miers, and Thomas Ristenpart. “Traceback for End-to-End Encrypted Messaging”. In: *ACM Conference on Computer and Communications Security*. 2019.

[79] Sandeep Tamrakar et al. “The Circle Game: Scalable Private Membership Test Using Trusted Hardware”. In: *ACM Asia Conference on Computer and Communications Security*. 2017.

[80] Johannes Götzfried et al. “Cache Attacks on Intel SGX”. In: *European Workshop on Systems Security*. 2017.

[81] Jo Van Bulck et al. “Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution”. In: *USENIX Security Symposium*. 2018.

[82] Nico Weichbrodt et al. “AsyncShock: Exploiting Synchronisation Bugs in Intel SGX Enclaves”. In: *European Symposium on Research in Computer Security*. 2016.

[83] Sara Ramezanian et al. “Private Membership Test Protocol with Low Communication Complexity”. In: *Digital Communications and Networks* (2019).

[84] Kurt Thomas et al. "Protecting Accounts from Credential Stuffing with Password Breach Alerting". In: *USENIX Security Symposium*. 2019.

[85] Lucy Li et al. "Protocols for Checking Compromised Credentials". In: *ACM Conference on Computer and Communications Security*. 2019.

[86] Ralph C. Merkle. "A Digital Signature Based on a Conventional Encryption Function". In: *International Conference on the Theory and Applications of Cryptographic Techniques*. 1987.

[87] Foteini Badimtsi, Ran Canetti, and Sophia Yakoubov. "Universally Composable Accumulators". In: *Cryptographers' Track at the RSA Conference*. 2020.

[88] David Derler, Christian Hanser, and Daniel Slamanig. "Revisiting Cryptographic Accumulators, Additional Properties and Relations to Other Primitives". In: *Cryptographers' Track at the RSA Conference*. 2015.

[89] Nikola Todorovic and Abhi Chaudhuri. *Using AI to Help Organizations Detect and Report Child Sexual Abuse Material Online*. Sept. 2018. URL: `https://www.blog.google/around-the-globe/google-europe/using-ai-help-organizations-detect-and-report-child-sexual-abuse-material-online/`.

# Appendices

## A Security Models for PHM and PMC

PHM systems are fundamentally limited to semi-honest security, because they depend on perceptual hashing of media content. Parties could collude to conceal media content or take advantage of how PHFs are imperfect approximations of perceptual similarity. We describe two scenarios in which PHM inherently cannot guarantee malicious security.

**Two-Party Malicious Security.** The parties (Sender and Recipient) agree out-of-band to defeat PHM. For example, the Sender encrypts $\mathsf{media}$ as $E(\mathsf{media})$. Encryption-induced pseudorandomness eliminates perceptual similarity, such that with high probability $d_H(\mathsf{PHF}(\mathsf{media}), \mathsf{PHF}(E(\mathsf{media}))) > \delta_H$ and PHM detection fails. Recipient receives $E(\mathsf{media})$ and decrypts, obtaining $\mathsf{media}$.

**One-Sided Malicious Security: Perceptual Manipulation.** Sender manipulates $\mathsf{media}$ to create $\mathsf{media}' = \{\mathsf{media}'_1, \ldots, \mathsf{media}'_n\}$, content elements that are collectively perceptually similar to $\mathsf{media}$ but $\forall \mathsf{media}'_i \in \mathsf{media}' : d_H(\mathsf{PHF}(\mathsf{media}), \mathsf{PHF}(\mathsf{media}'_i)) > \delta_H$. This property defeats PHM detection by definition.

The PMC constructions that we develop build on PHM and have these same limitations. There is, however, a specific type of one-sided malicious security that we *can* guarantee in our PMC protocols.

**One-Sided Malicious Security: PMC Cheating.** The Sender participates in PMC and cheats at some stage(s). For example, the Sender participates with spoofed content $\mathsf{media}'$, then sends $\mathsf{media}$ to an honest Recipient.

BFV Cryptosystem

**Public Parameters:** Moduli $q, t$, degree $n$, standard deviation $\sigma$, key space $R_3$, polynomial ring $R_q$, and decomposition basis $w < q$.

FHE.KeyGen. Sample $\mathsf{sk} \leftarrow_\$ R_3$, $e \leftarrow_\$ \chi$ and $p_1 \leftarrow_\$ R_q$.
Set $p_0 = -p_1 \cdot \mathsf{sk} + te$ and $\mathsf{pk} = (p_0, p_1)$. Output $(\mathsf{sk}, \mathsf{pk})$.

FHE.RlzKeyGen$(\mathsf{sk}, w)$. For $l = \log_w(q)$, sample $r_1 \leftarrow_\$ (R_q)^l$, $e \leftarrow_\$ \chi^l$. Output $(\mathsf{sk}^2 w - \mathsf{sk} r_1 + e, r_1)$

FHE.Enc$(\mathsf{pk}, m)$. Sample $u \leftarrow_\$ R_3$, $f, g \leftarrow_\$ \chi$. If $\mathsf{pk} = (p_0, p_1)$, output $(p_0 u + tg + m, p_1 u + tf) \in (R_q)^2$.

FHE.Dec$(\mathsf{sk}, c)$. If $c = (c_0, c_1)$, output $[\lfloor \frac{t}{q}[c_0 + c_1 \mathsf{sk}]_q \rceil]_t \in R_t$.

FHE.Add$(c, c')$. If $c = (c_0, c_1), c' = (c'_0, c'_1)$, output $(c_0 + c'_0, c_1 + c'_1)$.

FHE.Mul$(c, c')$. If $c = (c_0, c_1), c' = (c'_0, c'_1)$, output $[\lfloor \frac{t}{q}(c_0 c'_0, c_0 c'_1 + c_1 c'_0, c_1 c'_1) \rceil]_q$.

FHE.Sub(FHE.Enc$(\mathsf{pk}, p(x)), k)$ = FHE.Enc$(\mathsf{pk}, p(x^k))$ [66]

FHE.Absorb$(pk, m,$ FHE.Enc$(\mathsf{pk}, m'))$ = FHE.Enc$(\mathsf{pk}, m \cdot m')$ [66]

Figure 4: The BFV cryptosystem [39, 40].

We can provide one-sided malicious security for this scenario: The honest Recipient could participate in PMC, catching the Sender's cheating. Alternatively, the Sender could provide an authenticated PMC transcript for the Recipient to validate.

## B Primitives

We rely on the BFV cryptosystem in both our PEMC and PAMC constructions, we use the ElGamal cryptosystem in our PEMC protocol, and we use privacy-preserving comparison for our PAMC design. We expand here on the explanation of these primitives in Section 3.2.

**Brakerski/Fan-Vercauteren (BFV) Cryptosystem.** Recall that BFV plaintexts are polynomials from the quotient ring $\mathbb{Z}[X]/(X^n + 1)$ with plaintext coefficients modulo $t$ for a given modulus $t$ and $n$, a power of 2. Suppose $q$ is the ciphertext coefficient modulus with $q \equiv 1 \pmod{2n}$, and denote $R_q = \mathbb{Z}_q[X]/(X^n + 1)$ as the quotient ring of univariate polynomials with coefficients in $\mathbb{Z}_q$. BFV ciphertexts are $\in (R_q)^2$. Similarly, plaintexts are elements of $R_t = \mathbb{Z}_t[X]/(X^n + 1)$. We also fix a parameter $\sigma$ that defines a discrete Gaussian error distribution $\chi$. Every element of $\chi$ is a polynomial of degree less than $n$ with coefficients drawn from $\mathcal{N}(0, \sigma)$ and rounded to the nearest integer. Secret keys are chosen from the quotient ring $R_3 = \mathbb{Z}_3[X]/(X^n + 1)$ such that elements of $R_3$ are univariate polynomials with coefficients uniformly sampled from $\{-1, 0, 1\}$. The scheme is outlined in Figure 4.[43] Note that $[\cdot]_q$ returns a value in $[\frac{q}{2}, \frac{q}{2})$.

Addition and multiplication of ciphertexts is possible in the BFV scheme. Addition is cheap and is performed coefficient-wise. Multiplication of ciphertexts results in a 3-tuple $\in$

[43]We omit certain functions, including relinearization, for brevity.

Two-Party BFV Cryptosystem

**Public Parameters:** Moduli $q,t$, degree $n$, standard deviation $\sigma$, and key space $R_3$.

FHE.KeyAgg($p_1$)

Parties generate respective secret keys $\mathsf{sk}_c$ and $\mathsf{sk}_s$.
Client samples $e_c \leftarrow_{\$} \chi$ and sends $p_{0,c} = [-(p_1 \mathsf{sk}_c + e_c)]_q$.
Server samples $e_s \leftarrow_{\$} \chi$, computes $p_{0,s} = [-(p_1 \mathsf{sk}_s + e_s)]_q$ and aggregated public key $([p_{0,c} + p_{0,s}]_q, p_1)$.

FHE.CKS($c,\sigma_c$)

Suppose $c = (c_0, c_1)$ and Receiver knows new secret key $\mathsf{sk}'$.
Sender and Receiver have secret shares $\mathsf{sk}_s$ and $\mathsf{sk}_r$ respectively.
Sender samples $e_s \leftarrow_{\$} \chi_{CKS}(\sigma_c)$ and sends $h_s = [(\mathsf{sk}_s * c_1) + e_s]_q$.
Receiver samples $e_r \leftarrow_{\$} \chi_{CKS}(\sigma_c)$, then computes $h_r = [((\mathsf{sk}_r - \mathsf{sk}') * c_1) + e_r]_q$ and key-switched ciphertext $c' = ([c_0 + h_s + h_r]_q, c_1)$.

FHE.GKG($K_1,w,\tau^m$)

Suppose $g = \lfloor log_w(q) \rfloor$, $K_1 \in R_q^g$, $W = (1,\ldots,w^{g-1})$ and $\tau^m$ is a Galois element. Client and Server have secret shares $\mathsf{sk}_c, \mathsf{sk}_s$.
Client sets $e_c \leftarrow_{\$} \chi^g$, sends $K_{0,c} = [-\mathsf{sk}_c \cdot K_1 + \tau^m(\mathsf{sk}_c) \cdot W + e_c]_q$
Server sets $e_c \leftarrow_{\$} \chi^g$, $K_{0,s} = [-\mathsf{sk}_s \cdot K_1 + \tau^m(\mathsf{sk}_s) \cdot W + e_s]_q$
Server computes collective keys $(K_0, K_1)$ as $K_0 = K_{0,c} + K_{0,s}$.

Figure 5: A two-party variant of the BFV cryptosystem from Mouchet et al. [41]. Refer to the original text for a description of the distribution $\chi_{CKS}$.

$(R_q)^3$, which can be relinearized to a 2-tuple $\in (R_q)^2$ using relinearization keys generated by FHE.RlzKeyGen [40, 41].

In FHE.KeyAgg, the polynomial $p_1$ is public. FHE.CKS takes as input a ciphertext $c = \mathsf{FHE.Enc}(\mathsf{pk}_{\text{agg}}, m)$ encrypted under the aggregated public key $\mathsf{pk}_{\text{agg}}$ with noise parameter $\sigma_c$. The protocol outputs a ciphertext such that $\mathsf{FHE.Dec}(\mathsf{sk}_s + \mathsf{sk}_c, c) = \mathsf{FHE.Dec}(\mathsf{sk}', c')$. FHE.GKG can be used to generated collective Galois keys. The public parameters include $d$ polynomials $K_1$, a decomposition basis $w < q$, and Galois element $\tau^m$.

**ElGamal Cryptosystem.** Suppose $q > 2^k$ is prime and $E$ is an elliptic curve over $\mathbb{F}_q$. We fix a point $G$ on $E$ and consider the group $\mathbb{G}$ generated by $G$. ElGamal plaintexts are integers modulo $q$ and ciphertexts are elements of $\mathbb{G}^2$. Figure 6 outlines the overall scheme. Decryption of a ciphertext in this scheme is equivalent to solving the elliptic curve discrete log problem (ECDLP). This inefficiency in decryption does not affect our ability to test whether a given ciphertext decrypts to $m$, which is sufficient for our protocol.

**Privacy-Preserving Comparison.** For a $k$-bit string $x = x_k x_{k-1} \ldots x_1 \in \{0,1\}^k$, the 0-encoding $S_x^0$ and the 1-encoding

Elliptic Curve ElGamal Cryptosystem

**Public Parameters:** Prime $q$, elliptic curve $E$, and generator $G$.
PHE.KeyGen. Set $\mathsf{sk} \leftarrow_{\$} \mathbb{Z}_q^*$, output $(\mathsf{sk}, \mathsf{pk} = \mathsf{sk} \cdot G)$.
PHE.Enc($\mathsf{pk}, m$). Set $r \leftarrow_{\$} \mathbb{Z}_q^*$, output $(r \cdot G, m \cdot G + r \cdot \mathsf{pk})$.
PHE.DecChk($\mathsf{sk}, (C, C'), m$). Output $\mathsf{sk} \cdot C \overset{?}{=} C' - m \cdot G$.
PHE.Add($(C_1, C_1'), (C_2, C_2')$). Output $(C_1 + C_2, C_1' + C_2')$.
PHE.SMul($s, (C, C')$). Output $(s \cdot C, s \cdot C')$.

Figure 6: An elliptic curve variant of the ElGamal cryptosystem [46, 47].

$S_x^1$ of $x$ are

$$S_x^0 = \{x_k x_{k-1} \ldots x_{i+1} \,\|\, 1 \,\|\, 0^{i-1} : x_i = 0, 1 \le i \le n\}$$
$$S_x^1 = \{x_k x_{k-1} \ldots x_i \,\|\, 0^{i-1} : x_i = 1, 1 \le i \le n\}$$

where $0^i$ is the $i$-bit string of zeros. Lin and Tzeng previously showed that $x > y \iff |S_x^1 \cap S_y^0| = 1$ and transform PPC into a private set intersection (PSI) problem [72]. Liu et al. design an asymmetric commutative encryption scheme to efficiently solve this PSI reduction. We fix an elliptic curve $E$ over $\mathbb{F}_q$ where $q$ is prime. Choose a base point $G$ on $E(\mathbb{F}_q)$ and consider the group $\mathbb{G}$ generated by $G$. We define $\mathsf{CE.Enc} : \mathbb{F}_q \times \mathbb{G} \to \mathbb{G}$ as $\mathsf{CE.Enc}(\mathsf{sk}, M) = \mathsf{sk} \cdot M$. Notice that for $\mathsf{sk}_1, \mathsf{sk}_2$, and $M \in \mathbb{G}$, the encryption scheme is commutative: $\mathsf{CE.Enc}(\mathsf{sk}_1, \mathsf{CE.Enc}(\mathsf{sk}_2, M)) = \mathsf{sk}_1 \cdot (\mathsf{sk}_2 \cdot M) = \mathsf{sk}_2 \cdot (\mathsf{sk}_1 \cdot M) = \mathsf{CE.Enc}(\mathsf{sk}_2, \mathsf{CE.Enc}(\mathsf{sk}_1, M))$. Furthermore, for a fixed secret key $\mathsf{sk} \in \mathbb{F}_q$, the encryption function $\mathsf{CE.Enc}(\mathsf{sk}, \cdot)$ is bijective: for $M_1, M_2 \in \mathbb{G}$, $\mathsf{sk} \cdot M_1 = \mathsf{sk} \cdot M_2 \implies \log_G(\mathsf{sk} \cdot M_1) = \log_G(\mathsf{sk} \cdot M_2) \implies \log_G(M_1) = log_G(M_2) \implies M_1 = M_2$ and for $C \in \mathbb{G}$, $\mathsf{CE.Enc}(\mathsf{sk}, G^{\log_G(C) - \log_G(\mathsf{sk})}) = C$. Liu et. al use this encryption scheme to build an efficient PSI protocol that operates over 0- and 1-encodings of PPC inputs.[44] We defer a proof of correctness and security (in the semi-honest model) of the PPC protocol to the original text [49].

## C Private Approximate Equality Test

We provide two lemmas associated with the PAET step of our PAMC protocol.

**Lemma C.1.** Suppose $(\mathsf{sk}, \mathsf{pk})$ is a BFV key pair. For all $(a,b) \in \{0,1\}^{l_a} \times \{0,1\}^{l_b}$, we define $c_a$ and $c_b$ as

$$c_a = \mathsf{FHE.Enc}(\mathsf{sk}, \mathsf{FHE.Pack}_1(a))$$
$$c_b = \mathsf{FHE.Enc}(\mathsf{sk}, \mathsf{FHE.Pack}_2(b)).$$

Let $\zeta(c_a, c_b) = -2^{-1}\{(2 \cdot c_a - J_a) \cdot (2 \cdot c_b - J_b)\} + 2^{-1}\{J_a \cdot J_b\}$ where we define $J_a, J_b \in R_q$ as

$$J_a = \sum_{i=0}^{l_a - 1} x^i \qquad J_b = -\sum_{i=0}^{l_b - 1} x^{n-i}.$$

[44]We present an elliptic curve variant of the PSI-based PPC protocol in the Supplementary Information [56].

Table 10: The transformation ranges that we use in our evaluation of PHF predictive performance. For a given transform type and level, we sample uniformly at random from the range.

| *Transform* | *Level 1* | *Level 2* | *Level 3* |
|---|---|---|---|
| Rotation ($\Delta_r^\circ$) | $[0,5]$ | $[5,15]$ | $[15,30]$ |
| Noise Addition ($\sigma$) | $[0,2.5]$ | $[2.5,7.5]$ | $[7.5,17.5]$ |
| Cropping ($\Delta_c$) | $[0.9,1.0]$ | $[0.75,0.9]$ | $[0.55,0.75]$ |
| Gamma Correction ($\gamma$) | $[-0.1,0.1]$ | $[-0.25,-0.1] \cup [0.1,0.25]$ | $[-0.45,-0.25] \cup [0.25,0.45]$ |
| Rescaling ($\Delta_a$) | $[0.9,1] \cup [1,\frac{1}{0.9}]$ | $[0.75,0.9] \cup [\frac{1}{0.9},\frac{1}{0.75}]$ | $[0.55,0.75] \cup [\frac{1}{0.75},\frac{1}{0.55}]$ |

The polynomial $p_{a,b} \in R_t$, defined as

$$p_{a,b} = \mathsf{FHE.Dec}(\mathsf{sk}, \zeta(c_a, c_b)) = \sum_{i}^{n-1} p_i x^i$$

yields Hamming distances

$$p_i \equiv d_H(a^{(i)}, b) \pmod t$$

for $i \in [l_a - l_b + 1]$ where $a^{(i)}$ is the bit string $a_0 a_1 \cdots a_{i+l_b}$.

**Lemma C.2.** Suppose $(\mathsf{sk}, \mathsf{pk}) = \mathsf{FHE.Keygen}(q,t,n,\sigma)$ is a BFV key pair. For $i \in [\frac{n}{k}]$, all $x, y_i \in \{0,1\}^k$, we define ciphertexts $c_y$ and $c_{x_i}$ as

$$c_y = \mathsf{FHE.Enc}(\mathsf{sk}, \mathsf{FHE.Pack}_1(y_0 \,\|\, \cdots \,\|\, y_{\frac{n}{k}-1}))$$
$$c_{x_i} = \mathsf{FHE.Enc}(\mathsf{sk}, \mathsf{FHE.Pack}_2(0^k_1 \,\|\, \cdots \,\|\, 0^k_{i-1} \,\|\, x)).$$

We define the polynomials $p^{(i)} \in R_t$ for $i \in [\frac{n}{k}]$ as

$$p^{(i)} = \mathsf{FHE.Dec}(\mathsf{sk}, \zeta(c_y, c_{x_i})) = \sum_{j=0}^{n-1} p_j^{(i)} x^j,$$

yielding Hamming distances

$$p^{(i)}(0) = \left[\sum_{j=0}^{i-1} \|y_j\|_H\right] + d_H(y_i, x)$$

where $\|y_i\|_H = d_H(y_i, 0^k)$ is the Hamming norm of string $y_i$.

*Proof.* From Lemma C.1, we know that

$$\begin{aligned} p^{(i)}(0) &= d_H(y_0 \,\|\, \cdots \,\|\, y_{\frac{n}{k}-1}, 0^k_0 \,\|\, \cdots \,\|\, 0^k_{i-1} \,\|\, x) \\ &= \left[\sum_{j=0}^{i-1} d_H(y_j, 0^k)\right] + d_H(y_i, x) \\ &= \left[\sum_{j=0}^{i-1} \|y_j\|_H\right] + d_H(y_i, x) \end{aligned}$$ ■

## D PHF Predictive Performance

Table 10 provides the transform ranges that we use for evaluating PHF predictive performance in Section 4. We describe the construction of each PHF that we evaluate below. Recall that the PHFs operate on resized and grayscale images.

- Average Hash (aHash): Compares each pixel intensity (i.e., RGB mean) to the mean pixel intensity [13].
- blockHash: Partitions the image into blocks and compares each pixel intensity to the mean intensity in its block [11].
- Difference Hash (dHash): Compares each pixel intensity to neighboring pixel intensity [15].
- Facebook PDQ: Computes a weighted average of luminance using two-pass Jarosz filters, performs a discrete cosine transform (DCT), then compares each value in frequency space to the median value [18].
- pHash: Performs a DCT over pixel intensities, then compares frequency space values similar to PDQ [12].
- Wavelet Hash (wHash): Performs a discrete wavelet transform over pixel intensities, then compares frequency space values similar to PDQ [17].

## E Client-Revealing Protocols

The PEMC and PAMC constructions in Section 11 are server-revealing protocols. We describe how to modify our PEMC design so that it is both client- and server-revealing, then we describe how to modify our PAMC design so that it is only client-revealing. Recall that PEMC is equivalent to PAMC with $\delta_H = 0$, so our client-revealing PAMC construction can also function as a client-revealing PEMC protocol.

**Client- and Server-Revealing PEMC.** Adapting the PEMC protocol that we present in the main text to be both client- and server-revealing is trivial: the Server can relay the output of the protocol to the Client. This construction requires further trust in the Server, however, which could misrepresent the protocol output. In order to mitigate that risk, the Server could send a zero-knowledge argument that $\rho_{-x}(C_y, C'_y)$ decrypts to 0 (Section 9.1). If $\rho_{-x}(C_y, C'_y)$ decrypts to 0, we must have $C'_y = \mathsf{sk}_s \cdot C_y$. Only the Server knows $\mathsf{sk}_s = \log_{C_y}(C'_y)$, and the Server can construct a generalized Schnorr proof of knowledge of this discrete log. The Server can make a non-interactive argument in the random oracle model using the Fiat-Shamir heuristic. The additional cost of generating the proof is 2 EC point multiplications for the Server, and verification costs 1 multiplication for the Client. The additional communication cost is only 2 EC points for each equality test.

**Client-Revealing PAMC.** The Client and the Server can run a client-revealing variant of the PPC protocol (Section 3.2 and Appendix B), where the Client learns whether $d_H(x,y) \leq \delta_H$ (Section 10.1). This change results in a client-revealing PAMC protocol. As above, this design introducing an additional opportunity for Server cheating—the Server could misrepresent its PPC input (i.e., a very large or very small value) to induce a preferred PAMC output. We can mitigate this risk by modifying the PTC step to prove that $0 \leq p_{x,y}(0) - r(0) \leq \delta_H$. This property is achievable by using two PPC instances for each PTC, where the Server uses the same input for both instances and provide a NIZK argument that the inputs are identical.